CONTRIBUTIONS À LA CONCEPTION D'UN SYSTÈME OPÉRATIONNEL DE PLANIFICATION DE TRAJECTOIRES EN TEMPS RÉEL POUR LES DRONES

CONTRIBUTIONS TO THE CONCEPTION OF AN OPERATIONAL SYSTEM FOR REAL TIME PATH PLANNING FOR UNMANNED AERIAL VEHICLES

Une thèse soumise à la

Division des études supérieures et de la recherche
du Collège militaire royal du Canada

par

Vincent Rémi Roberge
Capitaine

en vue de l'obtention du diplôme de

Maîtrise ès sciences appliquées en Génie informatique

Avril 2011

Your file Votre référence
ISBN: 978-0-494-82204-3
Our file Notre référence
ISBN: 978-0-494-82204-3

Résumé

Roberge, Vincent Rémi. M.Sc.A. Collège militaire royal du Canada, avril 2011. *Contributions à la conception d'un système opérationnel de planification de trajectoires en temps réel pour les drones.* Thèse dirigée par M. G. Labonté, Ph.D., M. M. Tarbouchi, Ph.D.

Le développement de drones autonomes est d'un grand intérêt pour plusieurs organisations gouvernementales et militaires à travers le monde. Un aspect essentiel de l'autonomie d'un robot est son aptitude à planifier automatiquement sa trajectoire. Les solutions proposées antérieurement utilisent des modèles souvent trop simplistes pour être intégrées à un système réel. En outre, les approches qui tiennent compte de la dynamique du drone et de la complexité d'un environnement réaliste demandent habituellement une puissance de calcul trop grande pour être employées en temps réel. Dans ce mémoire, nous utilisons deux algorithmes d'optimisation non déterministes, l'algorithme génétique (AG) et l'algorithme d'optimisation par essaim de particules (OEP), pour surpasser cette complexité et calculer des trajectoires faisables et quasi optimales dans un environnement 3D, tout en respectant les caractéristiques dynamiques du véhicule aérien. Les qualités d'une route optimale sont représentées sous forme d'une fonction de coût à objectifs multiples et la trajectoire générée est composée de segments de droites et d'arcs de cercle. De plus, nous réduisons le temps de calcul de nos algorithmes en développant des versions parallèles suivant le paradigme « instruction unique, données multiples » et nous atteignons une performance temps réel sur un microprocesseur multicœur. Finalement, nous interfaçons nos algorithmes de planification à un pilote automatique commercial MicroPilot qui suit avec succès une trajectoire générée en vol d'essai. Après avoir obtenu une accélération quasi-linéaire de plus de 7.3 sur 8 cœurs et un temps d'exécution de 10 s pour chacun des algorithmes, nous concluons que la planification de trajectoires pour les drones en temps réel est possible grâce à une implémentation parallèle sur processeur multicœur. De plus, notre comparaison statistiquement significative révèle que l'AG est supérieur à l'OEP pour la planification de trajectoires pour les drones.

Mots-clés : drone, planification de trajectoires, algorithme génétique, optimisation par essaim de particules, programmation parallèle, pilote automatique, vol d'essai.

Abstract

Roberge, Vincent. M.A.Sc. Royal Military College of Canada, April 2011. *Contributions to the conception of an Operational System for Real Time Path Planning for Unmanned Aerial Vehicles.* Supervised by Dr. G. Labonté, Dr. M. Tarbouchi.

The development of autonomous Unmanned Aerial Vehicles (UAVs) is of high interest to many governmental and military organizations around the world. An essential aspect of robot autonomy is the ability for automatic path planning. Solutions that have been previously proposed often use models that are too simplistic and cannot be applied to real UAV systems. On the other hand, solutions that account for the dynamics of the UAV and the complexity of a realistic environment require significant computing power and cannot be used in real time or onboard the UAV. In this thesis, we use two non-deterministic optimization algorithms, the genetic algorithm (GA) and the particle swarm optimization algorithm (PSO), to cope with the complexity of the problem and compute feasible and quasi-optimal trajectories for UAVs in a 3D environment while considering the dynamic properties of the vehicle. The characteristics of the optimal path are represented in the form of a multi-objective cost function that we developed. The paths produced are composed of line segments and circular arcs. We reduce the execution time of our solutions by using the "single-program, multiple-data" parallel programming paradigm and we achieve real-time performance on standard COTS multi-core CPUs. We finally interface our path planning algorithms with a MicroPilot autopilot and successfully fly a generated trajectory in a test flight. After achieving a quasi-linear speedup of 7.3 on 8 cores and an execution time of 10 s for both algorithms, we conclude that by using a parallel implementation on standard multicore CPUs, real-time path planning for UAVs is possible. Moreover, our rigorous comparison of the two algorithms shows, with statistical significance, that the GA produces superior trajectories than the PSO

Keywords: UAV, path planning, genetic algorithm, particle swarm optimization, parallel programming, autopilot, test flight.

Table des Matières

Liste des tableaux

Liste des figures

Liste des symboles

P_R	Puissance requise (W)
P_{AS}	Puissance disponible au niveau de la mer (W)
P_A	Puissance disponible (W)
η	Efficacité de l'hélice (0 à $1, sans\ unité$)
g	Constante gravitationnelle ($9.80665\ m \cdot s^{-2}$)
ρ_S	Densité de l'air au niveau de la mer ($1.225\ kg \cdot m^{-3}$)
ρ	Densité de l'air ($kg \cdot m^{-3}$)
V_c	Vitesse de croisière ($m \cdot s^{-1}$)
θ	Angle entre le plan horizontal et le segment de droite ascendant (rad)
S	Surface des ailes (m^2)
b	Envergure des ailes (m)
C_{D0}	Coefficient de traînée globale de l'avion à portance nulle ($sans\ unité$)
e	Facteur d'efficacité d'Oswald ($sans\ unité$)
c	Consommation spécifique de carburant ($N \cdot (s \cdot W)^{-1}$)
RAC	Rapport air carburant ($environ\ 14.7, sans\ unité$)
T_s	Température de l'air au sol (K)
T	Température de l'air (K)
a	Taux de changement de la température en fonction de l'altitude, en dessous de 11 km ($6.5 \cdot 10^{-3}\ K \cdot m^{-1}$)
h	Altitude au-dessus du niveau de la mer (m)
W	Poids de l'avion (N)
n_{max}	Facteur de charge maximum permis pour un drone ($sans\ unité$)

Chapitre 1

Introduction

1.1 Planification de trajectoires pour les drones

Le problème de la planification de trajectoires est un sujet de recherche qui dure depuis plusieurs années à cause de sa complexité. Les méthodes initialement développées étaient efficaces pour des modèles primitifs de véhicules se déplaçant dans des environnements simples, mais ne peuvent être utilisées dans des situations plus complexes et réalistes telles qu'un drone volant dans un espace 3D. Au cours des années, les modèles utilisés ont été raffinés pour inclure le comportement dynamique du véhicule et les détails d'un environnement 3D plus réaliste. Le concept de la meilleure trajectoire a aussi évolué. Les approches classiques tentaient de calculer le chemin le plus court tandis que les méthodes récentes cherchent un chemin qui satisfait plusieurs contraintes. Dans le cas des drones, une trajectoire minimisant la distance, l'altitude et la consommation de carburant pourrait être préférable. Ceci n'est qu'un exemple; les qualités recherchées varient souvent d'après l'objectif de la mission. Trouver la trajectoire optimale est en fait un problème d'optimisation dans un espace de recherche multidimensionnel qui ne peut être résolu en temps polynomial par des méthodes déterministes. Ces dernières ont donc été remplacées par des algorithmes non déterministes qui supportent mieux la complexité du problème. On ne trouve plus la meilleure solution à un problème simple, mais une bonne solution à un problème complexe et réaliste dans un temps raisonnable.

Pour comprendre le problème de planification de trajectoires, il est important de bien saisir le sens de certains concepts. Pour une définition complète, veuillez-vous référer à [1].

1) **Espace environnement :** réfère à l'environnement physique dans lequel se trouve le véhicule, habituellement 2D pour un véhicule sur roues et 3D pour un drone;

2) **Configuration :** un vecteur qui définit l'état de l'objet, soit sa position, son orientation et sa forme. La configuration d'un véhicule rigide, tel un drone, dans un environnement 3D requiert 6 éléments $(x, y, z, lacet, tangage, roulis)$. Un robot équipé d'un bras mobile nécessiterait un plus grand vecteur de configuration;

3) **Espace des configurations :** l'espace de toutes les configurations possibles;

4) **État :** similaire à la configuration, mais munis de termes supplémentaires pour représenter le dynamisme du véhicule $(x, y, z, lacet, tangage, roulis, \Delta x, \Delta y, \Delta z, \Delta lacet, \Delta tangage, \Delta roulis)$;

5) **Espace des états :** l'espace de tous les états possibles;

6) **Espace libre :** l'espace d'environnement, de configurations ou d'états sans obstacle;

7) **Espace obstrué :** contraire de l'espace libre;

8) **Route :** courbe tracée dans l'espace de configurations; et

9) **Trajectoire :** courbe en fonction du temps tracée dans l'espace environnement.

D'après les définitions précédentes, le problème de planification de trajectoires pour les drones consiste à représenter l'espace environnement en se basant habituellement sur une carte de données numériques d'élévation, construire l'espace de configuration d'après la forme du véhicule, définir l'espace d'état en considérant les caractéristiques dynamiques du drone pour ensuite résoudre le problème et trouver la route optimale reliant le point de départ au point d'arrivée tout en optimisant différentes caractéristiques. Une fois calculée, la route doit être convertie en trajectoire faisable en fonction du temps pour ensuite être suivie par le drone.

D'après la description donnée ci-dessus, il est évident que l'approche directe est beaucoup trop complexe pour être possible. Plusieurs auteurs évitent cette complexité en divisant le problème en quatre étapes. L'approche standard décrite dans [1] peut se résumer comme suit :

1) construire une représentation de l'environnement;

2) définir une fonction de coût qui exprime les qualités de la route optimale;

3) utiliser un algorithme d'optimisation pour trouver une route qui minimise la fonction de coût; et

4) transformer ou lisser la route finale en une trajectoire faisable par le drone.

Un compte rendu des différentes publications scientifiques proposant des solutions à chacune des étapes ci-dessus est présenté en détail au chapitre 2.

La planification de trajectoires peut être confondue avec le système de contrôle autonome du drone. En fait, tel qu'expliqué dans [2] et illustré en rouge à la figure 1, le module de planification de trajectoires fait habituellement partie d'un système de contrôle supérieur. Ce système complexe fournit les données nécessaires au module de planification de trajectoires et suit le chemin généré à l'aide d'un module de navigation intégré. Le système de contrôle suit habituellement une architecture hiérarchique et permet d'identifier et de corriger les erreurs ou mals fonctionnements de ses sous-systèmes. Dans ce mémoire, nous abordons uniquement le module de planification de trajectoires et omettons tous détails reliés au système de contrôle autonome.

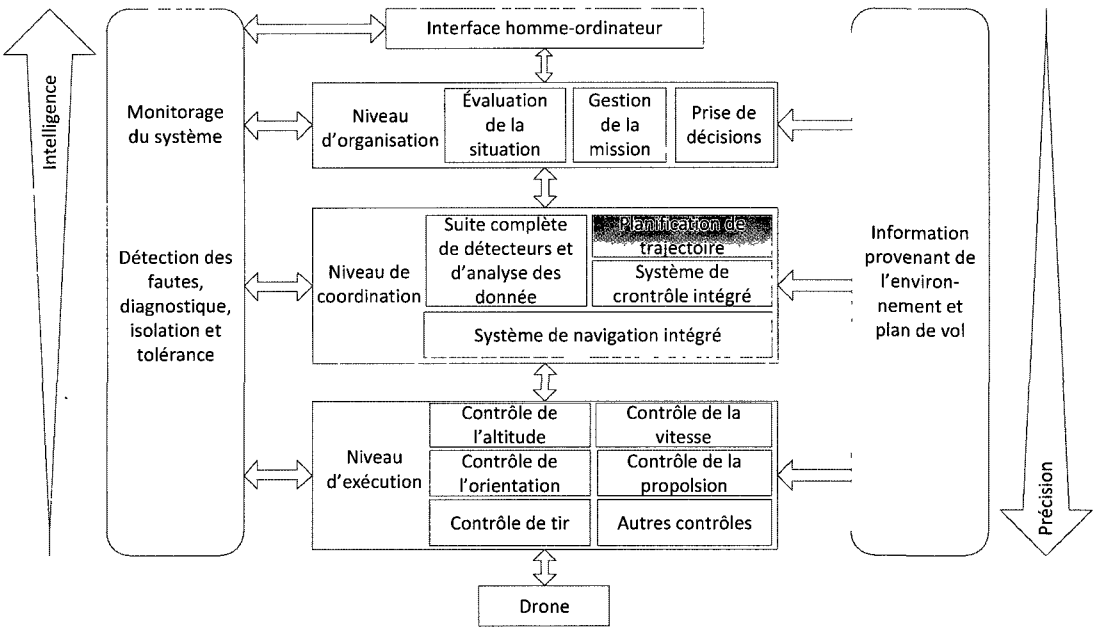


Figure 1. Architecture hiérarchique d'un système de contrôle autonome d'un drone (reproduit de [2])

1.2 Objectifs de recherche

Une étape essentielle au développement de drones autonomes est la conception d'un module de planification de trajectoires qui permet de générer automatiquement un trajet faisable et quasi optimal reliant un point de départ à un point d'arrivée. Puisque les solutions actuelles pour le développement d'un tel module sont incomplètes, nous proposons dans ce mémoire quelques contributions.

Le premier objectif de ce mémoire consiste à appliquer deux algorithmes d'optimisation non déterministes au problème de planification de trajectoires pour les drones. Les algorithmes utilisés sont l'algorithme génétique (AG) et l'optimisation par essaim de particules (OEP). Les trajectoires générées sont composées de segments de droite, d'arcs circulaires et de cercles sur plateaux horizontaux. Elles sont exemptes de discontinuité, elles minimisent la distance à parcourir, elles minimisent l'altitude moyenne afin d'améliorer le caractère furtif du drone, elles contournent les zones dangereuses, elles évitent toute collision avec le terrain et elles respectent les caractéristiques dynamiques du drone (puissance maximale, quantité maximale de carburant, force gravitationnelle maximale permise). L'utilisation d'algorithmes non déterministes permet d'affronter la complexité du problème et d'arriver à un résultat dans un temps limité.

Un second objectif consiste à évaluer la possibilité du calcul de trajectoires en temps réel. Afin de minimiser le temps d'exécution et de maximiser l'utilisation des microprocesseurs actuels, nous développons une version parallèle de chacun des deux algorithmes suivant le paradigme de programmation « instruction unique, données multiples ». On espère ainsi que le temps de calcul alors obtenu permette les applications en temps réel.

Un troisième objectif consiste à comparer la qualité des solutions produites par chacun des algorithmes. Puisque l'accélération obtenue par la parallélisation n'est pas nécessairement la même dans les deux cas, nous comparons la qualité des solutions générées par les versions parallèles de nos algorithmes. Nous utilisons un temps de calcul fixe et une fonction de coût unique. Notre comparaison se fait à l'aide de 40 scénarios distincts sur 8 terrains différents, dont 6 sont réels. Nous calculons chaque trajectoire 60 fois en utilisant l'AG puis l'OEP et permettons ainsi une analyse quantitative et statistiquement significative des deux approches.

Finalement, le dernier objectif de notre recherche consiste à interfacer nos algorithmes de planification de trajectoires à un pilote automatique commercial. Pour ce faire, nous installons

un pilote automatique MP2128g [3] fabriqué par Micropilot dans un avion miniature E-Flite Apprentice **[4]**. Nous développons aussi un logiciel qui permet d'utiliser des cartes d'élévation digitales réelles provenant de la base de données canadienne GeoBase [5] et de générer des trajectoires qui respectent les caractéristiques du drone sous forme de fichiers de vol compatibles avec le pilote automatique. Nous validons finalement notre interface par vol d'essai.

Tel que discuté au chapitre 2, ces objectifs répondent à des lacunes des solutions proposées antérieurement.

1.3 Motivations

L'utilisation de drone à des fins militaires ou civiles représente un avantage réel pour les gouvernements. Entre autres, les Forces canadiennes ont récemment renouvelé leur contrat avec MacDonald, Dettwiler and Associates Ltd. pour poursuivre l'utilisation du drone Heron pour des missions de surveillance en Afghanistan [6]. Toujours en support des opérations en Afghanistan, les Forces canadiennes utilisent aussi des mini-drones tels que le ScanEagle de Insitu [7] et le Maveric de Prioria [8]. Au Canada, les drones représentent une option intéressante pour la surveillance des côtes et de l'Arctique. Lors de sa visite au Collège militaire royale du Canada en novembre 2009, le Chef du personnel militaire, le Major général Semianiw a fortement encouragé toute recherche visant à augmenter la présence canadienne dans l'Arctique canadien. Le projet JUSTAS (CF's Joint UAV Surveillance and Target Acquisition System) lancé par le gouvernement canadien vise justement à acquérir de nouveaux drones qui serviraient, entre autres, à surveiller notre territoire canadien. Cependant, d'après [9] et [10], une des limitations qui auraient retardé le projet est le manque de pilotes. Même si le drone est inhabité, il doit quand même être piloté à distance. Le Canada peut acheter des drones, mais plus difficilement des pilotes. Il existe donc une motivation réelle pour le développement de drones autonomes. Or, un élément essentiel à l'autonomie est la planification automatique de trajectoires qui permettrait au drone de voyager d'un point de départ à un point d'arrivée en suivant une trajectoire optimale.

La planification de trajectoires pour les drones est un sujet intéressant du point de vue de la demande, mais aussi du point de vue académique et scientifique. Ce domaine de recherche rejoint plusieurs disciplines telles que l'aéronautique, le génie informatique matériel et le génie

logiciel. Le sujet choisi est d'autant plus intéressant puisqu'il couvre toutes les étapes du développement de la conception à la validation.

1.4 Organisation du mémoire

Ce mémoire est organisé comme suit. Nous présentons au chapitre 2 les différentes solutions qui ont été proposées dans le passé pour résoudre le problème de planification de trajectoires. Nous discutons de leurs lacunes et expliquons comment notre recherche aborde ces limites. Au chapitre 3, nous définissons la fonction de coût qui est ensuite utilisée par notre AG et notre OEP pour générer des trajectoires faisables. Cette fonction de coût décrit les qualités recherchées pour obtenir la trajectoire optimale. Nous expliquons ensuite les détails d'implémentation de l'AG et de l'OEP au chapitre 4 et 5. Nous traitons au chapitre 6 du lissage de la trajectoire finale par des arcs circulaires et des cercles sur plateau horizontal. La description et l'analyse de nos implémentations parallèles de l'AG et de l'OEP sont présentées au chapitre 7. Le chapitre 8 traite de la comparaison de la qualité des solutions produites par chacun des algorithmes. Finalement, nous discutons au chapitre 9 de l'installation du pilote automatique MP2128g dans un avion E-Flite Apprentice; du développement de l'outil logiciel qui utilise nos algorithmes pour générer des trajectoires faisables sous forme de fichiers de vol pour le pilote automatique; et de la validation de l'interfaçage par vol d'essai. Des suggestions pour travaux futures sont proposées dans le chapitre de conclusion.

Chapitre 2

Revue de la littérature

Il est important de réviser les avancements scientifiques publiés jusqu'à maintenant dans le domaine de la planification de trajectoires pour les drones afin de bien comprendre leurs faiblesses et mieux apprécier les solutions que nous proposons. Dans la littérature, plusieurs auteurs abordent le problème de la planification de trajectoires suivant une approche par étapes. En se basant sur [1], on peut résumer cette méthode à 1) la représentation de l'environnement; 2) la définition d'une fonction de coût; 3) l'utilisation d'un algorithme d'optimisation pour minimiser la fonction de coût et finalement; 4) le lissage de la route pour obtenir une trajectoire continue et faisable par le drone. Nous présentons dans ce chapitre un compte rendu des différentes contributions scientifiques organisées d'après ces quatre étapes.

2.1 Représentation de l'environnement

La représentation de l'environnement est la première étape du processus de planification de trajectoires. Il s'agit de discrétiser le monde physique dans lequel se déplace le véhicule en une représentation significative pour l'algorithme d'optimisation. Choisir la bonne représentation est important puisqu'elle affecte la performance de l'algorithme d'optimisation [11]. D'après [1], nous présentons dans cette section deux familles de représentations qui sont souvent utilisées : les graphes et les décompositions en cellules.

2.1.1 Représentation par graphes

Un graphe consiste en un ensemble de sommets reliés par des arêtes. Il réduit habituellement l'espace de recherche, mais peut nécessiter un effort de calcul considérable pour être généré. Comme illustré à la figure 2, un graphe de visibilité représente les obstacles par des polygones et relie chaque sommet aux autres sommets visibles. Cette représentation à l'avantage d'inclure le chemin le plus court, mais souffre de collisions possibles puisque le trajet final frôle les obstacles. Les graphes de visibilité peuvent être utilisés en 3D, mais la génération exacte est prouvée NP-difficile [1]. Pour éviter une possible collision, dans un graphe de Voronoï (figure 3), les arêtes sont dessinées équidistantes des obstacles maximisant ainsi la distance entre un chemin possible et l'obstacle. La solution d'un graphe de Voronoï 3D est complète, mais pas optimale [1].

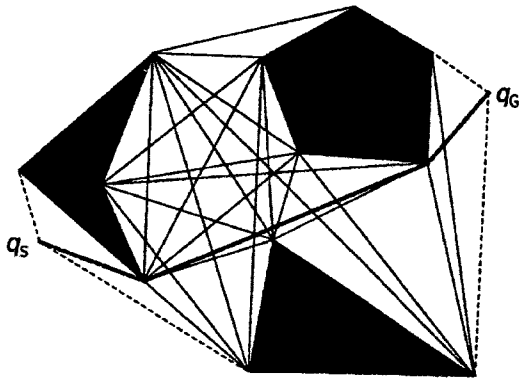


Figure 2. Exemple de graphe de visibilité où q_S est le point de départ, q_G est le point d'arrivée et les régions sombres représentent les obstacles (reproduit de [12])

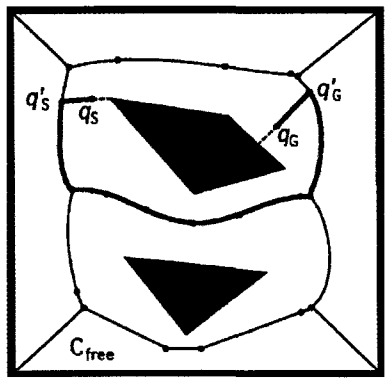


Figure 3. Exemple de graphe de Voronoï où q_S est le point de départ, q_G est le point d'arrivée et les régions sombres représentent les obstacles (reproduit de [12])

2.1.2 Représentation par décomposition cellulaire

La seconde famille de représentations consiste à exprimer l'environnement sous forme d'un ensemble de cellules. La décomposition peut être exacte lorsque l'espace libre est divisé en polygones ou approximative lorsque l'espace entier est décomposé en cellules identiques. Dans le cas de la planification de trajectoires pour les drones, une décomposition cellulaire approximative est souvent utilisée sous forme d'une matrice 2D où chaque cellule contient l'élévation du terrain. Les zones d'exclusion aérienne, ou zones dangereuses, peuvent être indiquées par une élévation plus haute que le plafond de vol du drone, mais sont souvent gardées dans une table séparée telle que dans [13], [14], [15] et [16]. Une représentation plus raffinée de la décomposition cellulaire approximative consiste à utiliser une décomposition par arbre de quadrants (figure 4). L'auteur de [13] démontre que cette représentation peut réduire de 40 % le temps d'exécution de la planification de trajectoires. D'un autre côté, l'arbre de quadrants doit être reconstruit lorsqu'un nouvel élément est détecté.

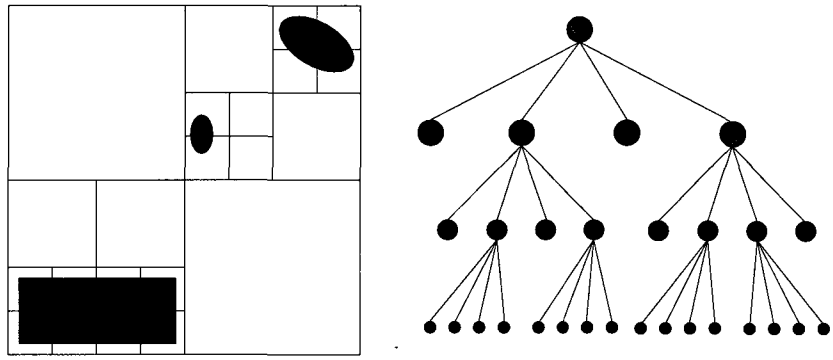


Figure 4. Exemple de représentation de l'environnement par arbre de quadrants

2.2 Fonction de coût

Le chemin optimal a souvent été associé au chemin le plus court. C'est par exemple le cas pour le problème du commis voyageur. Pour un drone, la trajectoire optimale est beaucoup plus complexe. Cependant, il est possible d'identifier les caractéristiques d'une telle solution (pas trop longue, maintient une basse altitude, évite les zones dangereuses, respecte les caractéristiques dynamiques du drone, etc.) et de les exprimer sous forme d'une fonction de coût. Un algorithme d'optimisation peut alors être utilisé pour trouver une solution qui minimise le coût. Définir une fonction de coût est toutefois assez difficile. Par exemple, dans son mémoire de maîtrise [15], Bélanger définit la fonction de coût suivante :

$$C = \sum_{i=1}^{n} \omega_1 l_i + \omega_2 h_i + \omega_3 f_i \qquad (1)$$

où l_i est la distance euclidienne du segment i, h_i est l'altitude moyenne du segment i multiplié par sa longueur et f_i est associé à la distance entre le segment i et les zones dangereuses. Les ω sont les poids associés à chaque terme et ont reçu une valeur de 1 par l'auteur. L'équation semble bonne, mais ses termes ne sont pas normalisés et la valeur de celui associé à l'altitude moyenne est grandement surévalué : une trajectoire ayant une altitude moyenne d'une unité plus haute a le même coût qu'une trajectoire deux fois plus longue. Dépendamment de l'échelle verticale utilisée, cette lacune peu grandement affecter la qualité des solutions générées. Il serait plus approprié d'utiliser une approche qui équilibre mieux chacun des objectifs comme les méthodes de somme des coûts normalisés, du compromis et de la programmation avec but qui sont expliquées dans [17].

2.3 L'Algorithme d'optimisation

Les premières solutions au problème de planification de trajectoires proposées dans les années 70 utilisent des algorithmes déterministes. Ces algorithmes sont efficaces pour résoudre des problèmes simples et permettent de trouver la meilleure solution. Des exemples d'algorithmes déterministes sont l'algorithme de Dijkstra, le A*, le A* dynamique avec objectif et le gradient de potentiel [18]. Les algorithmes déterministes ne peuvent cependant pas être utilisés pour des problèmes plus complexes comme la planification de trajectoires pour les drones dans un environnement 3D réaliste. Tel qu'illustré à la figure 5, les algorithmes déterministes ont été

remplacés par les algorithmes non déterministes qui permettent de trouver une bonne solution (pas nécessairement la meilleure, mais suffisamment bonne) à des problèmes plus complexes qui ne pourraient autrement être résolus en temps polynomial. L'utilisation d'algorithmes non déterministes permet aussi de contrôler le temps d'exécution, ce qui est critique pour une implémentation en temps réel.

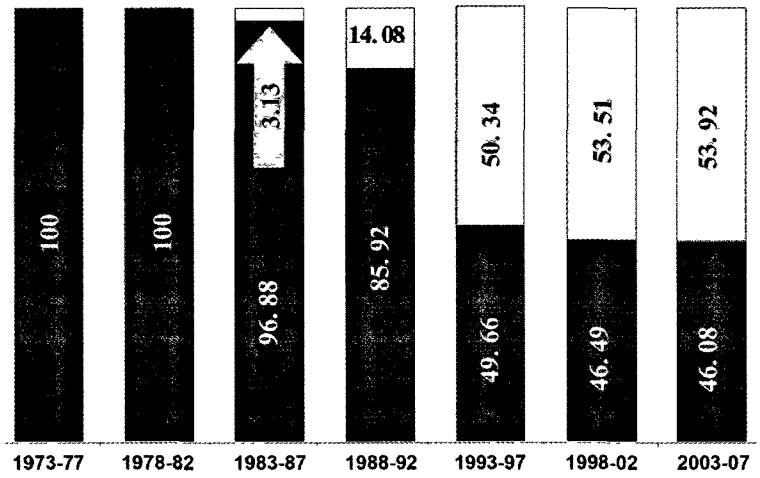


Figure 5. Pourcentage d'utilisation d'algorithmes déterministes (gris foncé) et non déterministes (gris pale) pour résoudre le problème de planification de trajectoires pris d'une étude de plus de 1 400 articles (reproduit de [19])

2.3.1 Algorithme génétique

L'algorithme génétique (AG) a été développé par John Holland dans les années 60 et publié pour la première fois en 1975 [20]. Basé sur la théorie génétique de l'évolution naturelle de Darwin, l'AG a été appliqué à plusieurs problèmes d'optimisation. De nos jours, il est l'algorithme le plus utilisé pour résoudre le problème de planification de trajectoires [19]. Son concept est simple : il s'agit de représenter des solutions aléatoires sous forme d'une population de chromosomes et de simuler le processus évolutif par la sélection, la reproduction et les mutations. Tout comme les individus biologiques qui s'adaptent à leur environnement, les solutions encodées évoluent pour minimiser la fonction de coût. Dans le cas de la planification de trajectoires pour les drones, une approche commune est d'encoder une trajectoire complète dans chaque chromosome sous la forme d'une série de coordonnées euclidiennes 2D ou 3D représentant les gènes [13], [14], [15], [16]. Par l'évolution, la population de chromosomes s'améliore et évolue vers une trajectoire quasi optimale. Le diagramme de fonctionnement de l'AG original se trouve à la figure 6 et quelques opérateurs génétiques, à la figure 7.

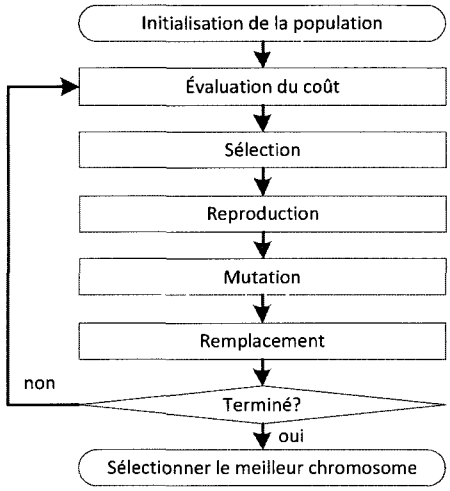


Figure 6. Diagramme de fonctionnement de l'AG

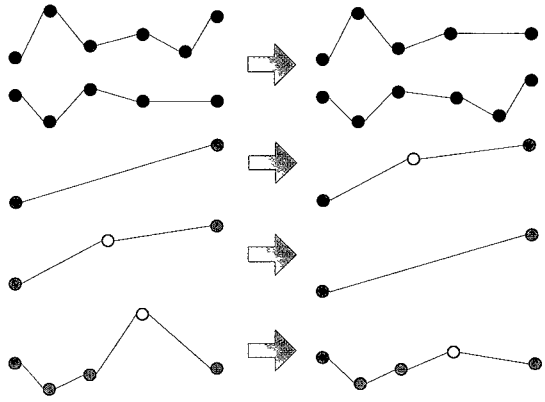


Figure 7. Dans l'ordre : reproduction, mutation d'addition, mutation de soustraction et mutation de modification

Comme il est expliqué dans [17], l'AG de base est très simple, mais doit être adapté au problème en question. En d'autres mots, il faut que l'usager détermine la représentation de chromosomes, la fonction de coût, le mécanisme de sélection, le mécanisme de reproduction, les différentes mutations possibles et le critère de terminaison. Cette tâche peut être compliquée, surtout dans le cas d'un problème d'optimisation à objectifs multiples (minimiser la longueur du trajet, minimiser l'altitude, respecter les contraintes dynamiques du drone, etc.). Le fonctionnement de l'AG est décrit en détail au chapitre 4.

Dans la littérature, on remarque que plusieurs auteurs modifient l'AG de base en vue d'améliorer sa performance. Cependant, ces modifications augmentent sa complexité et l'amélioration apportée est souvent limitée aux scénarios précis utilisés par l'auteur. Par exemple, dans [15], Bélanger utilise des mutations intelligentes telles que l'adoucissement des angles aigus pour corriger manuellement des trajectoires non faisables. Cette modification réduit l'aspect stochastique de l'algorithme et modifie quelque peu son comportement. Il serait préférable de modifier la fonction de coût et de pénaliser les solutions comportant des angles aigus au lieu d'altérer les opérateurs génétiques. La fonction de coût reste alors universelle et peut aussi être utilisée par d'autres algorithmes d'optimisation. L'ajout de mutations intelligentes est donc discutable. Dans [21], les auteurs introduisent un mécanisme de vibration où tous les gènes de tous les chromosomes sont modifiés simultanément. Ce processus est si radical qu'il doit être jumelé à l'élitisme pour conserver les meilleures solutions. Encore une fois, les résultats obtenus ne démontrent pas l'avantage de cette modification. Un dernier exemple se trouve dans [22] où

les auteurs emploient le recuit simulé pour contrôler le remplacement des parents par les enfants. Ceci ne fait qu'ajouter un deuxième mécanisme de convergence à l'AG (le premier étant la sélection) et diminue probablement l'exploration (mais augmente nécessairement la complexité des calculs).

2.3.2 Colonies de fourmis

L'algorithme d'optimisation par colonies de fourmis (OCF) a été proposé pour la première fois en 1992 par Dorigo [23] et appliqué au problème de planification de trajectoires en 1994 [24]. Il est basé sur le comportement naturel des fourmis qui tracent un trajet entre leur nid et une source de nourriture afin d'accumuler des réserves. Tel qu'illustré à la figure 8, les fourmis voyagent de la fourmilière et la nourriture tout en sécrétant de la phéromone (schéma 1). Initialement, elles choisissent un chemin de façon aléatoire et retournent sur leurs propres pas suivant leur phéromone (schéma 2). La phéromone s'évapore, mais garde une concentration plus élevée sur le chemin le plus court puisqu'il est parcouru plus rapidement et donc plus souvent. Au lieu de retourner sur leurs pas, les fourmis qui avaient initialement choisi un chemin plus long optent maintenant pour le chemin plus court dû au plus haut niveau de phéromone. À la longue, la colonie de fourmis s'organise et emprunte le chemin optimal (schéma 3).

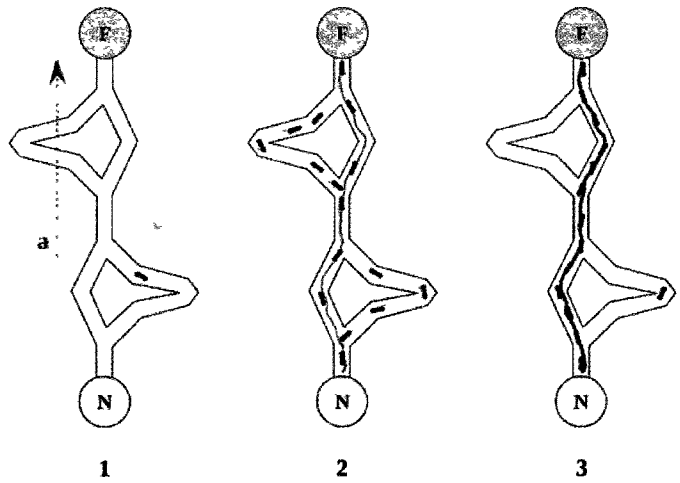


Figure 8. Comportement naturel des fourmis (reproduit de [12])

Ce même principe a été appliqué à la planification de trajectoires pour les drones. Dans ce cas, une fonction de coût complexe peut être utilisée au lieu de simplement considérer la longueur du trajet. Puisque les fourmis se déplacent initialement de façon aléatoire, il arrive que

l'OCF ne converge pas dans un temps raisonnable si l'espace de recherche est trop grand. Par exemple, lors de sa comparaison entre l'AG et l'OCF, Bélanger [15] a dû discrétiser sa carte d'élévation en une matrice 2D de 20 x 20 afin que les fourmis réussissent à atteindre le point d'arrivée dans un temps acceptables. Dans le cas de l'AG, il a utilisé une matrice 2D de 500 x 500. De plus, lorsqu'utilisé pour la planification de trajectoires, l'OCF est souvent implémenté sur un graphe (telle une grille orthogonale) où les fourmis se déplacent d'un sommet à l'autre en suivant les arêtes. Dans ce cas, l'espace de recherche est discret contrairement à celui de l'AG qui peut être continue.

2.3.3 Optimisation par essaim de particules

L'optimisation par essaim de particules (OEP) a été proposée pour la première fois en 1995 par Kennedy et Eberhart [25]. Tout comme le gracieux mouvement d'une volée d'oiseaux, l'algorithme simule une population de particules qui se déplace doucement vers la solution optimale dans un espace de recherche multidimensionnel. La position d'une particule représente une solution possible et est initialisée aléatoirement. Chaque particule se déplace selon leur vecteur de vitesse unique. À chaque itération, cette vitesse est modifiée pour s'orienter vers la meilleure position déjà occupée par la particule (mémoire locale) et la meilleure position de la meilleure particule de l'essaim (mémoire sociale). Tel qu'expliqué dans [25], cet ajustement tient compte de l'inertie de la particule et inclut une certaine influence stochastique. La mise à jour de la vitesse et de la position de la particule i au temps t se fait d'après les équations (2) et (3).

$$\begin{aligned} \boldsymbol{v_i}(t) = {} & C_1\boldsymbol{v_i}(t-1) + C_2\boldsymbol{r_1}.*\left(\boldsymbol{p_i}(t-1) - \boldsymbol{x_i}(t-1)\right) \\ & + C_3\boldsymbol{r_2}.*\left(\mathbf{g}(t-1) - \boldsymbol{x_i}(t-1)\right) \end{aligned} \tag{2}$$

$$\boldsymbol{x_i}(t) = \boldsymbol{x_i}(t-1) + \boldsymbol{v_i}(t) \tag{3}$$

Dans ces équations, les variables en gras représentent des vecteurs. $\boldsymbol{v_i}$ est la vitesse de la particule i, $\boldsymbol{x_i}$ est sa position, $\boldsymbol{p_i}$ est sa meilleure position déjà occupée, $\mathbf{g}$ est la meilleure position de la meilleure particule de l'essaim, $\boldsymbol{r_1}$ et $\boldsymbol{r_2}$ sont des vecteurs de valeurs aléatoires entre 0 et 1 et finalement, C_1, C_2 et C_3 sont des constantes.

L'algorithme présenté ci-dessus consiste en une approche globale : les particules s'orientent vers la meilleure solution de l'essaim. Par contre, il est aussi possible d'utiliser une approche locale où l'influence sociale d'une particule est limitée à son voisinage proche.

Eslam Pour applique ces deux méthodes au problème de la planification de trajectoires pour les drones et démontre que l'OEP local ralentit la convergence, mais améliore l'exploration [16]. Dans son implémentation, chaque particule représente une trajectoire possible dans un environnement 3D sous forme d'une série de coordonnées (x, y, z). Toutes les particules ont la même dimension définie par le nombre de points de passage des trajectoires. Par exemple, pour une trajectoire à 10 points de passage, les particules se déplacent dans un espace de recherche à 30 dimensions. Tous comme pour l'AG, plusieurs auteurs apportent des modifications à l'algorithme d'OEP dans le but d'accroître sa performance. Par exemple, Shi et Eberhart [26] proposent de diminuer linéairement la valeur du poids C_1 tout au long du processus itératif afin de réduire l'inertie des particules. Appliqué à la planification de trajectoires, Wang et Lu [27] intègre un régulateur PID (proportionnel, intégral, dérivé) pour faire varier la valeur de C_1 pour chacune des particules d'après la distance séparant la particule de la meilleure. Hongguo et autres [28] avancent eux aussi une technique semblable, mais réfèrent à cette variation du poids C_1 en utilisant le terme « mutation », ce qui est trompeur. Finalement, les auteurs de [29] comparent trois optimisations possibles de l'OEP appliquée à la planification de trajectoires en 2D : OEP avec variation linéaire du poids, OEP chaotique et OEP avec variation de deuxième ordre. Leurs résultats démontrent que ces modifications allongent le temps de calcul et que l'OEP avec variation linéaire du poids est préférable.

2.3.4 Recuit simulé

Présenté pour la première fois en 1983 [30], le recuit simulé est une méthode d'optimisation non déterministe qui consiste à simuler le procédé de recuit métallurgique où un atome chauffé quitte son minimum local d'énergie pour finalement atteindre un niveau d'énergie inférieur. Tout comme l'algorithme des essaims de particules, une solution possible est encodée dans un atome qui se déplace dans un espace multidimensionnel. Par contre, l'optimisation par recuit simulé n'utilise qu'une seule particule. Au début du procédé, lorsque la température T est élevée, la particule se déplace presque aléatoirement allant même vers une solution avec un plus haut coût afin de favoriser l'exploration. À mesure que le procédé avance et que la température baisse, cet aspect aléatoire diminue et la particule se dirige principalement vers une solution qui minimise la fonction de coût. Telle que définie dans [19], la probabilité d'accepter une solution de qualité supérieure est toujours de 1, mais définie comme suit pour une solution inférieure :

$$P_{acceptance} = e^{-\frac{F(\boldsymbol{x}(t))-F(\boldsymbol{x}(t+1))}{T}} \quad (4)$$

où $F(\boldsymbol{x}(t))$ est le coût de la solution actuelle $\boldsymbol{x}(t)$, $F(\boldsymbol{x}(t+1))$ est le coût de la solution candidate $\boldsymbol{x}(t+1)$ et T est la température qui est réduite à chaque itération. Le recuit simulé a déjà été employé pour résoudre le problème de planification de trajectoires [31], mais est plus souvent utilisé pour améliorer ou compléter un autre algorithme tel que le gradient de potentiel [19]. Contrairement aux algorithmes d'optimisation qui utilisent une population de solutions, le recuit simulé offre une exploration plus limitée de l'espace de recherche et est rarement appliqué problèmes à très grand nombre de dimensions comme celui la planification de trajectoires pour les drones.

2.3.5 Bande élastique

Le cinquième algorithme non déterministe présenté dans ce document est l'algorithme de la bande élastique. Cet algorithme est particulièrement intéressant puisque son application au problème de planification de trajectoires est très concrète. Relié au réseau de neurones en carte auto organisatrice, cet algorithme a initialement été proposé par Durbin et Willshaw en 1987 sous le nom de réseau élastique pour résoudre le problème du commis voyageur [32]. Le principe consiste à placer un réseau circulaire au centre des villes et étirer sa bordure pour qu'elle relie chacune des villes. L'élasticité du réseau est exprimée sous forme d'équation mathématique et minimise le trajet final. En 2005, Moreno et Castro appliquent une méthode similaire au problème de planification de trajectoires [33]. Leur technique consiste à connecter le point de départ au point d'arriver à l'aide d'une bande élastique définie par un nombre fixe de points de contrôle (PC) tel qu'illustré à la figure 9. À chaque étape du processus itératif, un des PC est choisi aléatoirement ainsi qu'un point imaginaire dans son entourage. Le coût des deux points est évalué indépendamment et reçoit une valeur de -1 si le point se situe dans l'espace libre et de 1 si le point se situe sur un obstacle. D'après le coût des deux points, le PC est alors attiré ou repoussé par l'autre point. Lors de ce déplacement, le PC est aussi soumis à une deuxième force, soit celle de la bande élastique. Cette seconde force minimise la longueur du trajet total. Après un nombre défini d'itérations, un nouveau PC est ajouté là où la tension est la plus grande assurant un certain lissage de la courbe.

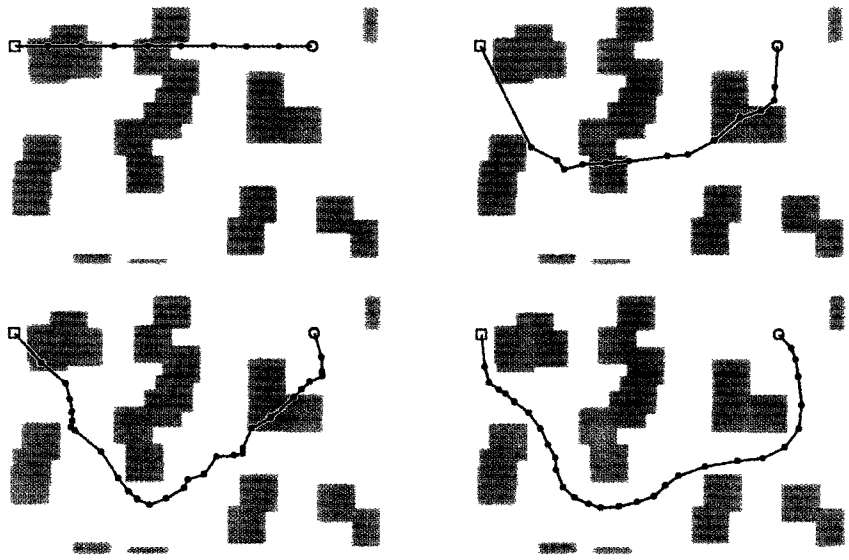


Figure 9. Quatre instances du processus itératif de l'algorithme de la bande élastique pour un véhicule dans un environnement 2D (Haut à gauche : PC = 10 à t = 0. Haut à droite : PC = 15 à t = 4000. Bas à gauche : PC = 30 à t = 16000. Bas à droite : PC = 30 à t = 40000.) (reproduit de [33])

L'algorithme de la bande élastique offre un avantage intéressant comparé aux algorithmes non déterministes présentés précédemment : la localité de la recherche. Au lieu de commencer la recherche avec plusieurs trajectoires aléatoires, le processus commence avec la trajectoire la plus courte et la modifie localement pour contourner les obstacles. L'implémentation des auteurs [33] calcule un chemin quasi optimal dans un environnement 2D de 800 x 500 unités en 360 ms sur un Pentium IV 2.4 GHz.

Dû à sa rapidité d'exécution, l'algorithme de la bande élastique présente un certain potentiel pour la planification de trajectoires pour les drones. Puisque cette approche n'a jamais été implémentée en 3D, nous avons développé une version prototype 3D qui utilise une carte d'élévation digitale, des obstacles cylindriques et une fonction de coût complexe qui minimise la longueur et l'altitude du trajet tout en respectant les propriétés dynamiques de drone. Au lieu de considérer l'état du point aléatoire et du PC (-1 si le point est sur un obstacle et 1 s'il est dans l'espace libre), nous calculons le coût de la trajectoire actuelle et celui de la trajectoire qui passerait par ce point aléatoire. Nous remarquons que cette deuxième trajectoire est de qualité inférieure dans plus de 95 % des itérations et la bande élastique est constamment repoussée sans jamais converger vers une bonne solution. Il semble donc que l'heuristique de la bande élastique est adaptée au scénario précis utilisé par les deux auteurs et illustré à la figure 9, mais n'est pas assez général pour être utilisé dans une situation plus complexe telle que la planification de

trajectoires 3D pour les drones. Il n'est donc pas surprenant qu'aucun article traitant de cet algorithme appliqué à la planification de trajectoires n'ait été publié depuis la parution de [33] en 2005.

2.4 Lissage de la trajectoire

Il est évident que la route composée de segments de droite générée par l'algorithme d'optimisation contient des discontinuités dans leur tangente, ou vitesse, et ne peut être suivie par un drone. Pour créer une trajectoire faisable, il faut donc lisser cette route. Les deux techniques principales proposées dans la littérature sont le lissage par spline et par arcs circulaires.

L'interpolation par spline produit une trajectoire sans discontinuité et adoucit toute imperfection. Cette méthode est avantageuse puisque la spline est rapidement calculée et peut être modifiée localement sans affecter le reste de la courbe. Cependant, dépendamment du degré utilisé, la courbe résultante peut s'éloigner considérablement de la route non lissée. Par exemple, dans son mémoire de maîtrise, Eslam Pour génère une série de points de passage (x, y, z) à l'aide d'un algorithme d'OEP et lisse ensuite cette route à l'aide d'une spline [16]. Son implémentation produit une trajectoire finale qui s'éloigne considérablement (quelques kilomètres) de la route trouvée par l'algorithme d'OEP. Cette trajectoire n'est plus nécessairement optimale et les contraintes de faisabilité doivent alors être vérifiées à nouveau. Pour éviter cette situation, il est possible d'utiliser la spline dans l'algorithme même d'optimisation et non après. C'est l'approche que prennent les auteurs de [34] lorsqu'ils encodent les paramètres de contrôle de la spline dans les particules du l'OEP au lieu d'encoder des points de passage (x, y, z).

Le lissage par arcs de cercle consiste à éliminer toutes discontinuités de la vitesse en reliant les segments de droite à l'aide d'arcs de cercle. L'avantage de cette méthode est que la trajectoire finale est simple et permet facilement de vérifier que toutes les contraintes dynamiques du drone sont respectées. Cette méthode est présentée en 2D dans [35] et en 3D dans [36]. Une étude complète sur la construction de trajectoire par segments de droite et arcs de cercle est présentée par Labonté dans [37]. Dans son ouvrage, Labonté discute des constructions complexes telles que les cercles sur plateaux horizontaux et les hélices. Il inclut les preuves et les formules mathématiques nécessaires à la construction des courbes, mais aussi toutes les

équations d'aéronautique essentielles pour s'assurer que les contraintes dynamiques du drone sont respectées.

2.5 Limitations des solutions proposées antérieurement

Les solutions au problème de planification de trajectoires présentées dans ce chapitre représentent un avancement scientifique important, mais restent quelque peu incomplètes puisqu'elles ne peuvent pas vraiment être utilisées en temps réel pour la navigation d'un drone.

Premièrement, la plupart des solutions proposées utilisent un modèle de véhicule trop primitif et un environnement trop simple. Dans le cas des drones, il est essentiel de considérer les contraintes dynamiques du véhicule et d'utiliser une représentation de l'environnement complète. Or, les méthodes qui attaquent cette complexité demandent habituellement une puissance de calcul et un temps d'exécution trop grand pour être utilisées en temps réel. Il est donc nécessaire de développer une solution qui modélise le problème d'une façon suffisamment simple pour minimiser le temps d'exécution tout en respectant les contraintes dynamiques du drone. C'est ce que font les auteurs de [15] et [16] qui réussissent à diminuer le temps de calcul à environ 60 s. Or, au chapitre 7, nous proposons d'utiliser un modèle dynamique semblable, mais de réduire le temps de calcul à 10 s en exploitant pleinement le potentiel des processeurs multicœurs de nos jours. D'après les longueurs des trajectoires générées et les vitesses de croisière des drones actuelles, nous montrons que ce temps permet l'utilisation de nos algorithmes en temps réel.

Deuxièmement, basé sur un sondage de plus de 1 400 articles scientifiques [19], l'AG est le moteur d'optimisation le plus utilisé pour la planification de trajectoires. Cependant, à notre connaissance, il n'existe pas de comparaisons rigoureuses des différents algorithmes de planification de trajectoires et on ne peut vraiment confirmer que l'AG est préférable. Pour que la comparaison soit rigoureuse, les méthodes proposées doivent utiliser la même représentation de l'environnement et la même fonction de coût. La qualité de leur solution (coût de la solution finale) et leur temps d'exécution pourraient alors être comparés. C'est pourquoi nous effectuons au chapitre 8 une comparaison méthodique de nos implémentations parallèles de l'AG et de l'OEP pour la planification de trajectoires pour les drones.

Finalement, très peu d'auteurs valident leurs algorithmes de planification de trajectoires par vols d'essai. La plupart se limitent à l'aspect théorique puisqu'il est souvent trop coûteux ou

peu commode d'effectuer des tests pratiques. Par exemple, trois étudiants du Collège Militaire Royal du Canada ont récemment publié des mémoires de maîtrise dans le domaine de la planification de trajectoires pour les drones sans effectuer de validation par vols d'essai [14], [15] et [16]. Afin de permettre la validation future d'algorithmes de planification de trajectoires par vols d'essai, nous présentons au chapitre 9 la plateforme d'expérimentation que nous avons développée. Ce système inclus un outil logiciel qui utilise des cartes d'élévation digitales GeoBase [5] pour générer des trajectoires sous forme de fichiers de vol compatibles avec le pilote automatique MP2128g de MicroPilot [3]. Nous avons aussi installé et configuré le MP2128g dans un avion téléguidé Apprentice fabriqué par E-Flite [4].

En conclusion, nous avons expliqué dans ce chapitre de révision qu'il est commun d'attaquer le problème de planification de trajectoire par une approche par étapes. Nous avons présenté les différentes options proposées dans la littérature pour chacun des stades. Nous avons ensuite identifié quelques limitations et expliqué comment notre recherche répond à ses lacunes. Aux chapitres suivants, nous développons un module expérimental de planification de trajectoires pour les drones suivant cette méthode par étape. La suite de ce document traite de notre représentation de l'environnement, de la définition de notre fonction de coût, de notre implémentation de l'AG et de l'OEP et de la technique de lissage que nous utilisons.

Chapitre 3

Fonction de coût

Dans le processus de planification de trajectoires, la fonction de coût sert à exprimer les qualités de la trajectoire optimale. L'algorithme d'optimisation est ensuite utilisé pour trouver une solution qui minimise cette fonction de coût et qui s'approche de la solution optimale. La fonction de coût peut être à objectif unique ou multiple. Dans notre implémentation, nous utilisons une fonction à objectifs multiples afin de générer des trajectoires faisables qui minimisent la longueur du trajet, minimisent l'altitude moyenne et évitent les zones dangereuses. Dans ce chapitre, nous présentons la fonction de coût que nous avons développée et que nous utilisons ensuite avec l'AG et l'OEP aux chapitres 4 et 5. Il est important de noter que cette fonction est commune aux deux algorithmes de planification de trajectoires.

3.1 Représentation de l'environnement

Avant de définir les qualités d'une trajectoire sous forme de fonction de coût, il est important d'expliquer la représentation que nous utilisons. Prenons par exemple le scénario illustré à la figure 10.

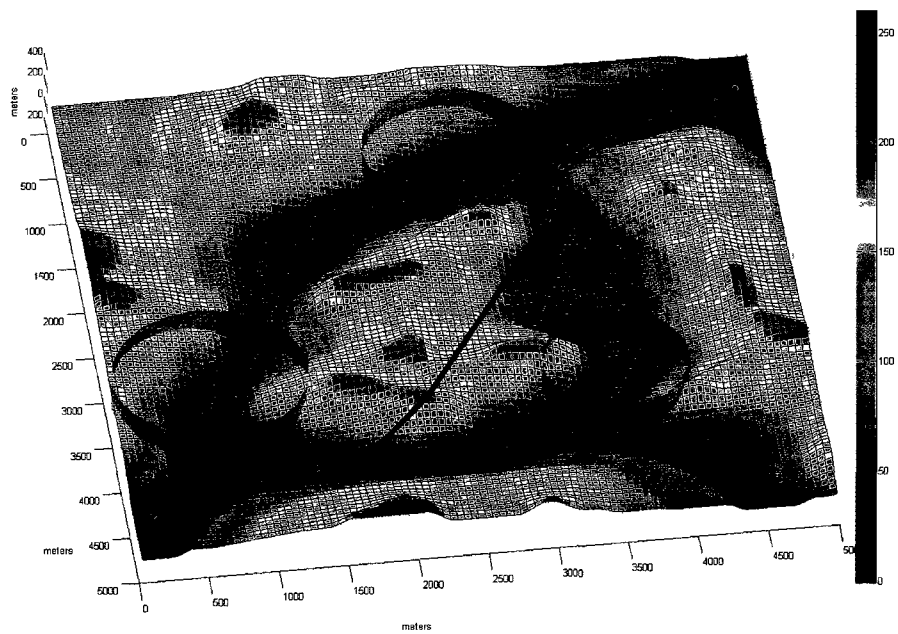
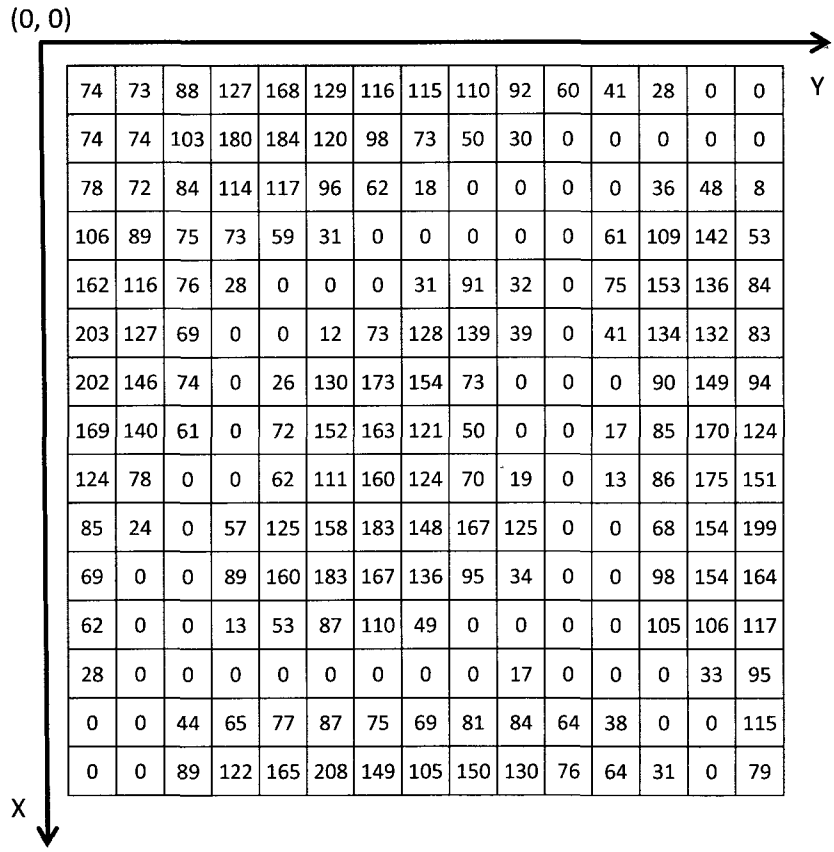


Figure 10. Trajectoire dans un environnement 3D

Nous encodons le terrain sous forme d'une matrice 2D où chaque élément représente l'élévation du terrain en mètres au-dessus du niveau de la mer (ASL). Tel que montré à la figure 11, l'origine du système de coordonnées est placée à l'extérieur de la matrice de sorte que la coordonnée $(1, 1)$ représente le point au centre de la première case en haut à gauche. La résolution utilisée est de 500 unités par 500 unités et l'échelle horizontale (axes des X et des Y), calculée d'après les dimensions réelles du terrain.



74	73	88	127	168	129	116	115	110	92	60	41	28	0	0
74	74	103	180	184	120	98	73	50	30	0	0	0	0	0
78	72	84	114	117	96	62	18	0	0	0	0	36	48	8
106	89	75	73	59	31	0	0	0	0	0	61	109	142	53
162	116	76	28	0	0	0	31	91	32	0	75	153	136	84
203	127	69	0	0	12	73	128	139	39	0	41	134	132	83
202	146	74	0	26	130	173	154	73	0	0	0	90	149	94
169	140	61	0	72	152	163	121	50	0	0	17	85	170	124
124	78	0	0	62	111	160	124	70	19	0	13	86	175	151
85	24	0	57	125	158	183	148	167	125	0	0	68	154	199
69	0	0	89	160	183	167	136	95	34	0	0	98	154	164
62	0	0	13	53	87	110	49	0	0	0	0	105	106	117
28	0	0	0	0	0	0	0	0	17	0	0	0	33	95
0	0	44	65	77	87	75	69	81	84	64	38	0	0	115
0	0	89	122	165	208	149	105	150	130	76	64	31	0	79

Figure 11. Représentation du terrain sous forme d'une matrice 2D

L'environnement dans lequel se déplace le drone peut aussi contenir des zones dangereuses (ou zones d'exclusion aérienne) que doit éviter le drone. Comme à l'équation (5), ces régions cylindriques sont définies sous forme d'une matrice où chaque ligne représente les coordonnées (x, y) du centre de la zone et son diamètre d.

$$zones\ dangereuses = \begin{bmatrix} x_1 & y_1 & d_1 \\ x_2 & y_2 & d_2 \\ \dots & \dots & \dots \\ x_n & y_n & d_n \end{bmatrix} \quad (5)$$

Finalement, comme à l'équation (6), nous représentons une trajectoire sous forme de matrice où chaque ligne équivaut aux coordonnées (x, y, z) d'un point de passage. Ces points sont ordonnés de façon à ce que (x_1, y_1, z_1) soit le point de départ et (x_n, y_n, z_n), le point d'arrivée. La trajectoire consiste donc en une série de segments de droite définis par ces points de passages. Au chapitre 6, lorsque nous lissons la trajectoire, nous remplaçons les intersections par des sections circulaires. Cependant, ces sections circulaires sont tout de même discrétisées sous forme d'une série de segments très courts et la représentation finale de la trajectoire reste la même, mais inclut beaucoup plus de points de passages.

$$trajectoire = \begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ \dots & \dots & \dots \\ x_n & y_n & z_n \end{bmatrix} \quad (6)$$

Notre fonction de coût évalue la faisabilité de la trajectoire pour un drone volant à une vitesse constante. Cette vitesse de croisière est définie dans le fichier de description du drone au tableau 15 (voir chapitre 9). Malgré que nous représentions les trajectoires sous forme de séries de points de passage, il est facile de les définir par une position en fonction du temps grâce à la vitesse constante.

3.2 Critères de faisabilité et d'optimisation

Nous définissons notre fonction de coût sous forme de critères de faisabilité et d'optimisation. Les critères de faisabilité doivent être respectés pour que la trajectoire soit réalisable par le drone. Ceux-ci inclus :

1) la puissance requise pour voler la trajectoire doit toujours être plus petite que la puissance disponible du drone;
2) la trajectoire ne doit jamais descendre plus près du sol que l'altitude minimum de vol tel que spécifié dans le fichier de configuration du drone (voir le tableau 15 au chapitre 9);

3) la quantité de carburant requise pour que le drone vole la trajectoire doit toujours être plus petite que sa quantité initiale de carburant; et
4) la trajectoire finale doit être dynamiquement faisable par le drone (c.-à-d. que les discontinuités de la vitesse aux intersections des segments de droite doivent être corrigées par des sections circulaires faisables).

D'un autre côté, les critères d'optimisation permettre de décrire les caractéristiques recherchées pour améliorer la qualité de la trajectoire faisable. Les critères d'optimisation inclus :

1) minimiser la longueur totale de la trajectoire;
2) minimiser l'altitude moyenne de la trajectoire pour accroître la furtivité du drone; et
3) éviter le plus possible les zones dangereuses ou d'exclusion aérienne.

Puisque nous sommes intéressés à trouver des trajectoires qui maximisent les critères d'optimisation tout en respectant les critères de faisabilité, nous devons intégrer ces deux aspects dans nos algorithmes d'optimisation. Tel qu'expliqué par les auteurs de [38], les approches possibles se regroupent en quatre catégories : pénaliser les solutions non faisables; séparer les solutions faisables de celles non faisables; utiliser des opérations spéciales pour assurer la faisabilité des solutions générées; ou adopter une méthode hybride. La première approche nécessite une analyse supplémentaire afin que la pénalité soit correctement appliquée. La deuxième approche est la plus simple, mais limite en quelque sorte l'optimisation. Il se pourrait qu'une solution soit quasi optimale, mais aussi infaisable à cause d'une simple imperfection. Or, si cette trajectoire est séparée des solutions faisables, elle ne peut influencer la qualité de la solution finale. La troisième méthode est plus difficilement applicable dans notre situation. Si l'on avait utilisé une représentation par graphe, il aurait été possible de vérifier la faisabilité des arêtes et d'appliquer cette méthode pour générer des solutions qui utilisent uniquement des arêtes faisables. Cependant, l'utilisation de graphe nécessite une puissance de calcul considérable.

Dans notre implémentation, nous incluons tous les critères de faisabilité et d'optimisation dans la fonction de coût. Nous utilisons la première méthode décrite ci-dessus et ajoutons une pénalité aux solutions non faisables afin de les différencier des solutions faisables. Notre approche permet la participation des solutions infaisables à l'optimisation des solutions faisables.

3.3 Fonction de coût

Nous définissons la fonction de coût comme suit :

$$Coût = C_{longueur} + C_{altitude} + C_{zones\ dangereuses} + C_{puissance} + C_{collision} + C_{carburant} + C_{lissage} + C_{longueur\ min\ des\ segments} \quad (7)$$

où $C_{longueur}$ est le terme associé à la longueur de la trajectoire; $C_{altitude}$ celui associé à l'altitude moyenne; $C_{zones\ dangereuses}$ celui associé à la violation des zones dangereuses; $C_{puissance}$ celui associé à une puissance nécessaire plus grande que celle disponible; $C_{collision}$ celui associé aux collisions entre la trajectoire et le terrain; $C_{carburant}$ celui associé au manque de carburant; $C_{lissage}$ celui associé aux connexions impossibles; et $C_{longueur\ min\ des\ segments}$ celui associé aux segments de la trajectoire qui sont plus petits qu'une longueur arbitraire. Comme nous l'expliquons plus tard dans cette section, ce dernier terme est utilisé afin d'estimer la possibilité du lissage lorsque $C_{lissage}$ n'est pas calculé. Le coût total d'une trajectoire est donc la sommation de ces huit termes, chacun représentant une qualité recherchée. Ces termes sont normalisés de façon particulière afin d'assurer que le coût totale d'une trajectoire faisable soit toujours plus petit que celui d'une trajectoire violant une ou plusieurs zones dangereuses. Similairement, le coût d'une trajectoire violant une ou plusieurs zones dangereuses est toujours plus petit que celui d'une trajectoire infaisable. Cette normalisation permet d'ordonner les solutions par ordre de qualité d'après leur coût, mais aussi d'identifier les caractéristiques d'une solution simplement en se référant au tableau 1. Par exemple, si le coût total d'une trajectoire est de 4.53, nous savons que cette dernière est dynamiquement faisable, mais traverse au moins une zone dangereuse.

Tableau 1. Signification de la valeur du coût total d'une trajectoire

Coût	Signification
[0, 2[	La trajectoire est faisable.
[2, 5[	La trajectoire est faisable, mais traverse au moins une zone dangereuse
[5, 11[	La trajectoire est infaisable dû au manque de puissance, mais peut possiblement être rendu faisable par l'addition de sections hélicoïdales au moment du lissage; et La trajectoire traverse peut-être une ou plusieurs zones dangereuses.
[11, 58[	La trajectoire est infaisable avec une ou plusieurs conditions violées; Elle traverse peut-être une ou plusieurs des zones dangereuses, et Elle dépasse peut-être la puissance disponible du drone.

3.3.1 Coût associé à la longueur de la trajectoire

Dans l'équation (7), le terme associé à la longueur de la trajectoire représente un critère d'optimisation et est défini comme suit :

$$C_{longueur} = 1 - \left(\frac{L_{P1P2}}{L_{traj}}\right) \qquad donc \;\; C_{longueur} \in [0,1[\qquad (8)$$

où L_{P1P2} est la longueur du segment de droite reliant le point de départ $\boldsymbol{P_1}$ au point d'arrivée $\boldsymbol{P_2}$ et L_{traj} est la somme des longueurs des segments de droite formant la trajectoire. Dans cette équation, L_{traj} est une approximation de la longueur de la trajectoire finale puisque cette dernière inclut aussi des connexions circulaires reliant chacun des segments (voir chapitre 6). Dans le cas d'une connexion par arc de cercle (voir figure 12), la longueur de l'arc C est ignorée puisque le trajet $\boldsymbol{C_1PC_2}$ est toujours plus long (résultant en une surestimation de la longueur de la trajectoire). Dans le cas d'une connexion par cercle sur plateau horizontal (voir figure 13), la circonférence du cercle C est ajoutée à la longueur des segments, résultant encore ici en une surestimation de la longueur. Puisque la longueur des segments de droite est habituellement beaucoup plus grande que la longueur des connexions, cette estimation est correcte. L'utilisation d'un ratio dans l'équation (8) permet de normaliser le coût associé à la longueur. Une trajectoire proche de la droite reliant $\boldsymbol{P_1}$ à $\boldsymbol{P_2}$ a un coût légèrement supérieur à 0. À l'opposé, une trajectoire très grande a un coût qui tend vers 1. Minimiser cette fonction résulte donc à minimiser la longueur totale de la trajectoire.

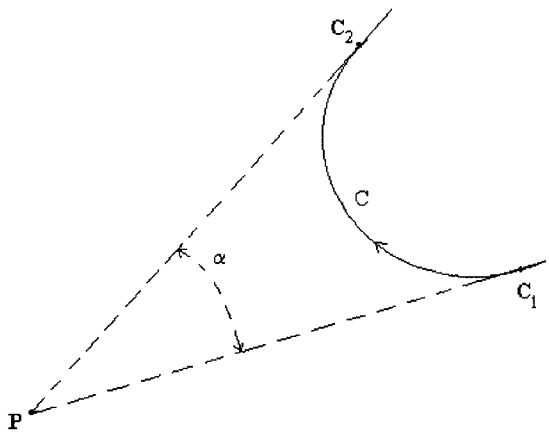


Figure 12. Connexion de deux segments de droite par arc de cercle (reproduit de [37])

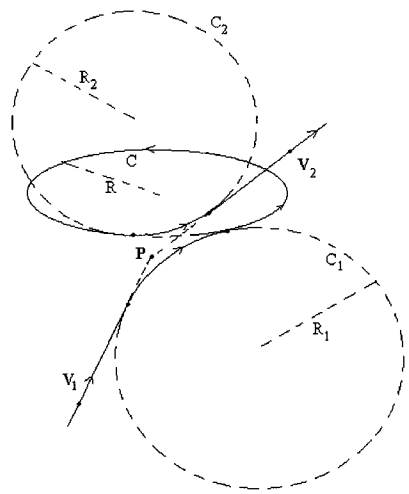


Figure 13. Connexion de deux segments de droite par cercle sur plateau horizontal (reproduit de [37])

3.3.2 Coût associé à l'altitude moyenne de la trajectoire

Dans l'équation (7), le coût associé à l'altitude moyenne de la trajectoire représente un critère d'optimisation et est défini comme suit :

$$C_{altitude} = \left(\frac{Z_{max} - A_{traj}}{Z_{max} - Z_{min}}\right) \qquad donc\ C_{altitude} \in [0,1[\tag{9}$$

où A_{traj} est l'altitude moyenne de la trajectoire, Z_{min} est la plus basse élévation de la carte et Z_{max}, la plus haute. Puisque le drone ne doit jamais descendre plus bas qu'une certaine altitude au-dessus du sol (tel que spécifiée dans le fichier de caractéristiques du drone au tableau 15 au chapitre 9), nous ajoutons cette altitude minimum à Z_{min} et Z_{max}. L'utilisation de Z_{min} et Z_{max} assure que l'évaluation du coût soit correcte, peu importe l'élévation du terrain. Ceci ne serait pas possible si l'altitude ASL avait été utilisée comme le font les auteurs de [14], [15] et [16]. De plus, l'équation (9) est définie sous forme de ratio afin de normaliser le coût associé à l'altitude moyenne à une valeur entre 0 et 1.

3.3.3 Coût associé à la violation des zones dangereuses

Dans l'équation (7), le coût associé à la pénétration d'une zone dangereuse représente un critère d'optimisation et est défini comme suit :

$$C_{zones\ dangereuses} = \begin{cases} 0, & L_{danger} = 0 \\ 2 + \left(\frac{L_{danger}}{\sum_{i=1}^{n} d_i}\right), & L_{danger} > 0 \end{cases} \qquad donc\ C_{z.d.} \in 0 \cup [2,3[\tag{10}$$

où L_{danger} est la distance que parcourt la trajectoire à l'intérieur des zones dangereuses et d_i est le diamètre de la zone dangereuse i. Cette définition est normalisée à des valeurs entre 2 et 3 et tient compte du scénario. Par exemple, prenons le cas où une trajectoire passe par le centre d'une zone dangereuse. Si le scénario n'inclut qu'une seule zone, le coût associé est de 3, soit la valeur maximale, et décrit bien la sévérité de franchir l'unique zone dangereuse. Par contre, si le scénario inclut 10 zones de mêmes diamètres, la violation est relativement moins sévère et le coût est alors évalué à 2.1. Cette valeur est toutefois plus grande que 2. La trajectoire est donc considérée comme étant moins bonne que n'importe quel chemin faisable qui évite les zones dangereuses.

3.3.4 Coût associé au manque de puissance disponible

Dans l'équation (7), le coût associé à une puissance nécessaire plus grande que celle disponible représente un critère de faisabilité et est défini comme suit :

$$C_{puissance} = \begin{cases} 0, & L_{puissance} = 0 \\ 5 + \left(\frac{L_{puissance}}{L_{traj}}\right), & L_{puissance} > 0 \end{cases} \quad donc\ C_{puis.} \in 0 \cup [5,6[\tag{11}$$

où $L_{puissance}$ est la somme des longueurs des segments de droite nécessitant une puissance supérieure à celle du drone et L_{traj} est la longueur totale de la trajectoire. $C_{puissance}$ est normalisé à des valeurs entre 5 et 6. Nous avons arbitrairement choisi ces valeurs pour identifier une trajectoire qui ne respecte pas la puissance maximale disponible comme étant moins bonne que n'importe quelle trajectoire faisable peu importe sa longueur, son altitude moyenne et son degré de violation des zones dangereuses. Malgré qu'une telle trajectoire est infaisable, elle peut possiblement être corrigée par l'ajout de sections hélicoïdales tel que présenté au chapitre 6. C'est pourquoi le coût assigné aux trajectoires dépassant la puissance disponible est toutefois moindre que celui assigné à celles qui, par exemple, heurtent le sol.

Puisque nous utilisons des trajectoires construites de segment de droite, il est possible de facilement calculer la puissance maximale requise pour voler chacun des segments. Tel que démontré dans [37], il faut vérifier que la puissance disponible du drone P_A est supérieure à la puissance requise P_R à chacune des extrémités du segment pour que ce segment soit faisable. On utilise les formules présentées dans [39] pour calculer les puissances requises et disponibles.

Prenons le segment $\boldsymbol{P_1P_2}$, la puissance requise P_{RP1} au point $\boldsymbol{P_1}$ est calculée à l'aide de l'équation (12) en utilisant le poids de l'avion W_1 et l'altitude ASL h_1 au point $\boldsymbol{P_1}$. La puissance requise P_{RP2} au point $\boldsymbol{P_2}$ est évaluée similairement en utilisant W_2 et h_2. Puisque P_R dépend du poids de l'avion, il est nécessaire de calculer la quantité de carburant restant à chacun des points de passage de la trajectoire avant de pouvoir évaluer P_R. Ce calcul est discuté à la section 3.3.6.

La puissance disponible P_A au point $\boldsymbol{P_1}$ et $\boldsymbol{P_2}$ est évaluée d'après l'équation (13).

$$P_R(W,h,\theta) = \frac{\eta\, g}{G}\left(K_1\, \rho(h)\; V_c^3 + W\, V_c \sin(\theta) + K_2 \frac{cos^2(\theta) W^2}{\rho(h)\; V_c}\right) \tag{12}$$

$$P_A(h) = P_{AS}\left(\frac{\rho(h)}{\rho_S}\right) \tag{13}$$

où

$$K_1 = \frac{S\, C_{D0}}{2} \tag{14}$$

$$K_2 = \frac{2}{\pi\, e\, b^2} \tag{15}$$

$$G = \eta\, g - c\, RAC\, V_c^2 \tag{16}$$

$$\rho(h) = \rho_S\left(1 - \frac{|a|h}{T_S}\right)^{4.2433} \tag{17}$$

Les définitions pour les symboles utilisés dans ces équations se trouvent dans la liste des symboles au début de ce document.

3.3.5 Coût associé aux collisions avec le terrain

Une trajectoire qui entre en collision avec le terrain est nécessairement infaisable. En fait, nous considérons toute trajectoire qui descend plus bas que l'altitude minimum prescrit au-dessus du terrain (telle que définie dans le fichier de caractéristiques du drone au tableau 15 au chapitre 9) comme étant infaisable. Dans l'équation (7), le coût associé à cette condition représente un critère de faisabilité et est défini comme suit :

$$C_{terrain} = \begin{cases} 0, & N_{terrain} = 0 \\ 11 + \left(\frac{N_{terrain}}{N_{total}}\right), & N_{terrain} > 0 \end{cases} \qquad donc\ C_{terrain} \in 0 \cup [11, 12[\tag{18}$$

où $N_{terrain}$ représente la distance parcourue par la trajectoire en dessous de l'altitude minimum spécifiée et N_{total} représente la longueur totale de la trajectoire. La valeur de ce terme est ici normalisée entre 11 et 12. Il est important de noter que, contrairement aux équations précédentes, nous employons ici la lettre N pour représenter les distances. En fait, nous utilisons l'algorithme de tracé de ligne de Bresenham [40] pour générer la liste de toutes les cases de la

carte d'élévation digitale par lesquelles passe la trajectoire. Nous comparons ensuite l'altitude de la trajectoire à celle du terrain pour chacune des cases. Nous comptons alors le nombre de cases où la trajectoire vole sous l'altitude minimum ($N_{terrain}$) et le nombre total de cases (N_{total}) pour déterminer la valeur de $C_{terrain}$.

3.3.6 Coût associé au manque de carburant

Dans l'équation (7), le coût associé à la consommation de carburant représente un critère de faisabilité et est défini comme suit :

$$C_{carburant} = \begin{cases} 0, & F_{traj} \leq F_{init} \\ 12 - \left(\frac{F_{P1P2}}{F_{traj}}\right), & F_{traj} > F_{init} \end{cases} \qquad donc\ C_{carb.} \in 0 \cup [11, 12[\tag{19}$$

où F_{P1P2} est la quantité de carburant nécessaire si le drone volait le segment de droite fictif reliant le point de départ $\boldsymbol{P_1}$ au point d'arrivée $\boldsymbol{P_2}$; F_{traj} est la quantité de carburant nécessaire pour voler la trajectoire; et F_{init}, la quantité initiale de carburant. Dans notre implémentation, nous pénalisons les trajectoires infaisables dues au manque de carburant et assignons un coût de 0 lorsque la quantité de carburant est suffisante. Il serait facile de calculer le coût de toutes les trajectoires sans augmenter le temps d'exécution puisque la quantité de carburant requis doit être évaluée pour déterminer la faisabilité. Cependant, nous pensons qu'il est inutile d'augmenter la complexité de la fonction de coût pour les trajectoires faisables. Les termes $C_{longueur}$ et $C_{altitude}$ favorisent déjà les trajectoires qui minimisent la quantité de carburant.

Dans [39], Labonté dérive des formules de consommation de carburant et présente une approximation valide et précise du poids (N) de carburant requis pour qu'un avion vole un segment de droite à vitesse constante V_c. Nous utilisons cette approximation, telle qu'écrite à l'équation (20), pour calculer le poids de carburant F nécessaire au vol de chacun des segments formant la trajectoire.

$$F = t \left(\frac{K_1\, \rho(h_m)\, V_c^3 + W_i\, V_c \sin(\theta) + K_2 \frac{cos^2(\theta) W_i^2}{\rho(h_m)\, V_c}}{\frac{G}{c\, g} + h_m + 2\, K_2 \frac{cos^2(\theta) W_i \frac{t}{2}}{\rho(h_m)\, V_c}} \right) \tag{20}$$

où K_1 est défini à l'équation (14); K_2, à l'équation (15); G, à l'équation (16); et $\rho(h_m)$, à l'équation (17). W_i est le poids du drone (incluant son carburant) au début du segment, h_m, l'altitude moyenne du segment et t, le temps nécessaire pour parcourir le segment à vitesse constante V_c. Les définitions des autres symboles utilisés se trouvent dans la liste des symboles au début de ce document.

3.3.7 Coût associé aux connexions impossibles

Dans l'équation (7), le coût associé au lissage possible de la trajectoire représente un critère de faisabilité et est défini comme suit :

$$C_{lissage} = \begin{cases} 0, & N_{impossible} = 0 \quad donc\ C_{lissage} \in 0 \\ 11 + \left(\frac{N_{impossible}}{N_{total}}\right), & N_{impossible} > 0 \qquad\qquad \cup [11, 12[\end{cases} \tag{21}$$

où N_{total} représente le nombre total de connexions nécessaires et $N_{impossible}$, le nombre de connexions impossibles. Une discussion complète sur le lissage de la trajectoire par connexions circulaires est présentée au chapitre 6. Il est toutefois important de spécifier que le temps de calcul nécessaire pour vérifier la faisabilité d'une connexion (sans même la générer) est très long. En fait, ce calcul prend 2.3 fois plus de temps que le calcul de tous les autres termes de la fonction de coût. Pour cette raison, notre implémentation de la fonction de coût permet de calculer ou non le terme associé au lissage de la trajectoire. Lorsque ce terme n'est pas évalué, une valeur de 12 est automatiquement assignée à $C_{lissage}$. Cette option permet de négliger le calcul au début du processus itératif de l'AG et de l'OEP, réduisant ainsi le temps d'exécution; et de l'inclure à la fin, assurant la qualité de la solution finale.

3.3.8 Coût associé aux segments trop courts

En réalité, notre implémentation n'ignore pas totalement la faisabilité du lissage en début d'exécution puisque nous utilisons aussi un terme associé à la longueur minimale des segments formant la trajectoire. Or, nous avons déterminé par expérimentation que la probabilité qu'une connexion soit faisable est plus élevée lorsque les deux segments à connecter ont une longueur plus grande que le rayon de courbure minimum R_{min} de l'arc de cercle. La valeur de R_{min} est spécifiée dans le fichier de description du drone (voir tableau 15 au chapitre 9). Dans l'équation (7), le coût associé aux segments trop courts est défini comme suit :

$$C_{l.\ min\ des\ segments} = \begin{cases} 0, & \forall\, i, \min(l_i) \geq R_{min} \\ 11, & \exists\, i, \min(l_i) < R_{min} \end{cases} \qquad donc \quad C_{l.\ min.} \in 0 \cup 11 \tag{22}$$

où i est l'indice des segments formant la trajectoire.

La figure 14 montre la structure de notre implémentation MATLAB de la fonction de coût. Sur l'illustration, chaque boîte représente une fonction. Celles qui se terminent « .m » sont définies dans leur propre fichier, tandis que les autres se trouvent dans le fichier de la fonction supérieure.

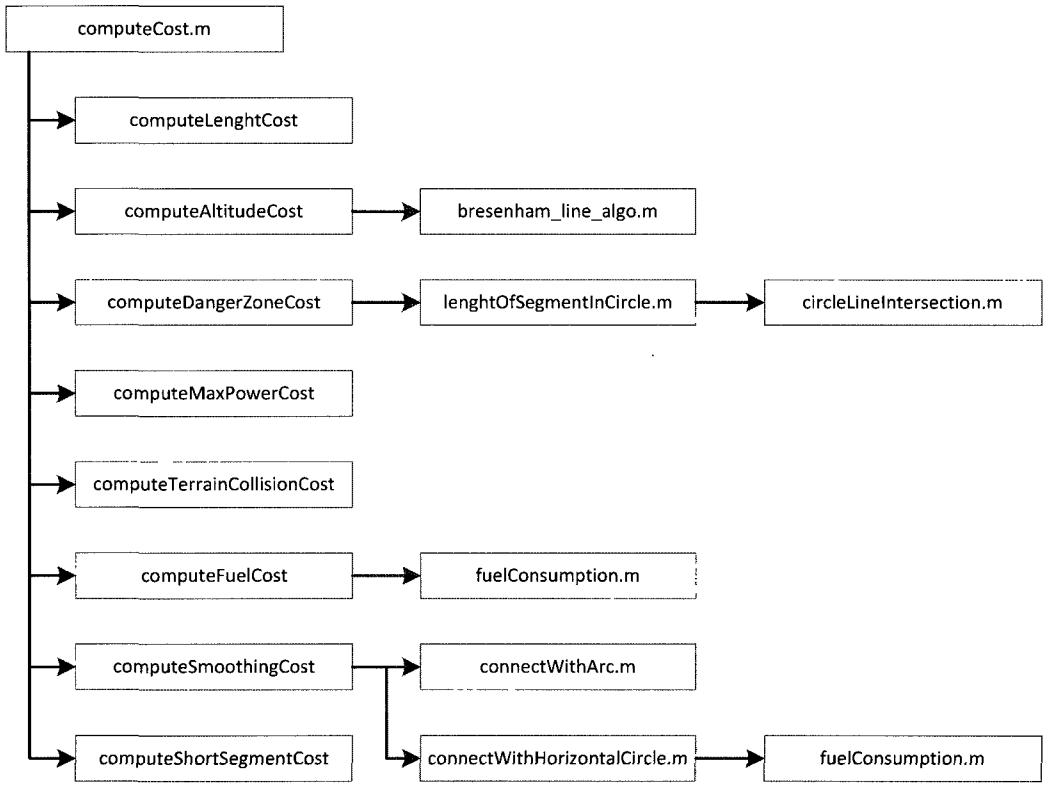


Figure 14. Diagramme de la structure de notre fonction de coût

Nous avons défini dans ce chapitre une fonction de coût qui représente les qualités d'une trajectoire optimale : minimise la longueur du trajet, minimise l'altitude moyenne, évite les zones dangereuses, tout en étant faisable. Ces propriétés s'appliquent à notre scénario, mais pourraient bien être modifiées pour une application différente. Pour l'implémentation de notre planificateur de trajectoire pour les drones, nous employons cette fonction de coût sans modification avec l'AG et l'OEP. Ces deux algorithmes non déterministes permettent de trouver des solutions qui minimisent la fonction de coût et qui s'approchent donc de notre définition de trajectoire optimale.

Chapitre 4

Algorithme génétique

L'algorithme génétique (AG) est l'approche non déterministe la plus utilisée pour résoudre le problème de planification de trajectoires [19]. C'est pourquoi nous l'avons choisi comme un des moteurs d'optimisation dans notre implémentation (l'autre étant l'OEP). Tel que mentionné au chapitre 2, l'AG original est très simple, mais doit toujours être adapté au problème en question afin d'être efficace [41]. Dans ce chapitre, nous présentons en détail notre implémentation MATLAB de l'AG pour notre module de planification de trajectoires ainsi que les optimisations apportées pour minimiser le temps de calcul.

4.1 Fonctionnement général

L'AG fait partie de la famille des algorithmes évolutionnistes et simule la théorie génétique de l'évolution naturelle de Darwin pour résoudre des problèmes d'optimisation. Cette méthode est particulièrement utile lorsqu'une approche déterministe ne peut être employée pour trouver une solution en temps polynomial tel que pour la planification de trajectoires pour les drones dans un environnement complexe. Dans un monde naturel, les individus les mieux adaptés à leur environnement ont une plus grande chance de survie et une meilleure probabilité de reproduction. En fait, ceux-ci se reproduisent habituellement avec d'autres individus supérieurs. Les gènes des parents sont alors transmis aux enfants qui héritent ainsi certaines caractéristiques de chacun de leurs parents. Au cours des générations, les individus évoluent et s'adaptent continuellement à leur environnement. Le processus évolutif inclut aussi des mutations génétiques. Celles qui favorisent l'individu ont une meilleure probabilité d'être transmises aux générations futures. Tout comme la théorie de Darwin, l'AG simule l'évolution de solutions qui s'adaptent à un problème spécifique. Dans l'AG, les chromosomes représentent des solutions et la fonction de coût décrit l'environnement. Le niveau d'adaptation d'un individu s'évalue avec cette fonction de coût. Au cours du processus itératif qui inclut la sélection des parents, la reproduction, les mutations et le remplacement des générations antérieures, les solutions se transforment pour minimiser la fonction de coût et améliorer leur qualité.

Pour le développement de notre module de planification de trajectoires pour les drones en MATLAB, nous utilisons l'AG pour simuler l'évolution d'une population de trajectoires afin

qu'elles s'adaptent et minimisent la fonction de coût défini au chapitre précédent. La représentation utilisée est la même qu'au chapitre 3 (chaque solution est représentée sous forme d'une suite de coordonnées (x, y, z) définissant les points de passage de la trajectoire). Le fonctionnement général de notre implémentation de l'AG est illustré à la figure 15.

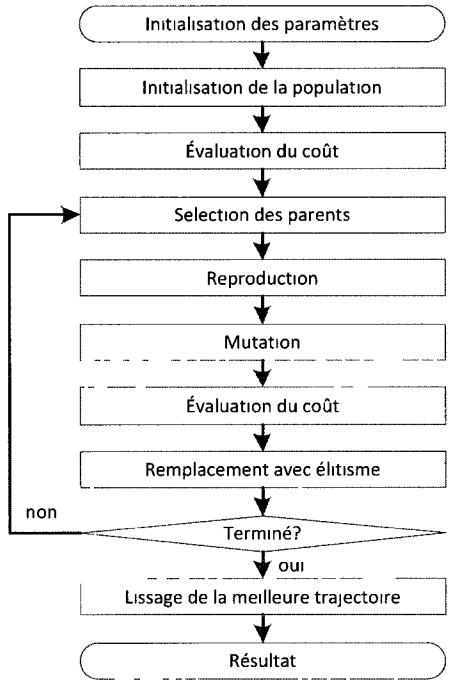


Figure 15. Diagramme de fonctionnement notre AG pour la planification de trajectoires

4.2 Détails d'implémentation

Nous discutons dans cette section des détails d'implémentation pour chacune des étapes illustrées à la figure 15.

1) **Initialisation des paramètres :** La première étape consiste à définir les paramètres d'entrée de notre AG. À l'exception de quelques-uns identifiés par un astérisque au tableau 2, ceux-ci sont tous passés en paramètre à la fonction. Nous avons inclus dans ce tableau des valeurs suggérées (que nous avons déterminées expérimentalement) pour les variables servant à la configuration de l'AG.

Tableau 2. Liste des paramètres de configuration de notre AG

Paramètres	Valeurs suggérées	Descriptions
Taille de la population	128	Nombre de chromosomes (ou solutions) utilisés pour le processus évolutif.
Nombre de générations	200	Nombre de générations (ou itérations) simulées. L'exécution de l'AG se termine lorsque ce nombre est atteint.
Longueur minimum des chromo.	6	Le nombre minimum de points de passage d'une trajectoire.
Longueur maximum des chromo.	12	Le nombre maximum de points de passage d'une trajectoire.
*Taux de mutations	80 %	La probabilité qu'un enfant subisse une mutation.
*Taux de survie (élitisme)	10 %	Le pourcentage des parents qui sont gardés à la nouvelle génération. Ceci permet de remplacer les pires enfants par les meilleurs parents.
$*\alpha, \beta$	2 et 0	Paramètres utilisés pour la sélection des parents d'après leur rang suivant une fonction de probabilité linéaire.
Intervalle d'échantillonnage initial	50 %	Représente la grandeur du voisinage en fonction des dimensions du terrain dans lequel un point de passage de la trajectoire peut être déplacé lors d'une mutation au début de l'exécution de l'AG.
Intervalle d'échantillonnage final	10 %	Comme à la définition précédente, mais en fin d'exécution de l'AG. En fait, le taux diminue linéairement de la valeur initiale à la valeur finale.
Vérifier le coût du lissage à partir de X % du processus évolution	80 %	Paramètre servant à réduire le nombre de générations auxquelles le coût lié au lissage de la trajectoire est évalué afin d'accélérer l'exécution de l'algorithme.
Vérifier le coût du lissage pour les X % meilleurs chromosomes	25 %	Paramètre servant à réduire le nombre de solutions dont le coût lié au lissage de la trajectoire est évalué afin d'accélérer l'exécution de l'algorithme.
Carte d'élévation digitale	s.o.	Une matrice 2D où chaque case contient l'élévation du terrain.
Échelle horizontale	s.o.	Le nombre de mètres que représente une case de la matrice 2D. Les échelles pour l'axe des X et l'axe des Y doivent être identiques puisqu'un seul paramètre est utilisé. L'échelle pour l'axe des Z est toujours de 1 m par unité.
Point de départ	s.o.	Les coordonnées (x, y, z) du point de départ de la trajectoire d'après de système de coordonnées défini à la section 3.1.
Point d'arrivée	s.o.	Les coordonnées (x, y, z) du point d'arrivée de la trajectoire d'après de système de coordonnées défini à la section 3.1.
Zones dangereuses	s.o.	Une série de coordonnées (x_i, y_i, z_i) définissant les zones dangereuses comme à la section 3.1.
Caractéristique du drone	s.o.	Une structure de donnée contenant toutes les caractéristiques du drone nécessaire pour que la fonction de coût puisse évaluer la puissance requise, la puissance disponible, la consommation de carburant, le rayon de courbure minimale, etc. (voir tableau 15 au chapitre 9)
* Contrairement aux autres variables qui sont passées en paramètre à la fonction AG, ces variables sont définies dans la fonction elle-même et ne peuvent être modifiées à l'exécution.		

2) **Initialisation de la population :** La population est encodée sous forme d'une structure de cellules MATLAB où chaque cellule contient un chromosome (ou trajectoire). Les chromosomes sont représentés sous forme de matrice de dimension m x 3 où m est le nombre de points de passage (x, y, z) de la trajectoire. La longueur initiale de chaque chromosome est définie aléatoirement d'après les paramètres d'initialisation. Les points formant les trajectoires sont aussi initialisés aléatoirement dans l'espace de recherche. Ces points ne sont donc pas ordonnés et les trajectoires initiales contiennent fort probablement plusieurs boucles. L'utilisation d'une structure de cellules permet de modifier la longueur des chromosomes au cours de l'exécution. Comme discuté plus loin, notre implémentation de l'opération de reproduction et de mutation permet de faire varier la longueur des trajectoires.
3) **Évaluation du coût :** L'évaluation du coût se fait séquentiellement pour chacun des chromosomes d'après la fonction définie au chapitre 3. Puisque les opérateurs génétiques peuvent générer un chromosome contenant un point de passage qui se répète, nous supprimons toujours les doublons avant d'évaluer le coût.
4) **Sélection des parents :** tel que décrit dans [17], une méthode commune consiste à sélectionner aléatoirement deux parents où la probabilité qu'un individu soit choisi est inversement proportionnelle à son coût. Le processus est répété jusqu'à ce le nombre de parents soit égale à la taille de la population. Le nombre prévu de fois n_i qu'un individu i soit sélectionné est défini par :

$$n_i = p_i * popsize = \left\lfloor popsize * \frac{\frac{1}{C_i}}{\sum_{i=1}^{popsize} \frac{1}{C_i}} \right\rfloor \quad (23)$$

où p_i est la probabilité que le chromosome i soit sélectionné, $popsize$ est la taille de la population et C_i est le coût du chromosome i. Les auteurs de [17] définissent alors l'erreur sélective comme étant la différence entre le nombre réel de fois qu'un chromosome est sélectionné et n_i. Puisqu'il se peut qu'un chromosome de bonne qualité ne soit pas sélectionné, cette méthode inclut un biais sélectif. De plus, nous ne pouvons utiliser cette approche à cause des pénalités arbitraires que nous donnons aux solutions non faisables au chapitre 3. Une bonne solution rendue infaisable par une petite imperfection recevrait alors une probabilité de sélection exagérément plus petite que n'importe quelle solution faisable.

Afin d'éviter le biais sélectif, notre implémentation emploie l'échantillonnage universel stochastique. Tel que défini dans [17], cette méthode utilise aussi une sélection proportionnelle à la qualité des individus, mais choisit tous les parents à l'aide d'une seule valeur aléatoire. Comme illustré à la figure 16 pour une population de n individus (ici $n = 4$), le procédé consiste à faire tourner une roulette à n flèches et à sélectionner les individus sous les flèches. Si plus d'une flèche s'arrêtent sur un individu, ce dernier est choisi plusieurs fois.

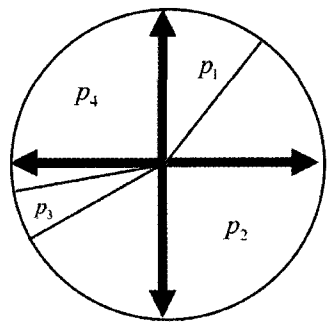


Figure 16. Échantillonnage universel stochastique (reproduit de [17])

Pour résoudre le problème relié aux valeurs disproportionnées des coûts (suite à notre fonction de coût au chapitre 3), nous ordonnons les chromosomes d'après leur qualité et définissons linéairement la probabilité de sélection d'après leur rang. Toujours selon [17], la probabilité p_i est calculée comme suit :

$$p_i = \frac{\alpha + \frac{rang_i}{popsize - 1}(\beta - \alpha)}{popsize} \tag{24}$$

où α et β sont des paramètres de contrôle tel que définis au tableau 2. Lorsque $\alpha = 2$ et $\beta = 0$ (comme nous le faisons), le meilleur individu a deux fois plus de chance d'être sélectionné que l'individu moyen. Évidemment, il faut toujours que $\alpha + \beta = 2$ afin que $\sum_{i=1}^{popsize} p_i = 1$.

5) **Reproduction :** Après avoir sélectionné les parents, nous effectuons une permutation aléatoirement de l'ensemble et prenons dans l'ordre deux parents pour générer deux enfants. La méthode de reproduction utilisée est celle du croisement en un point unique [17]. Tel qu'illustrée à la figure 17, elle consiste à choisir un point de segmentation chez le parent 1 et un autre chez le parent 2 pour ensuite unir la première partie du parent 1 à la deuxième du parent 2. Le choix du point de segmentation chez le

parent 1 est entièrement aléatoire. Dans le cas du parent 2, le choix reste stochastique, mais restreint de façon à respecter la longueur minimum et maximum de l'enfant. Le processus est répété pour générer un deuxième enfant. À la fin du processus de reproduction, nous avons un nombre égal de parents et d'enfants.

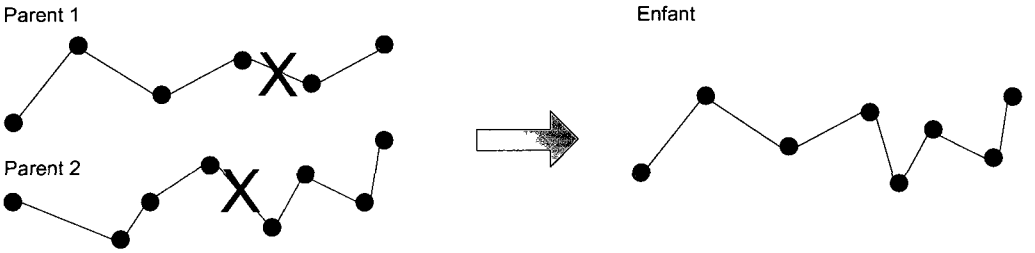


Figure 17. Reproduction par croisement en un point unique

6) **Mutation :** Les mutations servent à diversifier la population et à mieux explorer l'espace de recherche. Si l'AG n'incluait pas de mutation, la solution finale serait restreinte à un sous-ensemble des points formant les trajectoires initiales. L'implémentation des mutations varie beaucoup d'un auteur à l'autre. Certains auteurs, tel que [15] et [16], utilisent des mutations complexes qui analysent les trajectoires et tentent de les améliorer. Cette approche réduit en quelque sorte l'exploration de l'espace de recherche et va contre l'aspect stochastique des mutations biologiques. Dans notre AG, nous n'implémentons que trois mutations : l'addition, la soustraction et la modification d'un point de passage. Le choix de la mutation est fait aléatoirement. Dans le cas de l'addition et de la modification, la nouvelle position du point de passage se trouve dans un voisinage qui est réduit linéairement tout au long du processus itératif d'après les paramètres au tableau 2. Les trois mutations sont illustrées à la figure 18.

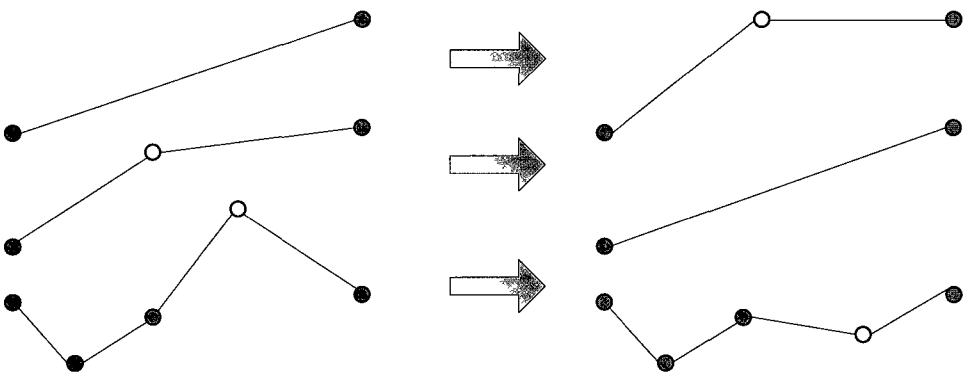


Figure 18. Dans l'ordre : mutation d'addition, mutation de soustraction et mutation de modification

7) **Évaluation du coût des enfants :** L'évaluation du coût se fait séquentiellement pour chacun des chromosomes d'après la fonction définie au chapitre 3.
8) **Remplacement :** Tout comme dans l'AG original, nous remplaçons les parents par la nouvelle génération. Cependant, afin d'améliorer la performance de l'algorithme [42], nous utilisons le principe d'élitisme où certains des meilleurs parents survivent à la génération suivante pour remplacer les pires enfants. Le nombre de chromosomes qui participent à l'élitisme est défini par un paramètre dans le tableau 2.
9) **Critère de terminaison :** Les auteurs de [17] proposent trois façons possibles de terminer l'exécution de l'AG. La première consiste à arrêter l'évolution lorsqu'un certain coût est atteint. Cette méthode peut difficilement être utilisée dans notre cas puisque nous ne connaissons pas le coût de la solution désirée. La trajectoire optimale sur un terrain montagneux peut avoir un coût beaucoup plus élevé que celle sur un terrain plat. La deuxième méthode consiste à arrêter l'exécution lorsque le coût de la meilleure solution ne diminue plus. Dans le cas d'une implémentation en temps réel, cette méthode est à éviter puisque le temps d'exécution ne peut être déterminé. Nous devons donc utiliser la troisième méthode qui consiste à exécuter l'algorithme pour un temps fixe, que nous implémentons ici par un nombre fixe de générations.
10) **Lissage :** La trajectoire finale obtenue par l'AG est définie comme une série de segments de droite. Ce trajet contient des sommets que nous lissons par des arcs de cercle et des cercles sur plateaux horizontaux. Cette méthode provient de [37] et est décrite en détail au chapitre 6.

4.3 Structure de notre implémentation de l'AG

Afin de gérer la complexité de l'implémentation de l'AG en MATLAB, nous avons utilisé le paradigme de programmation structurée et avons divisé notre programme en plusieurs fonctions tel que montré à la figure 19. Sur l'illustration, chaque boîte représente une fonction. Celles qui se terminent « .m » sont définies dans leur propre fichier, tandis que les autres se trouvent dans le fichier de la fonction supérieure.

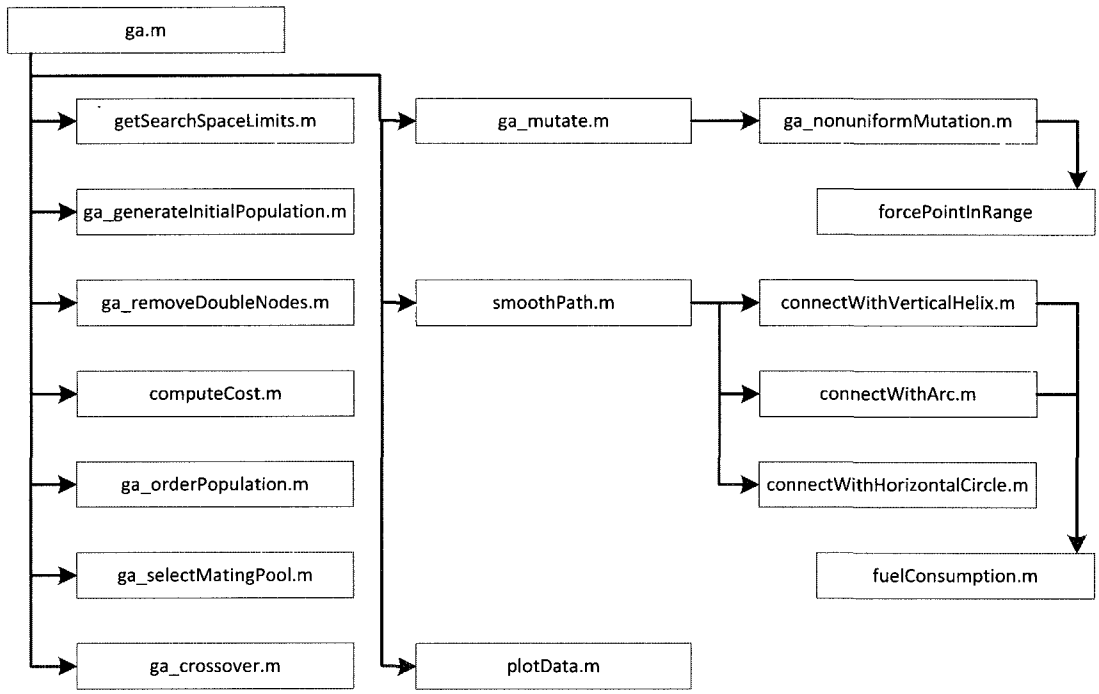


Figure 19. Diagramme de la structure de notre AG

4.4 Optimisations apportées

Puisque nous sommes intéressés à l'utilisation en temps réel de notre algorithme de planification de trajectoires, nous avons analysé le temps d'exécution de chacune des fonctions du programme afin d'optimiser leur performance. Tel que listé au tableau 3, le calcul de la fonction de coût est de loin la fonction nécessitant le plus long temps de calcul. Pour accélérer son exécution, nous avons apporté deux optimisations.

La première optimisation consiste à ajouter des variables temporaires dans les matrices des chromosomes. Puisque la reproduction et les mutations ne modifient qu'un ou deux points de la trajectoire, le calcul de la fonction de coût est quelque peu redondant d'une génération à l'autre. Il est donc possible de sauvegarder des sous-produits de ce calcul dans l'encodage même du chromosome. Au lieu d'inclure uniquement les coordonnées (x, y, z) de chacun des points de passage $\boldsymbol{P_i}$, nous ajoutons aussi 8 valeurs temporaires relatives au segment de droite suivant le point en question, soit le segment $\boldsymbol{P_i P_{i+1}}$.

Chaque ligne de la matrice représentant un chromosome devient alors :

$$[x_i \quad y_i \quad z_i \quad l_i \quad \mathbb{P}_{XY}(l_i) \quad N_{total_i} \quad N_{terrain_i} \quad A_{moy_i} \quad P_{danger_i} \quad F_i \quad PW_i] \tag{25}$$

où l_i est la longueur du segment $\boldsymbol{P_i P_{i+1}}$; $\mathbb{P}_{XY}(l_i)$ est la longueur de la projection de ce segment dans le plan XY; N_{total_i} est le nombre de cases traversées par ce segment (tel que défini au chapitre 3); $N_{terrain_i}$ est le nombre de cases où le segment descend sous l'altitude minimum spécifiée (aussi défini au chapitre 3); A_{moy_i} est l'altitude moyenne de ce segment; P_{danger_i} est le pourcentage de ce segment qui traverse une ou plusieurs zones dangereuses; F_i est la quantité de carburant nécessaire pour que le drone vole ce segment; et PW_i est la puissance requise pour que le drone suive ce segment. Ces valeurs temporaires sont utilisées lors de la prochaine évaluation de fonction de coût afin d'accélérer le calcul. Évidemment, cette modification augmente significativement la complexité de notre programme, mais le gain de performance est très encourageant.

La deuxième optimisation consiste à diminuer le nombre de fois que le coût du lissage est évalué. D'après les données du tableau 3, le calcul du coût de lissage représente 70 % du temps d'exécution de la fonction de coût. Nous ajoutons donc deux paramètres de configuration (voir tableau 2) afin de calculer le coût lié au lissage de la trajectoire seulement aux dernières générations du processus évolutif et uniquement pour les meilleurs individus. De plus, ce coût est évalué uniquement lorsque tous les autres critères de faisabilité sont respectés. Nous avons déterminé par expérimentation que cette modification réduit grandement le temps d'exécution sans affecter la qualité de la solution finale. On retrouve au tableau 3, les temps d'exécution des principales fonctions notre AG sans et avec les modifications. Le gain de performance est de plus de 320 %.

Tableau 3. Accélération de l'AG suite aux optimisations apportées

Fonctions	Temps d'exécution sans optimisation (s)	Temps d'exécution avec optimisations (s)	Accélération apportée par les optimisations
Algorithme génétique	154.266	36.605	321 %
Calcul du coût	148.266	30.221	390 %
Coût longueur	1.867	1.544	20 %
Coût altitude	0.232	0.204	14 %
Coût zones dangereuses	16.222	7.017	131 %
Coût puissance	3.280	3.339	* 0 %
Coût collision avec le terrain	11.215	5.101	120 %
Coût carburant	8.831	5.975	47 %
Coût lissage	103.183	4.208	2 352 %
Coût long. min des segments	0.092	0.154	* 0 %
Sélection	0.031	0.016	* 0 %
Reproduction	0.939	1.097	* 0 %
Mutation	2.077	1.811	* 0 %
Remplacement	0.134	0.132	* 0 %
Lissage	0.015	0.031	* 0 %

* Les optimisations apportées à notre implémentation de l'AG ne modifient en aucun cas le temps de calcul de ces fonctions. La différence (positive ou négative) visible dans cette table est due à l'influence du chronométrage, à l'imprécision du chronomètre et au bruit du système d'exploitation.

Note : Temps d'exécution pour un seul essai de l'AG pour 128 chromosomes, 100 générations et des trajectoires de 8 points de passages.

4.5 Démonstration

Nous présentons au chapitre 8 une comparaison rigoureuse de l'AG et de l'OEP pour la planification de trajectoire pour 40 scénarios sur 8 terrains différents. Nous avons créé ces scénarios très complexes afin de bien démontrer la performance de nos modules de planification de trajectoires. Comme nous le verrons, les trajectoires générées sont toutes très bonnes, mais il peut être difficile d'évaluer leur qualité puisque la trajectoire optimale n'est pas évidente. Nous présentons donc dans cette section deux scénarios simples où la trajectoire optimale est évidente. Ceci nous permet de comparer la trajectoire générée par notre AG à celle qui est optimale afin de confirmer le bon fonctionnement de notre algorithme. Les paramètres de configurations sont listés au tableau 2 et les caractéristiques du drone, au tableau 15 (au chapitre 9).

4.5.1 Scénario 1 : ligne droite à altitude constante

Le premier scénario est affiché à la figure 20 et consiste à relier un point de départ (WP1) à un point d'arrivée (WP2) sans aucune zone dangereuse. Puisque les deux extrémités sont placées sur la rivière à l'altitude minimum de vol du drone (soit 30 m au-dessus de la surface) et qu'aucun obstacle (dénivellation ou zone dangereuse) ne sépare ces extrémités, la trajectoire optimale est évidemment la ligne droite. Suivant la définition de notre fonction de coût au chapitre 3, cette trajectoire optimale a un coût de 0. On constate à la figure 20 que la trajectoire générée (ligne jaune solide) par notre implémentation de l'AG maintient l'altitude minimum et est très proche de la ligne droite (ligne jaune pointillée). Son coût est plus petit que 0.000. La trajectoire générée est aussi affichée en 3D à la figure 21.

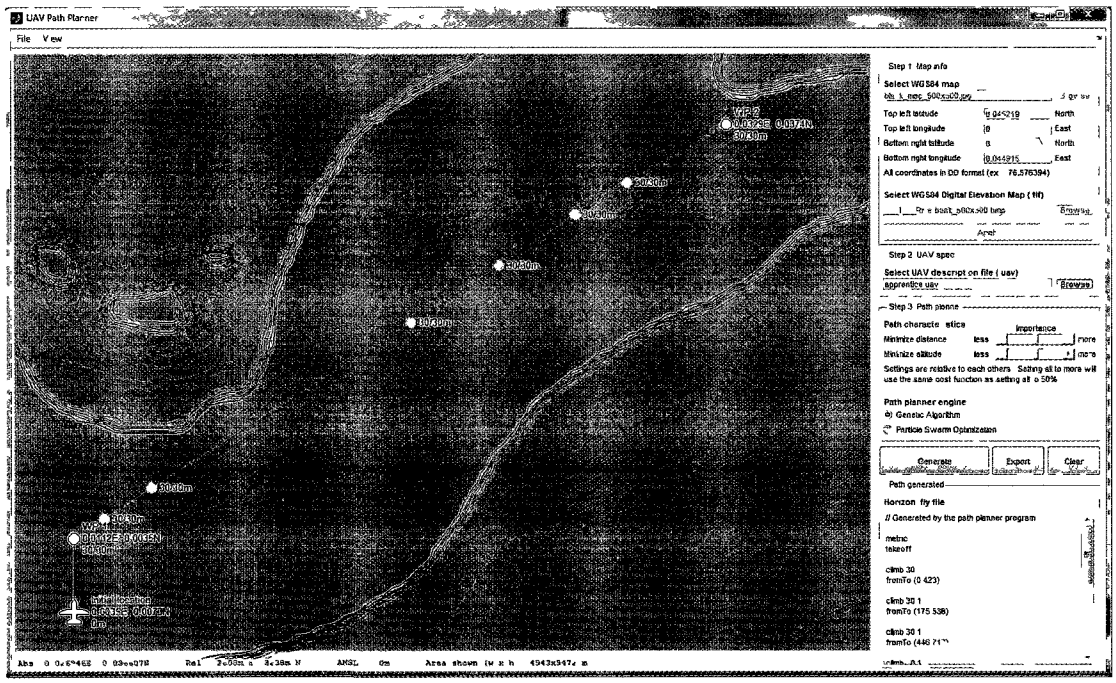


Figure 20. Trajectoire générée par l'AG pour le scénario simple n° 1

La figure 22 montre l'évolution du coût des chromosomes à chaque itération du processus évolutif de l'AG. Comme la population est initialisée aléatoirement, le coût moyen est d'abord très grand et diminue pour finalement atteindre une valeur très proche de 0. Puisque le terme de la fonction de coût associé au lissage de la trajectoire est évalué que pour les dernier 20 % du processus évolutif, le coût du meilleur chromosome reste plus grand que 11 jusqu'à la génération 160.

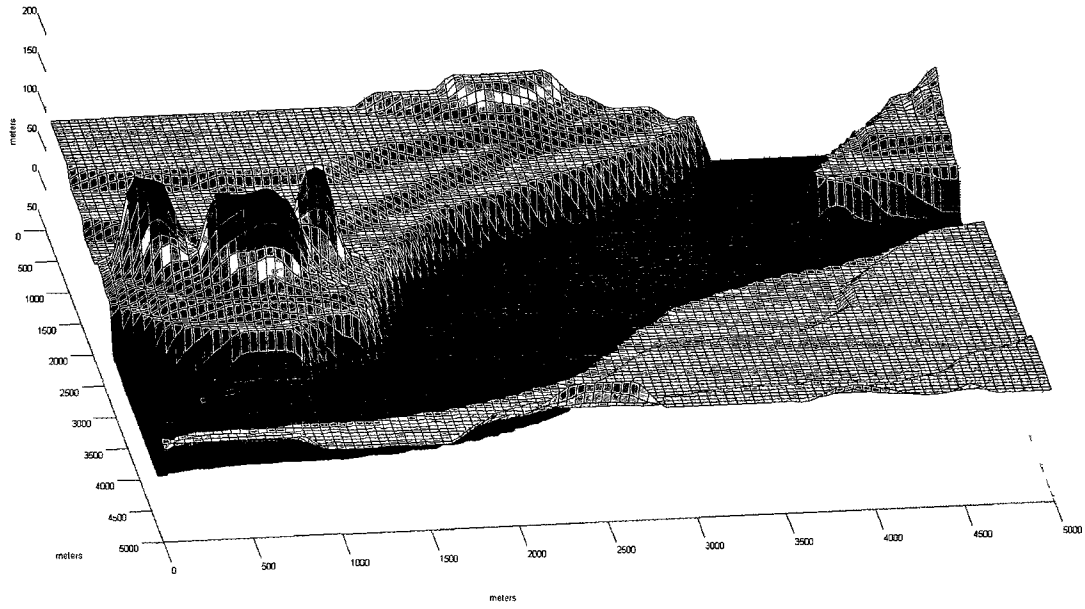


Figure 21. Visualisation 3D de la trajectoire générée par l'AG pour le scénario simple n° 1

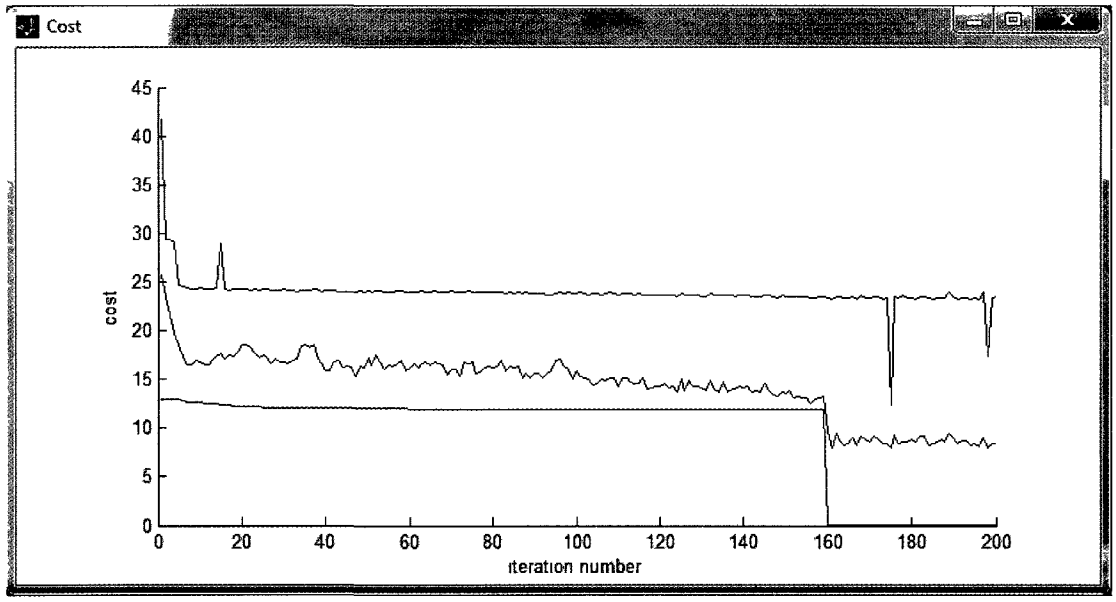


Figure 22. Coût de la meilleure solution (ligne bleue), coût moyen (ligne rouge) et coût de la pire solution (ligne noire) à chaque itération de l'AG

4.5.2 Scénario 2 : ligne courbe à altitude constante

Le deuxième scénario est illustré à la figure 23 et est identique à celui de l'exemple 1 à l'exception des deux zones dangereuses qui ont été ajoutées afin d'obstruer la ligne droite reliant le point WP1 au point WP2. La trajectoire optimale consiste à maintenir l'altitude minimum de vol (soit 30 m au-dessus de la surface de l'eau) et à contourner la première zone dangereuse par le bas et la seconde, par le haut. Afin de minimiser la distance à parcourir, la trajectoire optimale frôle chacune des zones dangereuses, mais sans jamais les pénetrer. On constate à la figure 23 que la trajectoire générée (ligne jaune solide) par notre implémentation de l'AG est très proche de la solution optimale que nous venons de décrire Son coût est de 0 036. La trajectoire générée est aussi dessinée en 3D à la figure 24 Tout comme pour l'exemple précédent, nous affichons le coût des chromosomes à chaque itération du processus évolutif de l'AG à la figure 25.

Figure 23 Trajectoire generée par l'AG (ligne jaune solide) pour le scenario simple n° 2

Nous avons présenté dans ce chapitre notre implémentation de l'AG pour la planification de trajectoires pour les drones. Nous utilisons l'AG original sans en modifier le comportement afin de garder l'efficacité prouvée de cette méthode. Notre AG utilise la representation et la fonction de coût défini au chapitre 3. Puisque nous sommes intéressés à utiliser notre algorithme dans une application en temps réel, nous avons apporté deux

optimisations à notre code et avons réduit de temps de calcul de plus de 320 %. Nous avons aussi démontré le bon fonctionnement de notre algorithme pour deux scénarios simples.

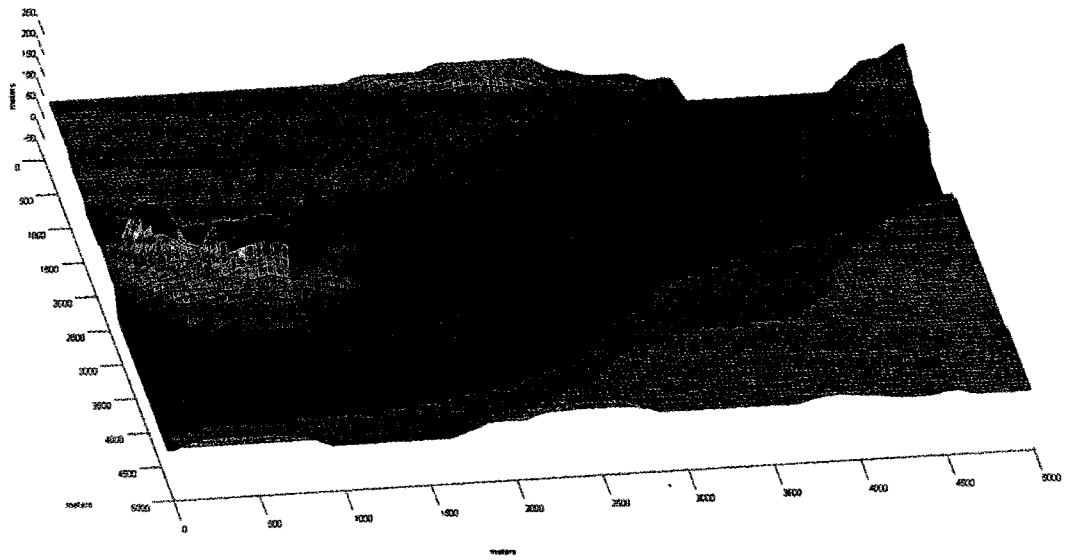


Figure 24. Visualisation 3D de la trajectoire générée par l'AG pour le scénario simple n° 2

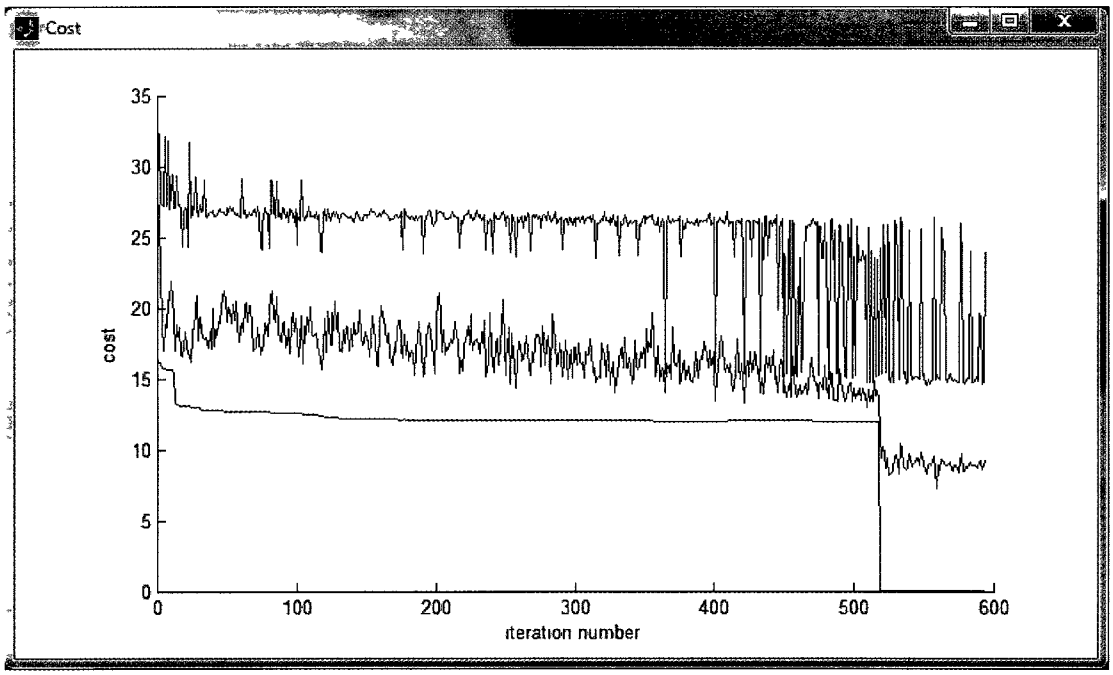


Figure 25. Coût de la meilleure solution (ligne bleue), coût moyen (ligne rouge) et coût de la pire solution (ligne noire) à chaque itération de l'AG

Chapitre 5

Algorithme d'optimisation par essaim de particules

Malgré que l'AG soit la méthode la plus utilisée pour la planification de trajectoires, l'optimisation par essaim de particules (OEP) est devenue très populaire au cours des dernières années [19]. De plus, certains auteurs trouvent que l'OEP est plus simple, nécessite un plus petit nombre d'évaluations avant de converger et produit des solutions supérieures à l'AG dans plusieurs problèmes d'optimisation [43]. L'OEP semble donc être une approche très compétitive à l'AG et nous la choisissons comme deuxième moteur d'optimisation pour notre module de planification de trajectoires. Dans ce chapitre, nous présentons en détail notre implémentation MATLAB de l'OEP ainsi que les optimisations apportées pour en minimiser le temps d'exécution.

5.1 Fonctionnement général

Publié pour la première fois en 1995 [25], l'OEP tire son origine de la simulation sur ordinateur du comportement social des oiseaux. Tel que mentionné dans [44], Kennedy et Eberhart reproduisent le mouvement complexe d'une volée d'oiseaux en contrôlant localement chaque animal suivant deux règles très simples : un oiseau tente de maintenir la même vitesse que ceux qui l'entourent et un oiseau accélère proportionnellement à la distance le séparant de l'objectif (source de nourriture). Suite à leur expérimentation, les deux auteurs réalisent le potentiel de leur modèle dans le domaine d'optimisation et développent alors l'OEP.

L'OEP simule le mouvement d'un essaim de particules dans un espace de recherche multidimensionnel. La position de chaque particule représente une solution possible (soit une trajectoire) qui se déplace vers une meilleure position (soit une meilleure solution). À chaque itération de la simulation, la vitesse et la position de chaque particule sont ajustées d'après la position précédente, la vitesse précédente, la meilleure position antérieurement occupée par la particule et la meilleure position antérieurement occupée par la meilleure particule de l'essaim. C'est en suivant ces règles simples que l'essaim de particules, initialement aléatoires, s'oriente et se dirige vers une solution optimale. Tel que défini dans [25], l'ajustement de la vitesse et de la position d'une particule i à un temps t se fait d'après les équations (26) et (27).

$$\boldsymbol{v_i}(t) = C_1\boldsymbol{v_i}(t-1) + C_2\boldsymbol{r_1}.*\left(\boldsymbol{p_i}(t-1) - \boldsymbol{x_i}(t-1)\right) + C_3\boldsymbol{r_2}.*\left(\mathbf{g}(t-1) - \boldsymbol{x_i}(t-1)\right) \quad (26)$$

$$\boldsymbol{x_i}(t) = \boldsymbol{x_i}(t-1) + \boldsymbol{v_i}(t) \quad (27)$$

Dans ces équations, les variables en gras représentent de vecteurs. $\boldsymbol{v_i}$ est la vitesse de la particule i, $\boldsymbol{x_i}$ est sa position, $\boldsymbol{p_i}$ est sa meilleure position déjà occupée, $\mathbf{g}$ est la meilleure position de la meilleure particule de l'essaim, $\boldsymbol{r_1}$ et $\boldsymbol{r_2}$ sont des vecteurs de valeurs aléatoires entre 0 et 1 et finalement, C_1, C_2 et C_3 sont des constantes.

Pour le développement de notre module de planification de trajectoires en MATLAB, nous utilisons l'OEP pour simuler le déplacement d'un essaim de trajectoires dans un espace multidimensionnel vers une position qui minimise la fonction de coût que nous avons définie. La représentation utilisée est la même qu'au chapitre 3 (chaque particule est représentée sous forme d'une suite de coordonnées (x, y, z) définissant les points de passage de la trajectoire). Le fonctionnement général de notre implémentation de l'OEP est illustré à la figure 26.

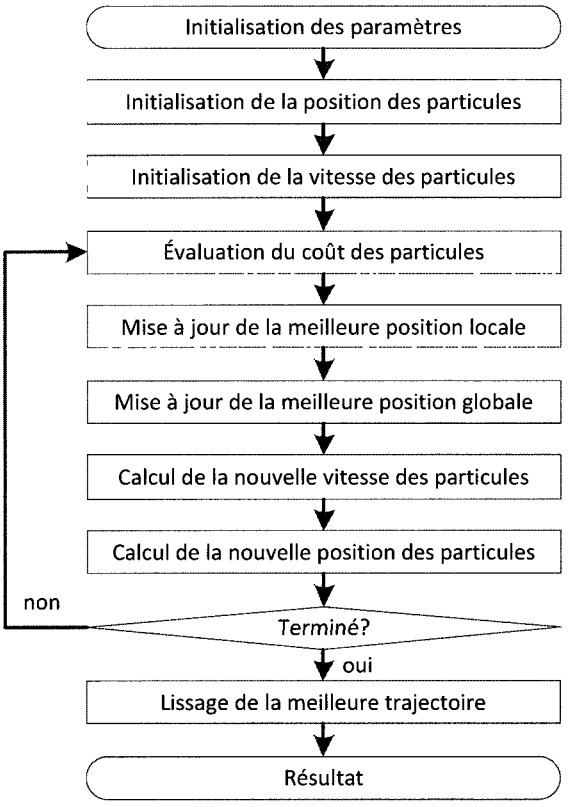


Figure 26. Diagramme de fonctionnement de notre algorithme d'OEP pour la planification de trajectoires

5.2 Détails d'implémentation

Nous discutons dans cette section des détails d'implémentation pour chacune des étapes illustrées à la figure 26.

1) **Initialisation des paramètres :** La première étape consiste à définir les paramètres d'entrée de notre algorithme d'OEP. À l'exception de quelques-uns identifiés par un astérisque au tableau 4, ceux-ci sont tous passés en paramètre à la fonction. Nous avons inclus dans ce tableau des valeurs suggérées (déterminées expérimentalement) pour les variables servant à la configuration de l'OEP.

Tableau 4. Liste des paramètres de configuration de notre algorithme d'OEP

Paramètres	Valeurs suggérées	Descriptions
Taille de l'essaim	128	Nombre de particules (ou solutions) utilisées.
Nombre d'itérations	200	Nombre d'itérations simulées. L'exécution de l'OEP se termine lorsque ce nombre est atteint.
Longueur des particules	8	Le nombre de points de passage d'une trajectoire. Contrairement à notre implémentation de l'AG, la longueur est ici fixe.
$*C_1$	0.9 à 0.2	Poids associé à la vitesse courante de la particule lors du calcul de la prochaine vitesse à l'équation (26). Pour réduire l'inertie de la particule tout au long du processus itératif, nous diminuons de façon linéaire la valeur de C_1.
$*C_2$	1	Poids associé à la meilleure position jamais occupée par la particule lors du calcul de la prochaine vitesse à l'équation (26).
$*C_3$	1	Poids associé à la meilleure position jamais occupée par l'essaim lors du calcul de la prochaine vitesse à l'équation (26).
Vérifier le coût du lissage à partir de X % du processus itératif	80 %	Paramètre servant à réduire le nombre d'itérations auxquelles le coût lié au lissage de la trajectoire est évalué afin d'accélérer l'exécution de l'algorithme.
Vérifier le coût du lissage pour les X % meilleures particules	25 %	Paramètre servant à réduire le nombre de solutions dont le coût lié au lissage de la trajectoire est évalué afin d'accélérer l'exécution de l'algorithme.
Carte d'élévation digitale	s.o.	Une matrice 2D où chaque case contient l'élévation du terrain.
Échelle horizontale	s.o.	Le nombre de mètres que représente une case de la matrice 2D. Les échelles pour l'axe des X et l'axe des Y doivent être identiques puisqu'un seul paramètre est utilisé. L'échelle pour l'axe des Z est toujours de 1 m par unité.
Point de départ	s.o.	Les coordonnées (x, y, z) du point de départ de la trajectoire d'après de système de coordonnées défini à la section 3.1.
Point d'arrivée	s.o.	Les coordonnées (x, y, z) du point d'arrivée de la trajectoire d'après de système de coordonnées défini à la section 3.1.
Zones dangereuses	s.o.	Une série de coordonnées (x_i, y_i, z_i) définissant les zones dangereuses comme à la section 3.1.

Caractéristique du drone	s.o.	Une structure de donnée contenant toutes les caractéristiques du drone nécessaire pour que la fonction de coût puisse évaluer la puissance requise, la puissance disponible, la consommation de carburant, le rayon de courbure minimale, etc. (voir tableau 15 au chapitre 9)
* Contrairement aux autres variables qui sont passées en paramètre à la fonction OEP, ces variables sont définies dans la fonction elle-même et ne peuvent être modifiées à l'exécution.		

2) **Initialisation de la position des particules :** Contrairement à l'AG, l'OEP ne permet pas de faire varier la longueur des solutions pendant le processus itératif. L'encodage de la position des particules ne nécessite donc pas de cellules de matrices et peut simplement être encodée dans une matrice 3D. La première dimension représente l'index des particules, la deuxième, l'index des points de passage et la troisième, les coordonnées (x, y, z) des points. Les positions des particules sont initialisées aléatoirement suivant une distribution uniforme dans l'espace de recherche. Ces points ne sont pas ordonnés et les trajectoires initiales contiennent fort probablement plusieurs boucles. Les meilleures positions jamais occupées par chaque particule (aussi appelé meilleures positions antérieures) sont aussi gardées en mémoire dans une matrice 3D de la même dimension et initiée avec les mêmes valeurs que celle des positions courantes.
3) **Initialisation de la vitesse des particules :** La vitesse de chacune des particules est initialement nulle et encodée de manière identique à la position, soit une matrice 3D.
4) **Évaluation du coût :** Le processus itératif débute avec l'évaluation du coût pour chacune des particules d'après la fonction définie au chapitre 3.
5) **Mise à jour de la meilleure position locale :** Pour chacune des particules, le coût de sa position courante est comparé à celui de sa meilleure position antérieure. Si la position courante est de meilleure qualité, la meilleure position antérieure est mise à jour.
6) **Mise à jour de la meilleure position globale :** La position courante de chaque particule est comparée à la meilleure position antérieure de l'essaim. Cette dernière est mise à jour si une particule de meilleure qualité est trouvée. Puisqu'il existe qu'une seule meilleure particule à un instant donné, cette implémentation s'appelle l'OEP globale. Une seconde approche, l'OEP locale, consiste à limiter l'influence sociale d'une particule à son voisinage. Dans ce cas, l'équation (26) n'utilise pas la meilleure particule de l'essaim, mais la meilleure dans le voisinage de la particule en question. Ce voisinage est souvent défini par l'index de la particule, mais peut aussi être relatif aux positons. Tel

que mentionné dans [44], l'OEP local ralentit la convergence, mais permet de maintenir une meilleure diversité de l'essaim. Afin de minimiser la complexité et le temps d'exécution de notre algorithme d'OEP séquentielle, nous implémentons la version globale. La version locale est utilisée pour notre implémentation parallèle au chapitre 7.

7) **Calcul de la vitesse des particules :** à toutes les itérations, la vitesse de chaque particule est ajustée d'après sa vitesse courante, sa position courante, sa meilleure position antérieure et la meilleure position antérieure de l'essaim suivant l'équation (26). Tel que suggéré plus tard par un des auteurs mêmes de l'OEP [26], il est possible de réduire la valeur de C_1 dans l'équation (26) de manière à diminuer l'inertie des particules tout au long du processus itératif. Une grande inertie en début d'optimisation favorise l'exploration de l'espace de recherche et minimise les chances de convergence prématurée vers un minimum local. D'un autre côté, une petite inertie en fin d'exécution réduit l'oscillation des particules et permet à l'essaim d'atteindre la meilleure position. Dans notre implémentation, nous réduisons C_1 de façon linéaire.
8) **Calcul de la position des particules :** à toutes les itérations, la position de chaque particule est mise à jour d'après sa vitesse suivant l'équation (26). Si la nouvelle position d'une particule se trouve à l'extérieure de l'espace de recherche, la particule est ramenée sur la frontière.
9) **Critère de terminaison :** Tout comme pour l'AG, l'exécution de l'OEP peut s'arrêter lorsque la meilleure particule atteint un certain coût, lorsque le coût de la meilleure particule ne diminue plus ou après un nombre fixe d'itérations. Afin d'assurer un temps fixe d'exécution, nous implémentons cette dernière méthode.
10) **Lissage :** La trajectoire finale obtenue par l'OEP est définie comme une série de segments de droite. Ce trajet contient des sommets que nous lissons par des arcs de cercle et des cercles sur plateaux horizontaux. Cette méthode provient de [37] et est décrite en détail au chapitre 6.

5.3 Structure de notre implémentation de l'OEP

Afin de gérer la complexité de l'implémentation de l'algorithme d'OEP en MATLAB, nous avons utilisé le paradigme de programmation structurée et avons divisé notre programme en plusieurs fonctions tel que montré à la figure 27. Sur l'illustration, chaque boîte représente

une fonction. Celles qui se terminent « .m » sont définies dans leur propre fichier, tandis que les autres se trouvent dans le fichier de la fonction supérieure.

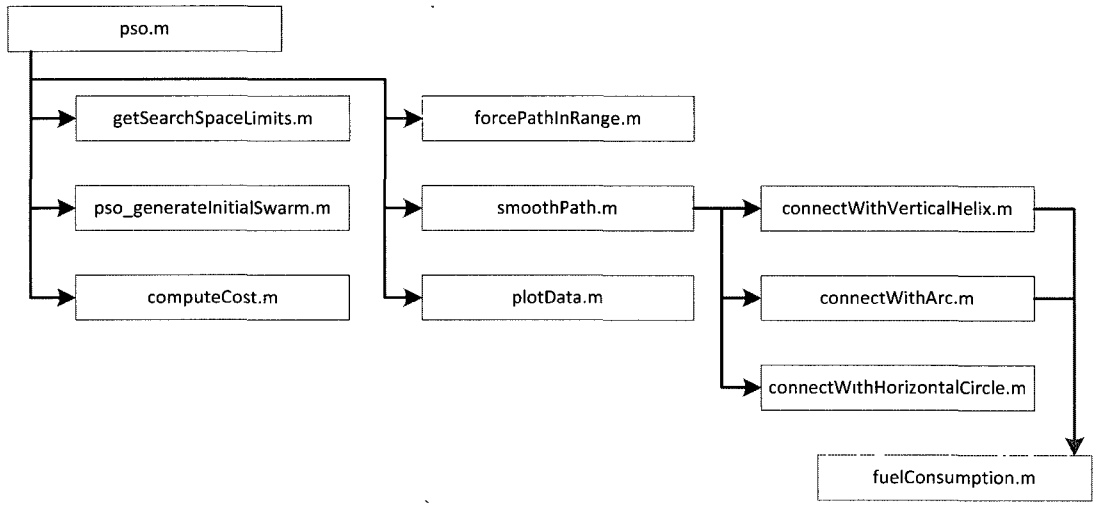


Figure 27. Diagramme de la structure de notre algorithme d'OEP

5.4 Optimisations apportées

Tout comme pour l'AG, nous sommes intéressés à minimiser le temps de calcul de notre OEP afin d'évaluer la possibilité d'une planification de trajectoires en temps réel. Cependant, contrairement à l'AG, la position des particules (les solutions possibles) est entièrement modifiée à chaque itération. On ne peut donc pas minimiser le temps de calcul de la fonction de coût par l'utilisation de valeurs temporaire. La seule optimisation possible est de réduire le nombre de fois que l'on calcule le terme de la fonction de coût associé au lissage possible. Cette optimisation ne modifie pas la qualité de la trajectoire faisable et permet de réduire le temps de calcul de 183 % tel que montré au tableau 5. Pour 128 particules et 100 itérations, l'OEP s'avère toutefois 45 % plus lent que l'AG.

Tableau 5. Accélération de l'OEP suite aux optimisations apportées

Fonctions	Temps d'exécution sans optimisation (s)	Temps d'exécution avec optimisations (s)	Accélération apportée par les optimisations
OEP	150.619	53.080	183 %
Calcul du coût	144 994	48.736	198 %
Coût longueur	2 290	1 973	* 0 %
Coût altitude	0.186	0.283	* 0 %
Coût zones dangereuses	16.464	16.385	* 0 %
Coût puissance	3.492	3.478	* 0 %
Coût collision avec le terrain	11.135	11.071	* 0 %
Coût carburant	9.430	9.211	* 0 %
Coût lissage	97.848	3.479	2 712 %
Coût long. min des segments	0.139	0.142	* 0 %
Lissage	0.031	0.016	* 0 %

* Les optimisations apportées à notre implémentation de l'OEP ne modifient en aucun cas le temps de calcul de ces fonctions La différence (positive ou négative) visible dans cette table est due à l'influence du chronométrage, à l'imprécision du chronomètre et au bruit du système d'exploitation

Note : Temps d'exécution pour un seul essai de l'OEP pour 128 particules, 100 itérations et des trajectoires de 8 points de passages.

5.5 Démonstrations

Comme nous l'avons expliqué précédemment, nous présentons au chapitre 8 les résultats de l'AG et de l'OEP pour 40 scénarios très complexes. Nous verrons que les trajectoires générées sont toutes très bonnes, mais il peut être difficile d'évaluer leur qualité puisque la trajectoire optimale n'est pas évidente. Nous présentons donc dans cette section deux scénarios simples où la trajectoire optimale est évidente. Ceci nous permet de comparer la trajectoire générée par notre OEP à celle qui est optimale afin de confirmer le bon fonctionnement de notre algorithme. Les paramètres de configurations sont listés au tableau 2 et les caractéristiques du drone, au tableau 15 (au chapitre 9).

5.5.1 Scénario 1 : ligne droite à altitude constante

Le premier scénario est affiché à la figure 28 et consiste à relier un point de départ (WP1) à un point d'arrivée (WP2) sans aucune zone dangereuse. Puisque les deux extrémités sont placées sur la rivière à l'altitude minimum de vol du drone (soit 30 m au-dessus de la surface) et qu'aucun obstacle (dénivellation ou zone dangereuse) ne sépare ces extrémités, la

trajectoire optimale est évidemment la ligne droite. Suivant la définition de notre fonction de coût au chapitre 3, cette trajectoire optimale a un coût de 0. On constate à la figure 28 que la trajectoire générée (ligne jaune solide) par notre implémentation de l'OEP maintient l'altitude minimum et est parfaitement superposée à la ligne droite (ligne jaune pointillée). Son coût est plus petit que 0.000. La trajectoire générée est aussi dessinée en 3D à la figure 29. Le coût des particules à chaque itération du processus itératif de l'OEP est affiché à la figure 30. Tout comme pour l'AG, on remarque que le coût de la meilleure solution reste supérieur à 11 jusqu'à l'itération 160 où le terme associé au lissage de la courbe est finalement évalué.

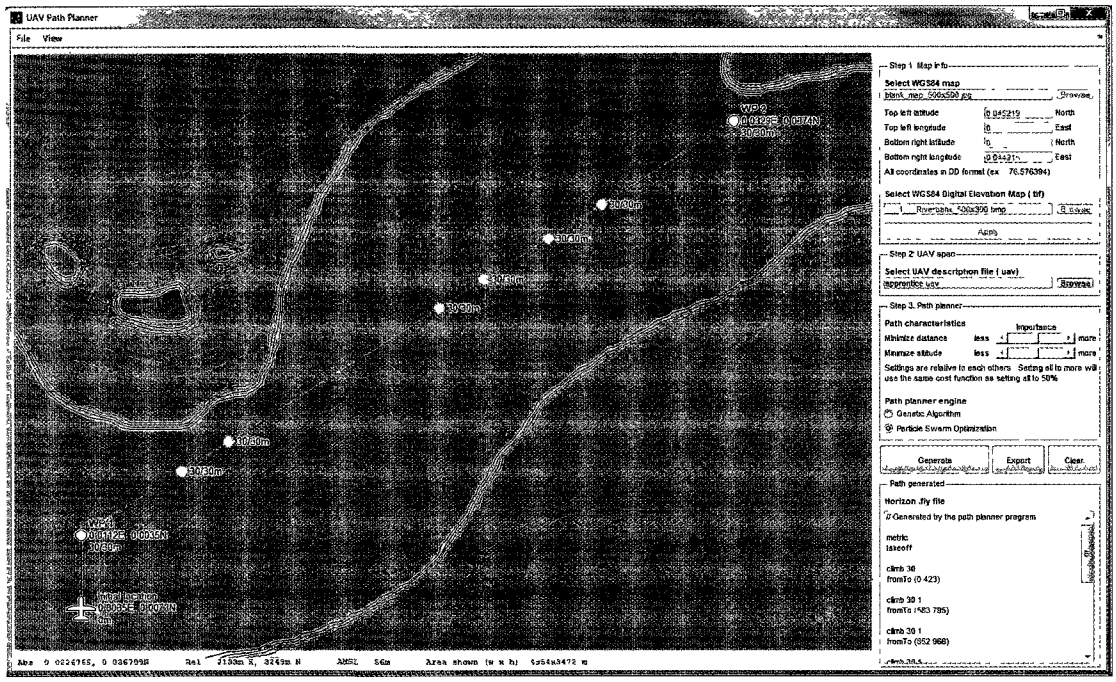


Figure 28. Trajectoire générée par l'OEP (ligne jaune solide) pour le scénario simple n° 1

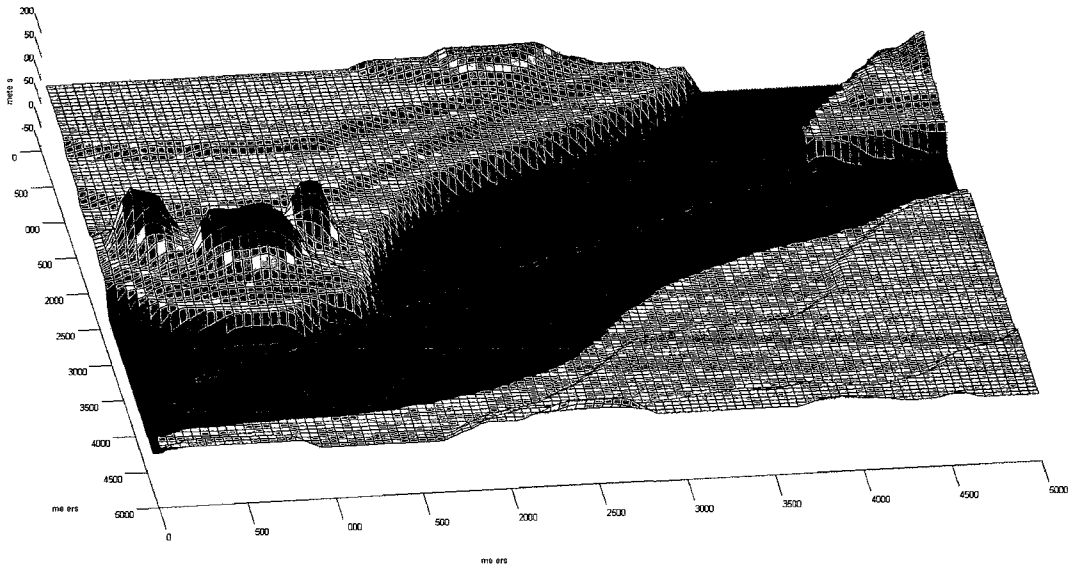


Figure 29 Visualisation 3D de la trajectoire generee par l'OEP pour le scenario simple n 1

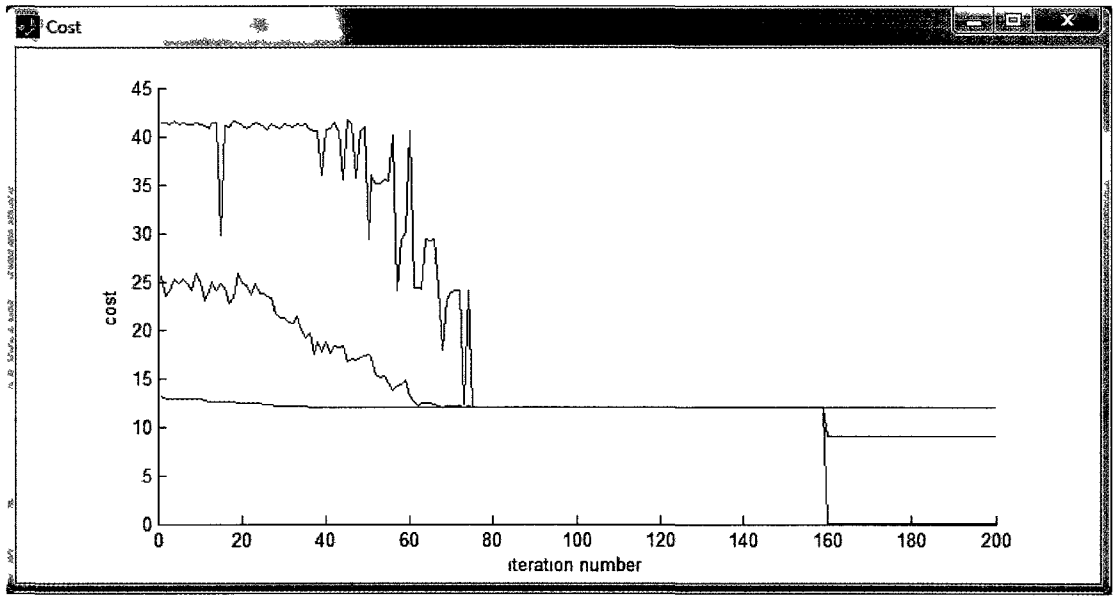


Figure 30 Coût de la meilleure solution (ligne bleue), coût moyen (ligne rouge) et coût de la pire solution (ligne noire) a chaque iteration de l'OEP

5.5.2 Scénario 2 : ligne courbe à altitude constante

Le deuxième scénario est illustré à la figure 31 et est identique à celui de l'exemple 1 à l'exception des deux zones dangereuses qui ont été ajoutées afin d'obstruer la ligne droite reliant le point WP1 au point WP2. La trajectoire optimale consiste à maintenir l'altitude minimum de vol (soit 30 m au-dessus de la surface de l'eau) et à contourner la première zone dangereuse par le bas et la seconde, par le haut. Afin de minimiser la distance à parcourir, la trajectoire optimale frôle chacune des zones dangereuses, mais sans jamais les pénétrer. On constate à la figure 31 que la trajectoire générée (ligne jaune solide) par notre implémentation de l'OEP est très proche de la solution optimale que nous venons de décrire. Son coût est de 0.031. La trajectoire générée est aussi dessinée en 3D à la figure 32. Tout comme pour l'exemple précédent, nous affichons le coût des particules à chaque itération de l'OEP à la figure 33.

Nous avons présenté dans ce chapitre notre implémentation de l'OEP pour la planification de trajectoires pour les drones. Tout comme pour notre AG, notre OEP utilise la représentation et la fonction de coût défini au chapitre 3. Puisque nous souhaitons utiliser notre algorithme dans une application en temps réel, nous avons optimisé notre code et avons réduit de temps de calcul de 183 %. Nous avons aussi démontré le bon fonctionnement de notre algorithme pour deux scénarios simples.

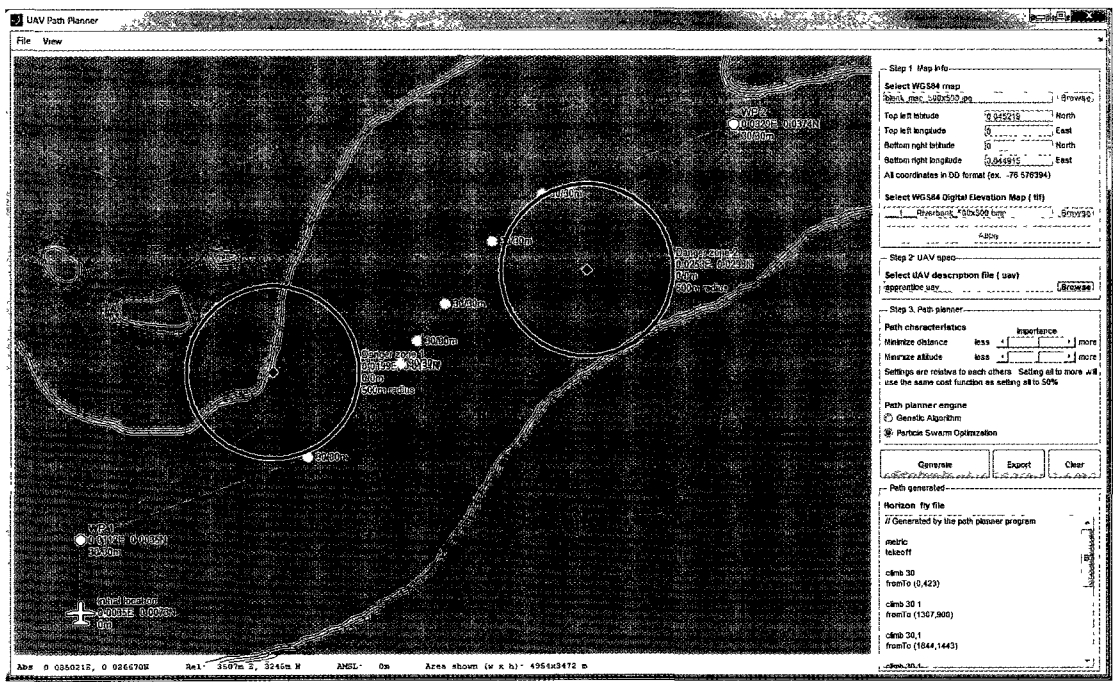


Figure 31. Trajectoire générée par l'OEP (ligne jaune solide) pour le scénario simple nº 2

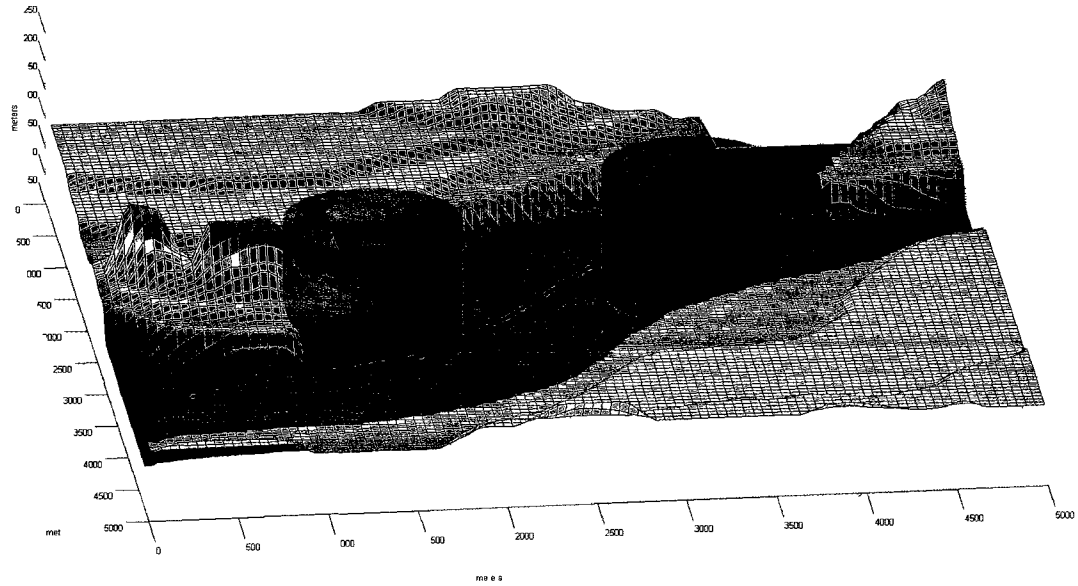


Figure 32 Visualisation 3D de la trajectoire generee par l'OEP pour le scenario simple n° 2

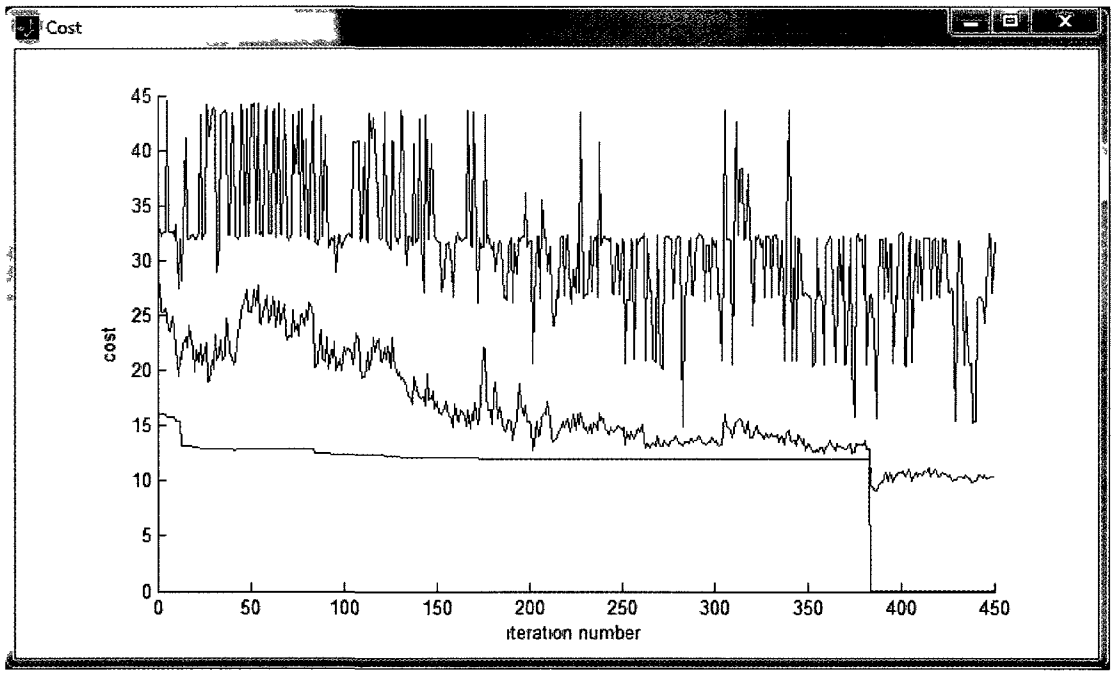


Figure 33 Coût de la meilleure solution (ligne bleue), coût moyen (ligne rouge) et coût de la pire solution (ligne noire) a chaque iteration de l'OEP

Chapitre 6

Lissage de la trajectoire

Les routes générées par les algorithmes d'optimisation sont formées de segments de droite et contiennent des discontinuités dans leur tangente, ou vitesse, aux points de connexion des segments. Contrairement à un véhicule terrestre qui peut s'arrêter pour se réorienter aux intersections, le drone à voilure fixe ne peut suivre un trajet segmenté. Comme nous l'avons présenté au chapitre 2, deux méthodes sont habituellement utilisées pour lisser la trajectoire : les splines et les connexions par arcs de cercle. L'avantage du lissage par arcs de cercle est que la trajectoire finale reste simple et peut facilement être analysée pour s'assurer qu'elle respecte les contraintes dynamiques du drone en tout point. Lors du développement de nos algorithmes de planification de trajectoires, nous implémentons cette méthode exactement comme elle est présentée par Labonté dans [37]. Notre module de lissage est utilisé pour générer la trajectoire finale, mais aussi par la fonction de coût durant le processus d'optimisation afin de s'assurer que la trajectoire calculée par l'AG ou l'OEP peut effectivement être lissée.

Dans son ouvrage, Labonté discute des connexions par arcs de cercle, des connexions par cercles sur plateau horizontal et des hélices. Il dérive les formules mathématiques nécessaires à la construction des courbes ainsi que les conditions de faisabilité à vérifier. Nous expliquons dans ce chapitre les principales étapes que nous avons suivies pour l'implémentation de ces constructions sans s'attarder aux détails des calculs. Puisque ce chapitre est entièrement basé sur [37], nous omettons de toujours y faire référence.

6.1 Connexion par un arc circulaire

Lors du lissage de la trajectoire, nous connectons chacun des segments de droite par des sections circulaires afin d'éliminer les sommets et d'assurer la continuité de la vitesse. Lorsque possible, nous utilisons des connexions par un seul arc de cercle. Comme on le voit à la figure 34, ces constructions sont simples et minimisent la distance à parcourir.

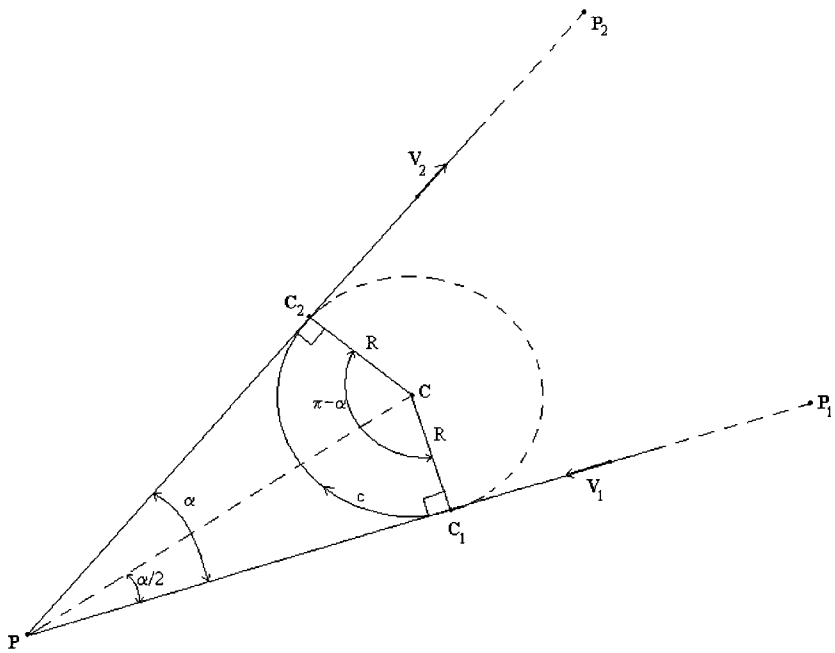


Figure 34. Connexion des segments P_1P et PP_2 par arc de cercle (reproduit de [37])

6.1.1 Construction de la connexion par un arc de cercle

Les étapes à suivre pour relier deux segments (prenons $\boldsymbol{P_1P}$ et $\boldsymbol{PP_2}$ à la figure 34) par arc de cercle sont les suivantes (notez que les vecteurs sont imprimés en gras) :

1) Calculer $\boldsymbol{\tau_1}$ et $\boldsymbol{\tau_2}$, les deux vecteurs unitaires orientés selon $\boldsymbol{V_1}$ et $\boldsymbol{V_2}$.
2) Calculer l'angle α.
3) Calculer $\boldsymbol{n}$, le vecteur unitaire normal au plan contenant les points $\boldsymbol{P_1}$, $\boldsymbol{P}$ et $\boldsymbol{P_2}$.
4) Calculer θ, l'angle entre l'axe des Z et le plan contenant les points $\boldsymbol{P_1}$, $\boldsymbol{P}$ et $\boldsymbol{P_2}$.
5) Calculer R, le rayon de l'arc de cercle, de façon à respecter le facteur de charge maximum du drone, l'angle de roulis maximum permise et le rayon de courbure minimum. Ces trois valeurs sont spécifiées dans le fichier de description du drone (voir tableau 15 au chapitre 9).
6) Calculer $\boldsymbol{C}$, le centre de l'arc de cercle d'après R, $\boldsymbol{\tau_1}$ et $\boldsymbol{\tau_2}$.
7) Finalement, calculer les points formant l'arc de cercle :

$$\boldsymbol{x}(t) = \boldsymbol{C} - R\ cosec(\alpha)(\cos(\alpha + \omega t)\,\boldsymbol{\tau_1} + \cos(\omega t)\,\boldsymbol{\tau_2}) \tag{28}$$

où ω est la vitesse angulaire, t est le temps et ωt varie de 0 à $(\pi - \alpha)$.

Dans notre implémentation, nous générons l'arc de cercle avec des valeurs de ωt de façon à ce que les points soient distancés d'après l'intervalle spécifié dans le fichier de description du drone (voir tableau 15 au chapitre 9).

6.1.2 Critères de faisabilité

La connexion de deux segments de droite par arc de cercle est préférable, mais pas toujours possible. Nous discutons dans cette section de trois conditions qui doivent être respectées pour que la trajectoire finale soit faisable.

Condition 1 : les segments $\boldsymbol{P_1P}$ et $\boldsymbol{PP_2}$ (voir figure 34) doivent être suffisamment long pour que les points de jonction entre l'arc et les deux droites se retrouvent sur les segments.

Condition 2 : l'arc de cercle ne s'approche pas plus près du sol que l'altitude de vol minimum spécifiée dans le fichier de description du drone (voir tableau 15 au chapitre 9). Cette vérification se fait similairement à celle faite pour les segments de droite à la section 3.3.5.

Condition 3 : la puissance requise pour voler la trajectoire ne doit pas être plus grande que celle disponible du drone. Étonnamment, il se peut que la puissance du drone soit suffisante pour voler les deux segments, mais pas assez grande pour faire la connexion par arc de cercle. Cette situation est évidente à la figure 35 où le drone doit voler verticalement pour passer du segment du bas à celui du haut. Pour vérifier la condition 3, nous calculons le taux d'ascension maximum requis sur l'arc de cercle et comparons cette valeur au taux d'ascension maximum possible du drone d'après son altitude, son poids actuel et sa puissance.

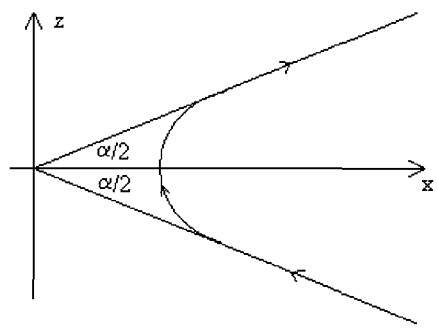


Figure 35. Connexion par arc circulaire d'un segment ascendant à un autre segment ascendant (reproduit de [37])

Puisque l'arc de cercle est toujours plus court que le suivi des segments $\boldsymbol{C_1P}$ *et* $\boldsymbol{PC_2}$, la quantité de carburant requise pour que le drone vole la connexion est ignorée. Dans le cas où l'une des trois conditions n'est pas remplie, la connexion par arc de cercle simple n'est pas possible et notre module de lissage tente alors de relier les deux segments par un cercle sur plateau horizontal.

6.2 Connexion par un cercle sur plateau horizontal

La connexion par un cercle sur plateau horizontal a l'avantage de nécessiter une puissance égale ou plus petite que celle requise par les deux segments de droite. La condition 3 de la section précédente est donc toujours valide lorsque les deux segments de droite sont faisables. Tel que visible à la figure 36, cette construction inclus un premier arc circulaire C_1 pour rejoindre le cercle horizontal C suivi d'un deuxième arc circulaire C_2 pour quitter le cercle et atteindre le deuxième segment de droite.

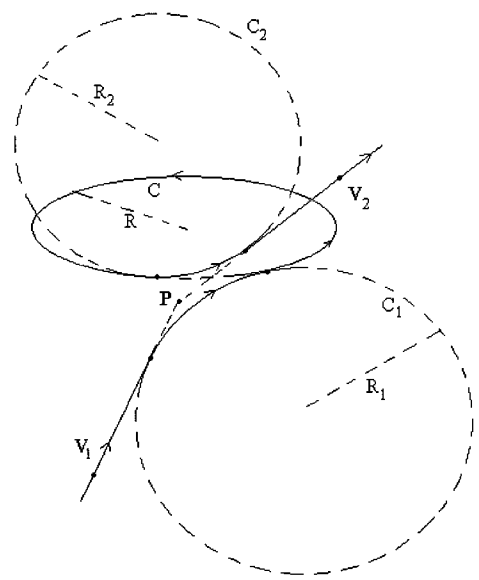


Figure 36. Connexion de deux segments de droite par cercle sur plateau horizontal (reproduit de [37])

6.2.1 Construction de la connexion par un cercle sur plateau horizontal

Tout comme à la section précédente, nous présentons ici les lignes directrices à suivre pour connecter deux segments de droite (prenons $\boldsymbol{P_1P}$ et $\boldsymbol{PP_2}$ à la figure 38) à l'aide l'un cercle sur plateau horizontal (notez que les vecteurs sont imprimés en gras).

1) La première étape consiste à transférer les points $\boldsymbol{P_1}$, $\boldsymbol{P}$ et $\boldsymbol{P_2}$ dans un système de coordonnées spécial centré sur le point $\boldsymbol{P}$ et orienté comme à la figure 37. Ce transfert se fait à l'aide d'une matrice de rotation et permet de simplifier les équations utilisées.

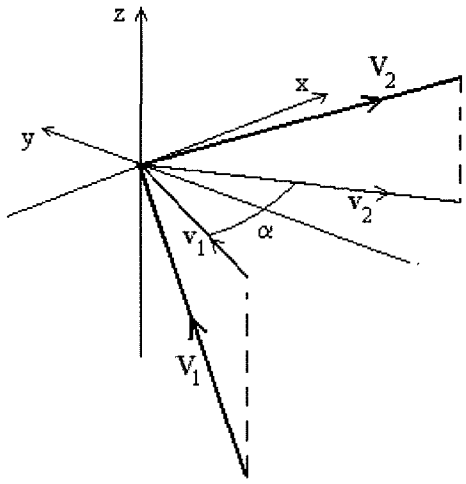


Figure 37. Projection des vecteurs V_1 et V_2 dans le plan XY du système de coordonnées spécial (reproduit de [37])

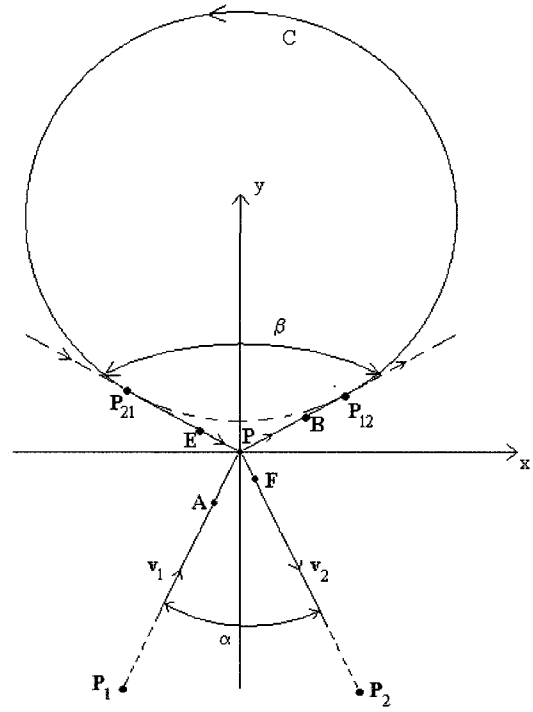


Figure 38. Vue de haut du plan XY montrant la connexion des segments P_1P et PP_2 par cercle sur plateau horizontal dans le système de coordonnées spécial (reproduit de [37])

2) D'après les points $\boldsymbol{P_1}$, $\boldsymbol{P}$ et $\boldsymbol{P_2}$ dans le système de coordonnées spécial, calculer les vecteurs $\boldsymbol{v_1}$ et $\boldsymbol{v_2}$, soient les projections de $\boldsymbol{V_1}$ et $\boldsymbol{V_2}$ dans le plan XY.
3) Calculer α, l'angle entre $-\boldsymbol{v_1}$ et $\boldsymbol{v_2}$.
4) Calculer θ_1 et θ_2, les angles entre l'axe des Z et les vecteurs $\boldsymbol{V_1}$ et $\boldsymbol{V_2}$.
5) Calculer d, soit la longueur des segments $\boldsymbol{PP_{12}}$ et $\boldsymbol{P_{21}P}$ de façon à laisser suffisamment d'espace pour relier les segments $\boldsymbol{P_1P}$ et $\boldsymbol{PP_2}$ au cercle horizontal par des arcs de cercle simples.
6) Calculer R_c, le rayon minimum du cercle horizontal C de façon similaire à la section précédente et son centre $\boldsymbol{C_C}$, d'après les valeurs de d et R_c.
7) Calculer β, l'angle entre les segments $\boldsymbol{PP_{12}}$ et $\boldsymbol{P_{21}P}$.

8) Tel qu'illustré aux figure 39 et figure 40, l'arrivée vers le cercle horizontal C se fait par le segment de droite entrant $\boldsymbol{P_1P}$, l'arc circulaire C_1 et un segment transitoire et tangent au cercle horizontal. Le départ se fait de façon semblable et inclut aussi un segment transitoire et tangent au cercle horizontal, l'arc circulaire C_2 et finalement le segment sortant $\boldsymbol{PP_2}$. Le calcul des arcs circulaires se fait comme à la section précédente d'après les vecteurs unitaires orientés selon $\boldsymbol{V_1}$ et $\boldsymbol{V_{12}}$ pour le premier arc et $\boldsymbol{V_{12}}$ et $\boldsymbol{V_2}$ pour le second.

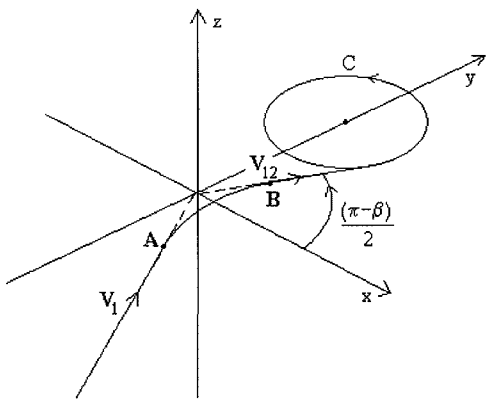


Figure 39. Segment de droite ascendant P_1P suivi de l'arc circulaire C_1 et d'un segment de droite transitoire dans le plan XY et tangent au cercle C
(reproduit de [37])

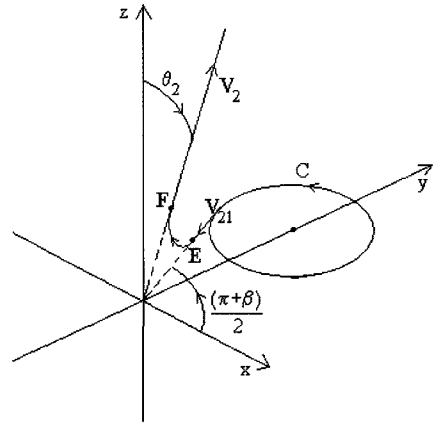


Figure 40. Segment de droite transitoire dans le plan XY et tangente au cercle C suivi de l'arc circulaire C_2 et du segment de droite ascendant PP_2
(reproduit de [37])

9) Finalement, les points formant le cercle horizontal peuvent être calculés comme suit :

$$\boldsymbol{x_C}(t) = \boldsymbol{C_C} + R_c[cos(\omega t), \quad sin(\omega t), \quad 0] \tag{29}$$

où ω est la vitesse angulaire, t est le temps et ωt varie de $-\beta/2$ à $\pi + \beta/2$. La série de points dessinant les deux arcs de cercle et le cercle est générée de la même façon qu'à la section précédente. La connexion finale reliant les segments $\boldsymbol{P_1P}$ et $\boldsymbol{PP_2}$ par cercle sur plateau horizontal est la concaténation de points de passages formant C_1, C et C_2 transférés dans le système de coordonnées original.

6.2.2 Critères de faisabilité

Trois conditions doivent être vérifiées pour déterminer la faisabilité de la connexion par cercle sur plateau horizontal. Contrairement à celle par arc de cercle, il n'est pas nécessaire de

vérifier la condition de puissance requise et disponible puisqu'elle est toujours valide lorsque les deux segments sont faisables.

Condition 1 : le segment $\boldsymbol{P_1P}$ (voir figure 38) doit être suffisamment long pour que le point de jonction entre l'arc C_1 et la droite contenant $\boldsymbol{P_1P}$ se trouve sur le segment. Similairement, la longueur du segment $\boldsymbol{PP_2}$ doit aussi être vérifiée.

Condition 2 : ni l'arc C_1, le cercle C ou l'arc C_2 ne s'approchent plus près du sol que l'altitude de vol minimum spécifiée dans le fichier de description du drone (voir tableau 15 au chapitre 9). Cette vérification se fait similairement à celle faite pour les segments de droite à la section 3.3.5.

Condition 3 : le drone transporte suffisamment de carburant pour voler la connexion. Contrairement à celle par arc de cercle, la connexion par cercle sur plateau horizontal allonge la trajectoire. Nous calculons donc la distance additionnelle et la quantité de carburant supplémentaire et ajustons les termes $C_{longueur}$ et $C_{carburant}$ de la fonction de coût.

Si l'une des trois conditions n'est pas respectée, la connexion est infaisable et nous ajustons le terme $C_{lissage}$ de la fonction de coût de manière à identifier que la trajectoire finale inclus une discontinuité dans sa vitesse et ne peut être volée par le drone.

6.3 Trajectoire hélicoïdale

Lors du lissage de la trajectoire, nous utilisons des sections hélicoïdales pour remplacer tout segment de droite demandant une puissance trop grande. Prenons deux segments qui se suivent $\boldsymbol{P_1P}$ et $\boldsymbol{PP_2}$. Dans le cas où $\boldsymbol{P_1P}$ est faisable et $\boldsymbol{PP_2}$ infaisable dû à la puissance requise trop élevée, il est possible de remplacer $\boldsymbol{PP_2}$ par une hélice verticale qui monte jusqu'à l'altitude de $\boldsymbol{P_2}$ suivie d'un segment de droite qui connecte l'hélice au point $\boldsymbol{P_2}$. Évidemment, ce scénario est possible uniquement lorsque $\boldsymbol{P_2}$ se trouve à une altitude en dessous du plafond de service du drone. Ce scénario est illustré à la figure 41. Comme il est visible à la figure 42, cette construction est semblable à celle de la connexion par cercle sur plateau horizontal et inclus :

1) un premier segment de droite $\boldsymbol{P_1A}$;
2) un arc de cercle C_1 qui relie $\boldsymbol{A}$ à $\boldsymbol{B}$;
3) un segment de droite transitoire qui se connecte à l'hélice reliant $\boldsymbol{B}$ à $\boldsymbol{P_{12}}$;
4) une hélice verticale qui relie $\boldsymbol{P_{12}}$ à $\boldsymbol{P_{21}}$;

5) un deuxième segment de droite transitoire qui quitte l'hélice et relie $\boldsymbol{P_{21}}$ à $\boldsymbol{E}$;
6) un deuxième arc de cercle C_2 qui relie $\boldsymbol{E}$ à $\boldsymbol{F}$; et
7) un deuxième segment de droite horizontal qui relie $\boldsymbol{F}$ à $\boldsymbol{P_2}$.

Il est important de noter que les deux segments transitoires sont inclinés par rapport à l'horizontale afin de se connecter à l'hélice sans aucune discontinuité dans la vitesse.

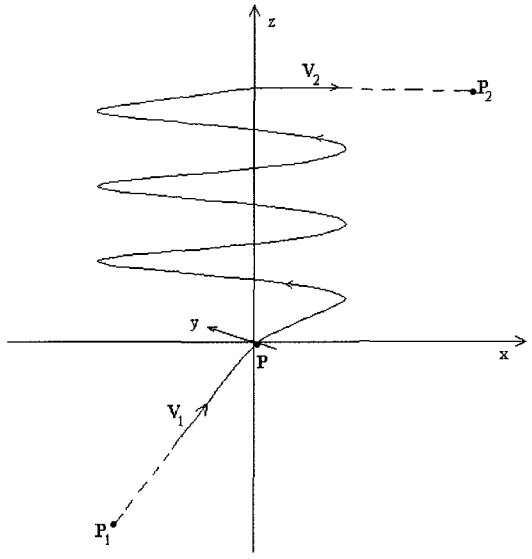


Figure 41. Vue de côté du segment ascendant, de la section hélicoïdale et du segment de droite horizontal pour rejoindre le point P_2 (reproduit de [37])

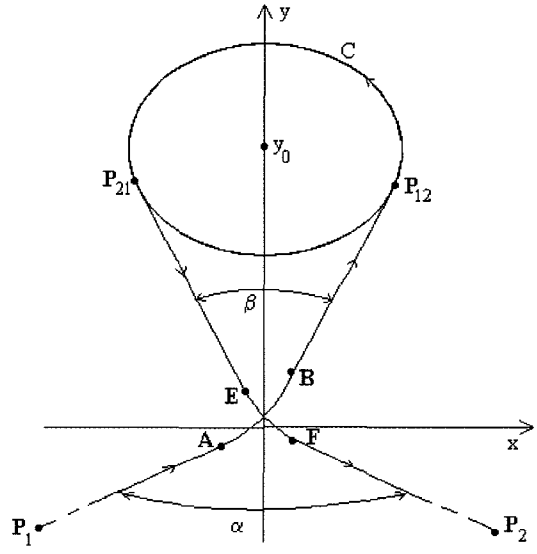


Figure 42. Vue de haut du segment ascendant, de la section hélicoïdale et du segment de droite horizontal pour rejoindre le point P_2 (reproduit de [37])

6.3.1 Construction de la trajectoire hélicoïdale

La construction de la trajectoire hélicoïdale se fait de façon semblable à celle de la connexion par cercle sur plateau horizontal et requiert un changement de système de coordonnées. Le cercle C de la section précédente est remplacé par une hélice H d'un même rayon. La vitesse ascensionnelle sur l'hélice est légèrement inférieure à la vitesse ascensionnelle maximale et calculée de façon à ce que l'hélice arrive au point $\boldsymbol{P_{21}}$ exactement à la bonne altitude. La série de points formant l'hélice est obtenue d'après l'équation suivante :

$$\boldsymbol{x_H}(t) = [R_H\, cos(\omega t),\quad y_0 + R_H\, sin(\omega t),\quad z_0 + V_3 t] \tag{30}$$

où R_H est le rayon de l'hélice; ω est la vitesse angulaire; t est le temps; y_0 est la coordonnée Y du centre de l'hélice; et z_0 est la coordonnée Z de l'extrémité inférieure de l'hélice.

6.3.2 Critères de faisabilité

Les conditions à vérifier pour déterminer la faisabilité de la trajectoire hélicoïdale sont exactement les même que celles pour la connexion par cercle horizontal. Il faut d'abord s'assurer que les points $\boldsymbol{P_1}$, $\boldsymbol{P}$ et $\boldsymbol{P_2}$ sont suffisamment distancés pour permettre la construction des arcs de cercle C_1 et C_2. L'altitude de l'arc C_1 et de la première rotation de l'hélice doit être vérifiée pour détecter toute violation de l'altitude minimum de vol. Finalement, nous calculons la quantité de carburant nécessaire pour voler la trajectoire et ajustons le terme $C_{carburant}$ de la fonction de coût. Si l'une des trois conditions n'est pas respectée, le segment de droite ne peut être remplacé par une trajectoire hélicoïdale et la trajectoire reste infaisable à cause de la trop grande puissance requise.

En conclusion, les trajectoires générées par les moteurs d'optimisation présentés aux chapitres 4 et 5 sont formées de segments de droite et contiennent donc des discontinuités dans leurs vitesses. Nous avons décrit dans ce chapitre comment nous utilisons des arcs de cercle, des cercles sur plateau horizontal et des hélices verticales pour lisser le trajet. La méthode utilisée provient directement de [37] et permet la vérification des contraintes de faisabilité en tout point. Notre fonction de coût, notre AG, notre OEP et notre fonction de lissage composent les éléments d'un module complet de planification de trajectoires pour les drones. Les trajectoires générées sont quasi optimales et respectent les contraintes dynamiques du drone. La suite de ce mémoire traite de la parallélisation des moteurs d'optimisation, de leur comparaison et de l'intégration du module de planification de trajectoires à un pilote automatique commercial.

Chapitre 7

Implémentation parallèle

En 1965, Gordon E. Moore publie un article dans le journal Electronics soulignant les progrès dans le domaine des semi-conducteurs et note que le nombre de composantes sur les circuits intégrés double chaque année depuis le début de la décennie [45]. En fait, l'intervalle auquel la densité des transistors double varie un peu et se tient plutôt vers 18 à 24 mois [46]. Encore aujourd'hui, cette estimation s'avère vraie et l'on y réfère maintenant par la loi de Moore. C'est en 2005, lorsque la course aux GHz plafonne aux alentours des 4 GHz que les concepteurs de circuits intégrés développent les premiers microprocesseurs x86 à deux cœurs afin de maximiser l'utilisation du nombre toujours grandissant de transistors. Bien établie et abordable, la technologie multicœur se retrouve de nos jours dans la majorité des ordinateurs personnels ou portables. Présentement à six cœurs, on prévoit que le nombre de cœurs sur un même microprocesseur augmentera plus ou moins selon la loi de Moore [47].

Dans ce chapitre, nous expliquons comment nous employons le paradigme de programmation parallèle « Instruction simple, données multiples » afin d'utiliser pleinement la technologie multicœur pour minimiser le temps de calcul des moteurs d'optimisation présentés antérieurement. L'accélération atteinte est quasi linéaire et permet le calcul des trajectoires en temps réel, ce qui n'est pas possible avec la version séquentielle. En plus du gain de performance, nous observons que la qualité des trajectoires obtenues avec les versions parallèles de nos algorithmes est supérieure à celle obtenue avec les versions séquentielles.

Tout au long de ce chapitre, nous employons les termes processus et processeurs. Comme il est défini dans [48], un processus est un programme en exécution ainsi que ses données (pile en mémoire, contenu des registres et compteur de programme). De son côté, le processeur est l'élément physique sur lequel s'exécute le processus. Un processeur unique peut exécuter un seul processus à la fois. Un processeur multicœur est en fait composé de plusieurs processeurs localisés sur un même circuit intégré et permet l'exécution simultanée de plusieurs processus. Un programme parallèle atteint habituellement la meilleure performance lorsque le nombre de processus est égal au nombre de processeurs ou cœurs.

7.1 Programmation parallèle en MATLAB

MATLAB [49] est un outil de développement et un langage de programmation de référence dans le domaine scientifique. Tel que souligné par les auteurs de [50], MATLAB permet le développement rapide de prototypes logiciels tout en assurant une excellente performance de calcul matriciel. Doté d'une multitude de boîtes à outils, MATLAB est utilisé dans plusieurs domaines techniques tels le traitement d'images et de signaux, les systèmes de contrôle, les finances, la simulation, etc. Depuis l'arrivée des microprocesseurs multicœurs, MATLAB aborde aussi le domaine du calcul haute performance avec la mise au point de deux boîtes à outils pour la programmation parallèle : soit le « Parallel Computing Toolbox (PCT) » pour les ordinateurs multicœurs à mémoire partagée et le « MATLAB Distributed Computing Server (DCS) » pour les grappes d'ordinateurs. Dans le cadre de notre recherche, nous utilisons le PCT pour paralléliser l'exécution des deux algorithmes de planification de trajectoires présentés antérieurement.

Le PCT de MATLAB contrôle l'exécution simultanée de plusieurs instances du moteur de calcul MATLAB et la communication entre ces processus. Quoique le PCT soit conçu pour les systèmes parallèles à mémoire partagée, il utilise l'implémentation MPICH2 de « Message Passing Interface (MPI) » et effectue la communication entre les processus à l'aide de messages comme dans un système à mémoire distribuée. En d'autres mots, les processus MATLAB ne partagent pas un espace de mémoire commune, mais communiquent à l'aide de messages qu'ils envoient au processus `mpiexec` qui joue le rôle d'ordonnanceur. Heureusement, le PCT abstrait cette communication et offre à l'utilisateur une série limitée, mais suffisante, de fonctions semblables à celle de MPI.

7.1.1 Programmation parallèle implicite (parfor)

Le PCT de MATLAB permet de paralléliser un programme implicitement ou explicitement. Le code est implicitement parallèle lorsque la communication et la synchronisation entre les processus ne sont pas explicitement définies par le programmeur. En MATLAB, une section de code parallèle implicite consiste en une boucle `parfor`. L'exécution des itérations de la boucle se fait en parallèle telle qu'organisée par le PCT. On retrouve à la figure 43 un exemple d'un programme MATLAB parallèle implicite pour calculer la somme d'une série de 1 000 nombres entre 0 et 100 à l'aide de 4 processus. On remarque à la ligne 10 la création des 4

processus et à la ligne 13, l'emploi de la boucle parallèle. Lors de l'utilisation de la boucle parallèle, une itération subséquente ne peut dépendre d'une itération précédente puisque le PCT ne garantit pas l'ordre dans lequel les itérations seront exécutées.

```
1.  function parallel_sum_parfor_example()
2.
3.      % Données initiales
4.      size = 1000;                          % dimension du vecteur de données
5.      num_processes = 4;                    % nombre de processus
6.      data = round(100*rand(1,size));       % vecteur de données
7.      sum = 0;                              % sommes des éléments du vecteur
8.
9.      % Initialisation des processus
10.     matlabpool('open','local',num_processes);
11.
12.     % Début de la section parallèle
13.     parfor i = 1:size
14.         sum = sum + data(i);
15.     end
16.     % fin de la section parallèle
17.
18.     fprintf('La somme des %d nombres est: %d\n',size, sum);
19. end
```

Figure 43. Exemple de programmation parallèle implicite en MATLAB PCT

Il est évident que la parallélisation implicite de PCT est facile à utiliser, mais quelque peu limitée. Adaptée pour des calculs simples tels que la simulation Monte-Carlo [50], cette technique peut difficilement être employée pour paralléliser l'AG ou l'OEP. De plus, les auteurs de [48] observent que l'accélération obtenue par la parallélisation implicite est significativement inférieure à celle obtenue par la parallélisation explicite.

7.1.2 Programmation parallèle explicite (spmd)

Pour un programme plus complexe que celui montré à la figure 43, il est nécessaire d'utiliser la parallélisation explicite afin de manuellement gérer le code exécuté, la communication et la synchronisation des processus. Nous présentons à la figure 44, le même programme qu'à la figure 43, mais utilisant la parallélisation explicite. Il est évident que cette seconde technique est beaucoup plus flexible, mais aussi plus complexe à utiliser. Comme à l'exemple précédent, les 4 processus sont créés à la ligne 10 et la section parallèle commence à la ligne 13 avec la commande `spmd` (Single Program, Multiple Data). Cette commande signifie que les variables précédemment définies sont dupliquées et uniques à chaque processus tandis que le code est commun à tous les processus. Par exemple, chaque processus possède leur propre copie de la variable `local_sum`. Malgré un programme unique, il est possible d'utiliser des énoncées

conditionnelles pour exécuter du code différent sur chacun des processus. Par exemple, aux lignes 15-19, tous les processus calculent la somme de leurs éléments respectifs du vecteur de données. Aux lignes 21-29, les processus 2, 3 et 4 envoient leur somme partielle au processus 1. Finalement, aux lignes 31-34, le processus 1 affiche la somme globale à l'écran. Il est important de noter que les commandes `labReceive` et `labSend` bloquent jusqu'à ce que les données soient envoyées ou reçues (tout comme les commandes `MPI_Send` et `MPI_Recv` de MPI).

```
1.  function parallel_sum_spmd_example()
2.
3.      % Données initiales
4.      size = 1000;                          % dimension du vecteur de données
5.      num_processes = 4;                    % nombre de processus
6.      data = round(100*rand(1,size));       % vecteur de données
7.      sum = 0;                              % sommes des éléments du vecteur
8.
9.      % Initialisation des processus
10.     matlabpool('open','local',num_processes);
11.
12.     % Début de la section parallèle
13.     spmd(num_processes)
14.
15.         % Tous les processus: calculent des sommes partielles
16.         local_sum = 0;
17.         for i = labindex:num_processes:size
18.             local_sum = local_sum + data(i);
19.         end
20.
21.         % Processus 1: addition des sommes partielles
22.         if labindex == 1
23.             sum = local_sum;              % somme calculée par le proc. 1
24.             for i = 2:num_processes       % sommes calculées par les autres
25.                sum = sum + labReceive(i); % processus.
26.             end
27.         else
28.             labSend(local_sum, 1);        % tous les processus envoient
29.         end                               % leur somme partielle au proc. 1
30.
31.         % Processus 1: affichage de la somme totale à l'écran
32.         if labindex == 1
33.             fprintf('La somme des %d nombres est: %d',size, sum);
34.         end
35.
36.     end
37.     % fin de la section parallèle
38. end
```

Figure 44. Exemple de programmation parallèle explicite en MATLAB PCT

Cet exemple utilise uniquement les fonctions `matlabpool`, `spmd`, `labSend`, `labReceive` et la variable de contrôle `labindex`, mais il existe une dizaine d'autres fonctions de PCT [51]. Dans le cas où l'on veut paralléliser l'AG et l'OEP, il est évident que l'approche explicite est mieux adaptée. De plus, les fonctions de PCT implémentent un sous-ensemble des fonctions de MPI.

L'approche utilisée pour paralléliser explicitement un programme MATLAB est donc la même que celle utilisée pour un programme MPI, qui est la référence dans le domaine de la programmation parallèle [52].

7.1.3 Évaluation de la performance de MATLAB « Parallel Computing Toolbox »

Avant de discuter la parallélisation de l'AG et de l'OEP, il est important d'évaluer la performance du compilateur utilisé, le MATLAB PCT. Dans le cas où la communication entre les processus est très rapide, une implémentation parallèle à grains fins, qui nécessite beaucoup de communication, est acceptable. Cependant, si la communication est relativement lente, une implémentation parallèle à gros-grains est préférable puisque la communication doit être minimisée.

Nous présentons au tableau 6 les spécifications du système test utilisé et aux figure 45, figure 46 et figure 47, les temps de communication pour les trois fonctions que nous utilisons dans notre implémentation parallèle de l'AG et de l'OEP. On remarque que plus le nombre de processus est grand, plus le temps de communication et long. Il est donc nécessaire de minimiser la communication si l'on veut assurer l'extensibilité de notre implémentation. Puisqu'il est intéressant de comparer la performance de MATLAB PCT à celle de MPICH2 (une des implémentations C les plus populaires de MPI), nous avons inclus les temps de communication pour ces deux technologies sur le premier graphique.

Tableau 6. Spécification de l'ordinateur multicœur utilisé

Modèle	Dell PowerEdge 2900
Processeurs	2 Intel Xeon quadruples cœurs E5310, 1.6 GHz
Mémoire vive	8 Go, DDR2, 667 MHz
OS utilisé pour MATLAB	Windows 7 Entreprise, 64 bits
Version de MATLAB	MATLAB 7.11.0.584 (2010 b) 64 bits Parallel Computing Toolbox v5.0
OS utilisé pour MPI	Linux Ubuntu 10.04
Version de MPI	gcc 4.4.3-4ubuntu5 mpich2 1.2.1.1-4

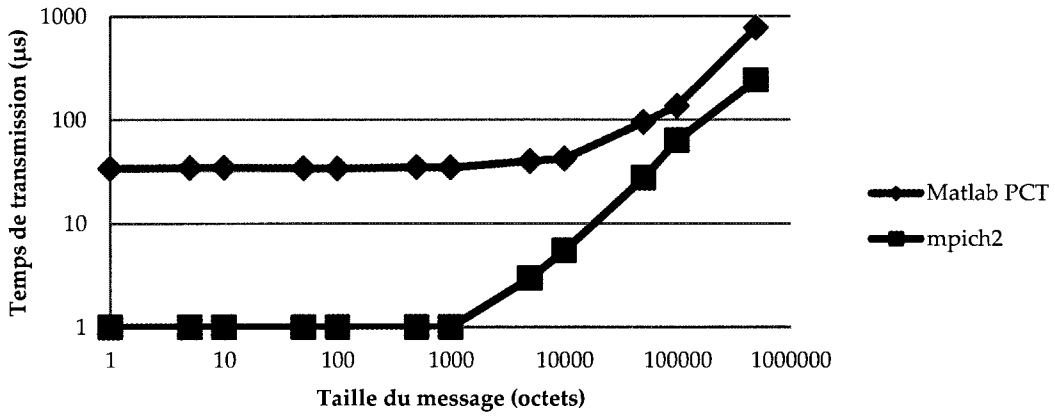


Figure 45. Temps d'exécution des commandes labSend/labReceive de MATLAB et de celles équivalentes, MPI_Send/MPI_Recv de MPICH2

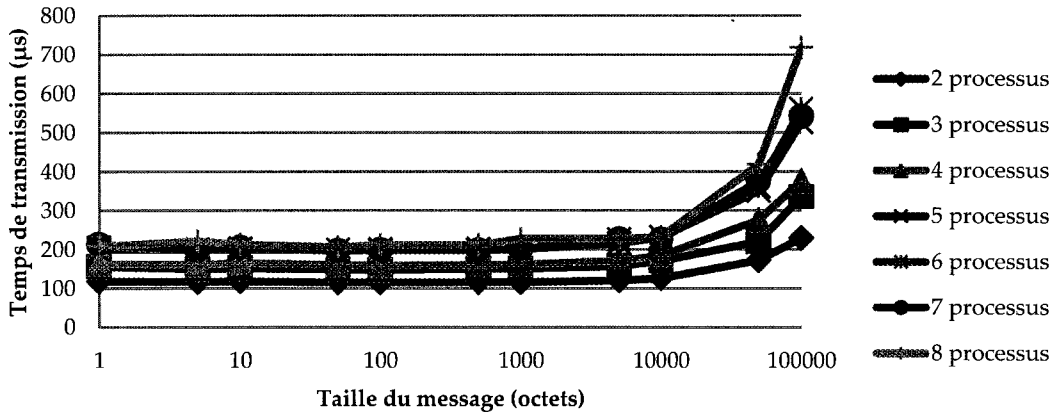


Figure 46. Temps d'exécution de la commande labBroadcast de MATLAB (équivalente à MPI_Bcast de MPI)

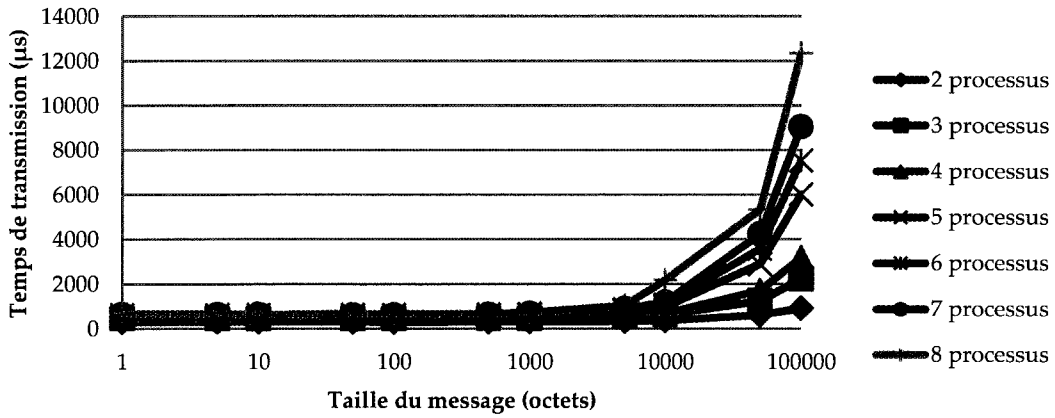


Figure 47. Temps d'exécution de la commande gcat de MATLAB (équivalente à MPI_Allgather de MPI)

7.2 Approches possibles pour paralléliser l'AG et l'OEP

Dans la littérature, il semble que plusieurs auteurs se sont attardés à la parallélisation de l'AG tandis que peu, à la parallélisation de l'OEP. En réalité, cette différence du nombre de publications sur chacun des sujets vient du fait que l'AG date des années 60 tandis que l'OEP, des années 90. Comme l'explique les auteurs de [53], les approches de parallélisation utilisées pour l'AG peuvent aussi bien être appliquées à l'OEP ou même à n'importe quelle heuristique d'optimisation basée sur les populations.

Tel que décrit dans [54], les différentes approches utilisées pour paralléliser l'AG peuvent se regrouper en quatre catégories : AG parallèle maître-esclave, AG parallèle à grains fins, AG parallèle à gros-grains et AG parallèle hybride. Les schémas illustrant la topologie de chacune des catégories sont présentés à la figure 48. La suite de cette section est un résumé de chacune des 4 catégories comme elles sont présentées dans [54].

L'AG parallèle maître-esclave n'utilise qu'une seule population. Le processus maître contrôle l'exécution du programme et utilise les processus esclaves pour effectuer le calcul du coût, de la reproduction et/ou des mutations en parallèle. Dans la plupart des implémentations, seul le calcul du coût est fait en parallèle puisqu'il demande habituellement une puissance de calcul beaucoup plus grande que les autres opérations. Un avantage de l'AG parallèle maître-esclave est qu'il ne requiert aucun matériel spécialisé; il peut être exécuté sur un système à mémoire partagée (un ordinateur multicœur) ou distribuée (une grappe d'ordinateurs). De plus, son comportement est identique à celui de l'AG séquentiel puisqu'une population unique est utilisée.

L'AG parallèle à grains fins utilise aussi une population unique. Cependant, le fonctionnement de l'algorithme est fortement lié au matériel utilisé. Habituellement implémentés sur un système massivement parallèle tel qu'une grille 2D, les chromosomes de la population sont distribués sur les multiples processeurs et peuvent uniquement interagir avec les chromosomes avoisinants, selon la topologie du matériel. Le comportement de l'AG est donc modifié.

L'AG parallèle à gros-grains utilise plusieurs populations qui évoluent simultanément sur chacun des processus. L'algorithme inclut un procédé de migration qui permet un échange de chromosomes entre les populations. La circonstance, la fréquence, la destination et le choix des

chromosomes pour la migration varient d'un auteur à l'autre. Le comportement de cet AG parallèle diffère de l'AG séquentiel. Tel que mentionné par l'auteur de [54], un nombre de migrations trop petit résulte en une convergence rapide, mais un résultat final inférieur. Cependant, il existe un nombre de migrations optimal où l'AG parallèle se comporte semblablement à l'AG séquentiel tout en minimisant la communication entre les processus. L'AG parallèle à gros-grains est adapté aux systèmes à mémoire partagée (un ordinateur multicœur) ou distribuée (une grappe d'ordinateur).

Finalement, l'AG parallèle hybride est un mixte des deux méthodes précédentes. Cet algorithme est conçu sur mesure, pour un problème spécifique, exécuté sur un système en particulier. Par exemple, la topologie au bas à droite de la figure 48 illustre un algorithme hybride sur un réseau interconnecté de quatre grilles 2D. L'AG parallèle hybride est très complexe et s'applique à des calculs scientifiques demandant une très grande puissance de calcul.

Dans la situation où nous voulons 1) paralléliser l'AG et l'OEP pour la planification de trajectoires, 2) maximiser l'utilisation des processeurs multicœurs et 3) minimiser le temps de calcul, il semble évident que l'approche maître-esclave et l'approche à gros-grains sont préférables.

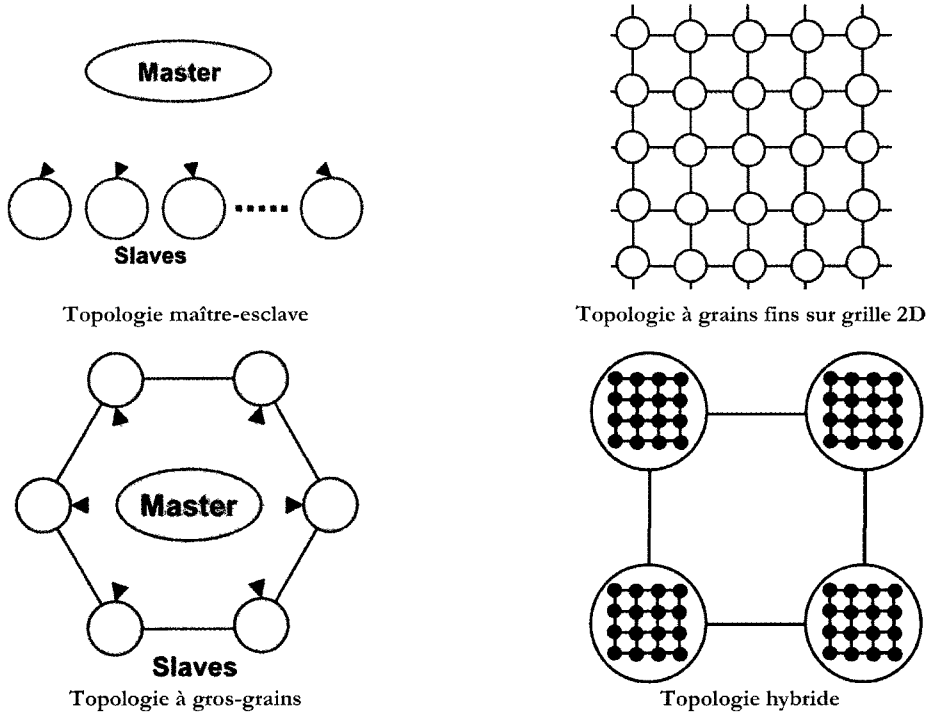


Figure 48. Les 4 principales catégories d'AG parallèles (reproduit de [55])

7.3 Parallélisation de l'algorithme génétique

Suite à la discussion ci-dessus, seules les approches maître-esclave et à gros-grains semblent avantageuses pour les systèmes multicœurs à mémoire partagée. Dans cette section, nous analysons chacune de ces méthodes afin de choisir celle qui est optimale pour l'AG appliqué à la planification de trajectoires pour les drones.

7.3.1 AG parallèle maître-esclave à population globale

Si l'on se base sur les temps d'exécution des principales fonctions de l'AG séquentiel tel que présenté dans le tableau 7, l'approche maître-esclave semble être tout à fait appropriée pour paralléliser l'AG. Il est évident que la fonction de coût nécessite une puissance de calcul bien plus grande que les autres fonctions. Or, quelques simples modifications au code séquentiel permettent de partager le calcul des coûts des chromosomes sur plusieurs processeurs. Par exemple, pour 128 chromosomes et 8 processus, chaque processus calcule le coût de 16 chromosomes à chaque génération. Cette technique utilise une seule population globale, ne modifie aucunement le comportement de l'AG, est facile à implémenter et semble possiblement apporter une accélération de 8. On retrouve à la figure 49 le diagramme de fonctionnement de cet AG parallèle maître-esclave avec population globale. Ce diagramme montre deux processus, mais notre implémentation fonctionne avec n'importe quel nombre de processus.

Tableau 7. Temps d'exécution des principales fonctions de l'AG séquentiel pour 128 chromosomes, 100 générations et une trajectoire à 8 points de passage

Fonctions	Temps d'exécution de la version séquentielle avec optimisations (s)
Algorithme génétique	36.605
Calcul du coût	30.221
Sélection	0.016
Reproduction	1.097
Mutation	1.811
Remplacement	0.132
Lissage	0.031

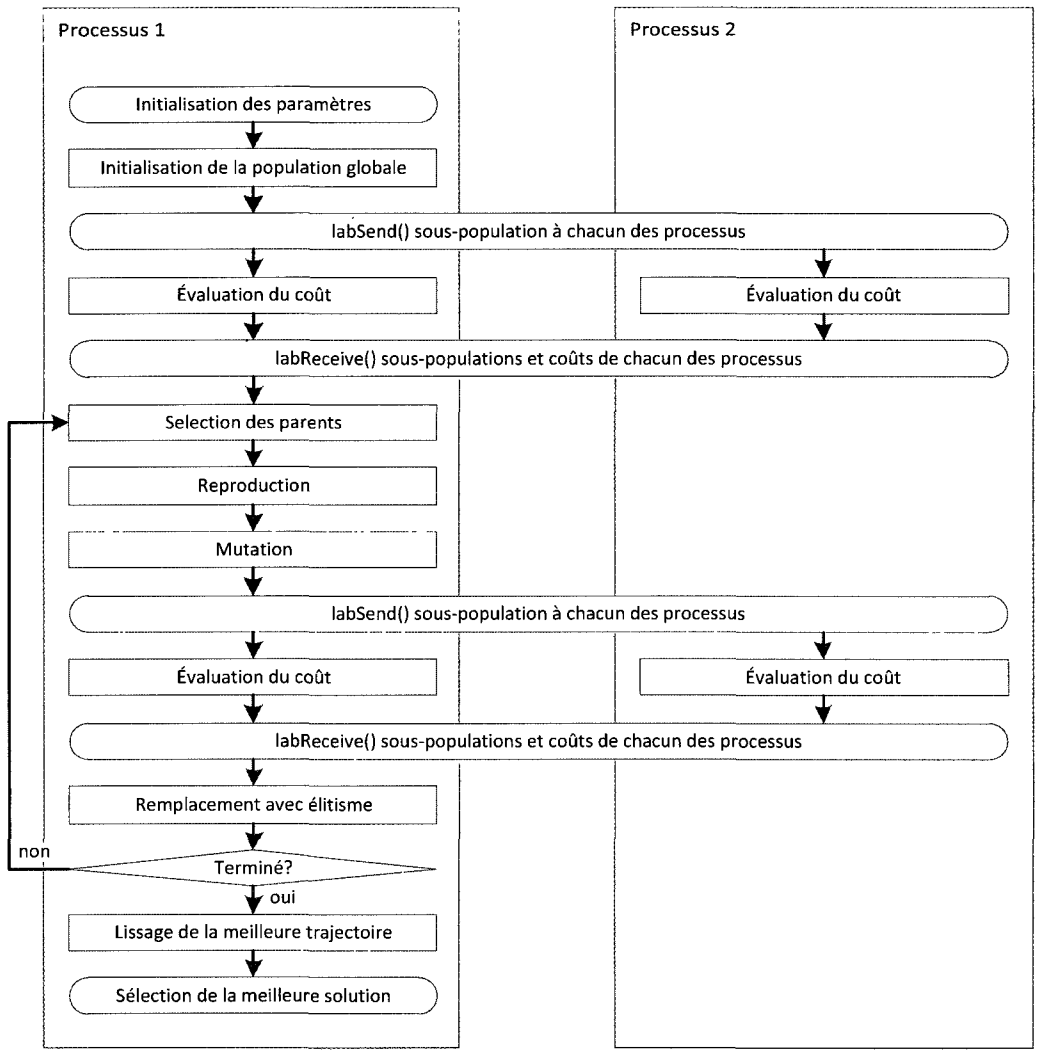


Figure 49. Diagramme de fonctionnement de notre implémentation parallèle maître-esclave de l'AG

7.3.1.1 Surdébit causé par la communication entre les processus

En fait, la discussion ci-dessus est incomplète puisque nous avons omis deux éléments importants qui influent sur l'accélération : le temps de communication et la section intrinsèquement séquentielle. Analysons d'abord la communication introduite par la parallélisation et son effet sur l'accélération de notre implémentation de l'AG maître-esclave. Reprenons l'exemple où nous répartissons le calcul du coût des 128 chromosomes sur huit processeurs. Dans ce scénario, le processus 1 doit envoyer 16 chromosomes aux processus 2 à 8. Les 8 processus calculent ensuite simultanément le coût de leurs 16 chromosomes et, finalement,

les processus 2 à 8 retournent leurs résultats au processus 1. Les processus 2 à 8 doivent aussi retourner leurs 16 chromosomes au processus 1 puisque ceux-ci contiennent des valeurs temporaires comme nous l'avons expliqué au chapitre 4. D'après les équations (31) et (32), le processus 1 doit envoyer 7 messages de 14 080 octets et recevoir 7 messages de 128 octets et 7 autres messages de 14 080 octets.

$$16\ chromosomes * \frac{10\ points}{chromosome} * \frac{11\ doubles}{points} * \frac{8\ octets}{double} = 14\ 080\ octets \quad (31)$$

$$16\ coûts * \frac{1\ double}{coût} * \frac{8\ octets}{double} = 128\ octets \quad (32)$$

On doit ensuite se référer à la figure 45 pour déterminer que le temps nécessaire pour transmettre un message contenant 16 chromosomes est de 0.051 ms et celui nécessaire pour transmettre un message contenant 16 coûts est de 0.034 ms. Or, comme nous l'avons expliqué au chapitre 4, chaque chromosome est représenté par un ensemble de cellules de matrices afin de permettre une longueur variable des chromosomes. Puisque le PCT de MATLAB ne permet pas la transmission d'ensembles de cellules, les chromosomes doivent être empaquetés dans une matrice avant d'être envoyé et dépaquetés après être reçu. Par expérimentation, nous avons mesuré que le temps d'empaquetage des 16 chromosomes est de 0.822 ms et celui du dépaquetage, de 0.456 ms. Le temps de transmission des 16 chromosomes est donc 1.329 ms.

L'accélération que l'on obtiendrait si l'on distribuait le calcul des 128 chromosomes sur 8 processeurs en tenant compte de la communication supplémentaire peut maintenant être dérivée comme suit :

1) Calcul du temps d'exécution de la fonction de coût $t_{coût}$ pour 1 chromosome d'après information du tableau 7 :

$$t_{coût} = \frac{30.221\ s}{128\ chromosomes * 100\ générations} = 2.361\ ms \quad (33)$$

2) Calcul du temps nécessaire t_o (temps de surdébit) pour que le processus 1 transmette les 7 messages contenant les chromosomes et reçoive les 7 messages contenant les coûts et les 7 messages contenant les chromosomes avec les valeurs temporaires modifiées :

$$\begin{aligned} t_o &= 7 * 1.329\,ms + 7 * 0.034\,ms + 7 * 1.329\,ms \\ &= 18.844\,ms \end{aligned} \tag{34}$$

3) Calcul du temps d'exécution séquentielle T_s et parallèle T_p de la fonction de coût pour 128 chromosomes :

$$\begin{aligned} T_s &= 128 * t_{coût} \\ &= 128 * 2.361\,ms \\ &= 302.208\,ms \end{aligned} \tag{35}$$

$$\begin{aligned} T_p &= 16 * t_{coût} + t_o \\ &= 16 * 2.361\,ms + 18.844\,ms \\ &= 56.620\,ms \end{aligned} \tag{36}$$

4) Calcul de l'accélération S_p tel que définie dans [47] :

$$\begin{aligned} S_p &= \frac{T_s}{T_p} \\ &= \frac{302.208\,ms}{56.620\,ms} \\ &= 5.34 \end{aligned} \tag{37}$$

En raison de la communication engendrée par la parallélisation de l'AG suivant l'approche maître-esclave, nous obtenons une accélération de 5.34 sur 8 processeurs, ce qui est assurément inférieur à l'accélération de 8 proposée antérieurement. En fait, cette estimation est encore trop optimiste puisque la parallélisation engendre d'autres surdébits tels que la synchronisation des processus.

7.3.1.2 Loi d'Amdahl

Il est expliqué dans [47] qu'on ne peut arbitrairement réduire le temps d'exécution d'un programme parallèle en ajoutant des processeurs. Comme nous l'avons vu, la communication engendrée par la parallélisation engendre un surdébit important qui réduit l'accélération possible. Ce surdébit grandit habituellement avec le nombre de processus utilisés. Le deuxième élément qui limite l'accélération possible est la partie intrinsèquement séquentielle du programme (comme l'initialisation du programme, la lecture d'un fichier d'entrée, l'écriture d'un fichier de sortie, etc.). Cette limitation est expliquée par la loi d'Amdahl [47] énoncée à l'équation (38) ou n est le nombre de processeurs.

$$S_p(n) = \frac{T_s}{f * T_s + \frac{1-f}{n} T_s} = \frac{1}{f + \frac{1-f}{n}} = \frac{n}{1 + (n-1) * f} \leq \frac{1}{f} \tag{38}$$

Tel qu'illustré à la figure 50, si un programme séquentiel prend T_s temps à exécuter, qu'une fraction f de ce programme est intrinsèquement séquentielle, que l'autre fraction $1 - f$ est parallélisable, l'accélération maximale possible est inférieure ou égale à $1/f$.

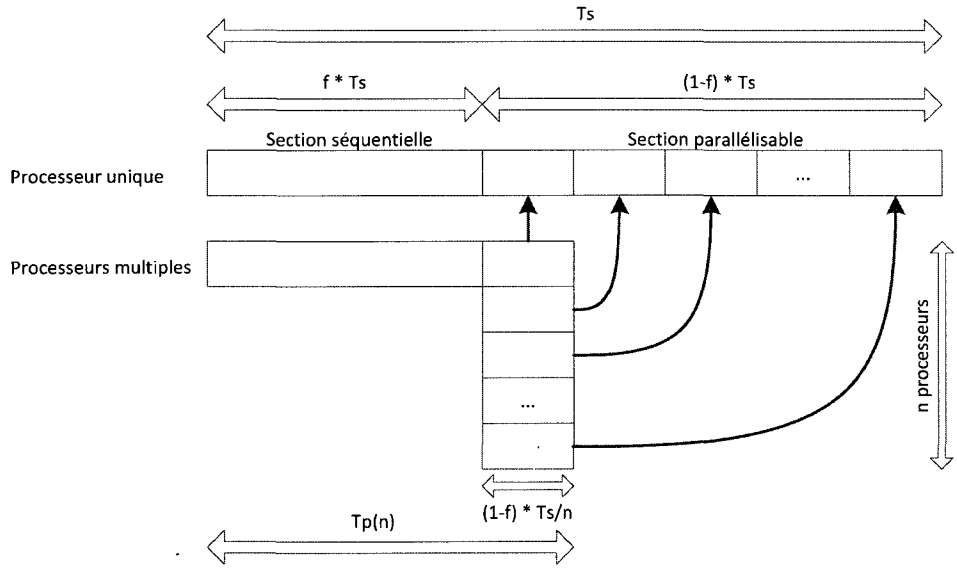


Figure 50. Illustration de la loi d'Amdahl pour n processeurs sans surdébit (reproduit de [56])

Dans le cas de l'AG parallèle maître-esclave, on peut déterminer le temps d'exécution de la section séquentielle à partir du tableau 7. On peut ensuite calculer l'accélération maximum possible d'après la loi d'Amdahl comme suit :

$$f = \frac{36.605 - 30.221}{36.605} = 0.1744 \tag{39}$$

$$S_p(n) = \frac{n}{1 + (n-1) * f} \leq \frac{1}{f} = \frac{1}{0.1744} = 5.73 \tag{40}$$

$$S_p(8) = \frac{8}{1 + (8-1) * 0.174} = 3.60 \tag{41}$$

Ceci dit, peu importe le nombre de processeurs utilisés, l'accélération maximale possible pour l'AG parallèle maître-esclave présenté antérieurement est de 5.73. Pour 8 processeurs, l'accélération n'est que de 3.60. De plus, il est possible de combiner le surdébit dû à la communication et la loi d'Amdahl afin de trouver une meilleure estimation de l'accélération :

$$S_p(n) = \frac{T_s}{f * T_s + \frac{1-f}{n} T_s + T_o} \tag{42}$$

où nous ajoutons le surdébit T_o au temps d'exécution parallèle

$$or,\ T_o = 100 * 0.018844\ s = 1.8844\ s \tag{43}$$

puisque T_o est le surdébit dû à la communication pour 100 générations

$$alors,\ S_p(8) = \frac{36.605}{0.174 * 36.605 + \frac{1 - 0.174}{8} 36.605 + 1.8844} = \frac{36.605}{12.046} = 3.04 \tag{44}$$

Par analyse, nous venons de démontrer que l'accélération que l'on obtiendrait en parallélisant l'AG par approche maître-esclave est de 3.04 (pour 128 chromosomes, 100 générations, sur l'ordinateur octocœurs spécifié au tableau 6). Cette accélération est semblable à celle obtenue par les auteurs de [52], mais n'est pas très avantageuse.

7.3.2 AG parallèle à gros-grain avec populations locales et migrations

Notre deuxième implémentation de l'AG parallèle consiste à la parallélisation gros-grains avec multiples populations locales. Comme illustré à la figure 51, chaque processus simule l'évolution d'une population locale. Cette approche est beaucoup plus avantageuse que la précédente puisque la section intrinsèquement séquentielle est pratiquement nulle et la loi

d'Amdahl permet alors une accélération linéaire. Afin de maintenir un comportement semblable à celui de l'AG séquentielle à population globale, nous avons ajouté un mécanisme de migration où tous les chromosomes de chacun des processus sont envoyés au processus 1 qui les redistribue ensuite aléatoirement.

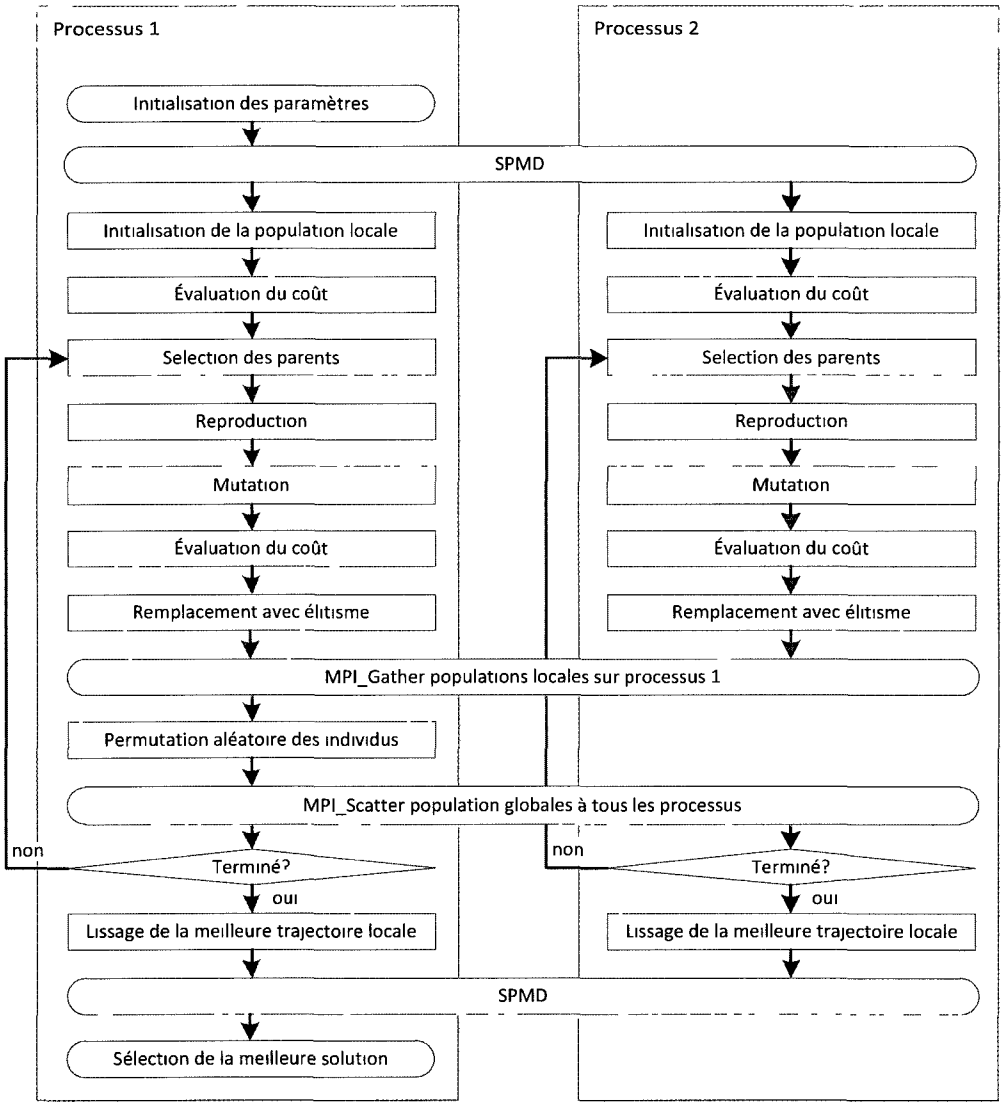


Figure 51. Diagramme de fonctionnement de notre implémentation parallèle à gros-grains de l'AG

Or, afin de limiter la communication et d'éviter le problème rencontré avec l'approche maître-esclave, il faut que ce processus de migration soit appliqué un nombre limité de fois pendant l'exécution de l'AG. Nous avons donc calculé une trajectoire 200 fois en utilisant l'AG maître-

esclave (population globale) et l'AG à gros-grains (populations locales) avec 0, 5, 10, 15 et 20 migrations. Les distributions des coûts des trajectoires calculées par chacune des implémentations sont présentées à la figure 52 à l'aide de la fonction « boxplot » de MATLAB [57]. Cette fonction représente une distribution de données par un rectangle à pattes dont la base représente le 25e centile (soit q_1), la division, le 50e (soit q_2) et le haut, le 75e (soit q_3). Les pattes s'étendent jusqu'aux valeurs extrêmes sans couvrir les données aberrantes. Une donnée est considérée aberrante lorsqu'elle est plus petite que $q_1 - 1.5 * (q_3 - q_1)$ et plus grande que $q_3 - 1.5 * (q_3 - q_1)$. À la figure 52, on remarque que l'AG à gros-grains (populations locales) avec un petit nombre de migrations produit des trajectoires de qualité semblable et même supérieure à l'AG original (population globale). En fait, la moyenne du coût des trajectoires obtenues par l'AG original est de 0.7041 tandis que celle de l'AG parallèle à gros-grains avec 5 migrations est de 0.5584. Nous confirmons ensuite la signifiance statistique de cette différence avec le test-T [58] et obtenons une valeur p de 0.0001 ce qui signifie qu'il y a 0.01 % de chance que cette différence soit attribuable au hasard. Nous pouvons donc conclure que la parallélisation à gros-grains permet non seulement de réduire le temps de calcul, mais aussi d'améliorer significativement la qualité des solutions. En fait, comme l'explique les auteurs de [59], l'utilisation de populations locales et indépendantes améliore l'exploration de l'espace de recherche et permet plus facilement d'échapper aux minima locaux tandis que les migrations améliorent la convergence de l'algorithme.

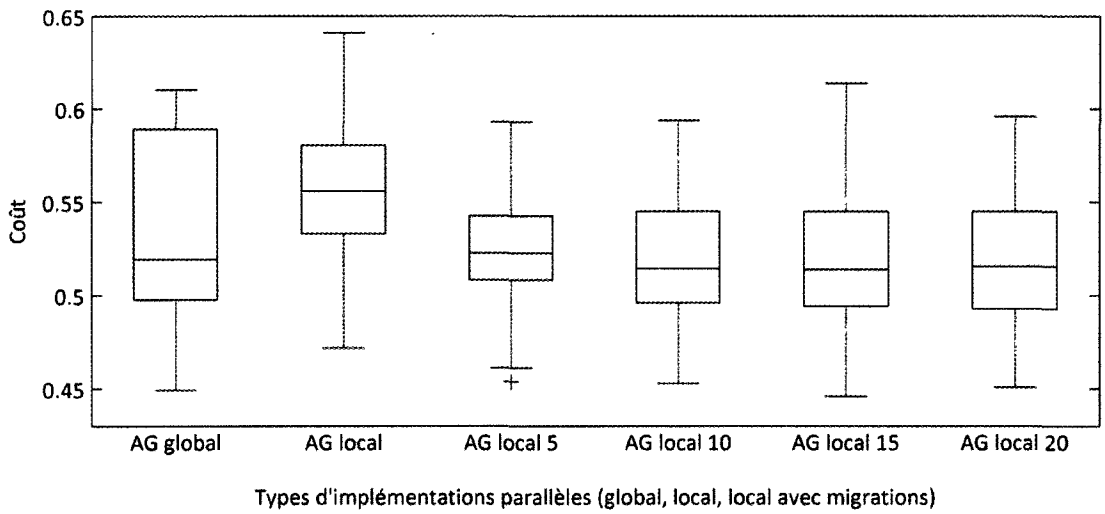


Figure 52. Coût des trajectoires calculées par différentes implémentations parallèles de l'AG (128 chromosomes, 350 générations, 6 processeurs, terrain 1, scénario 3, trajectoires calculées 200 fois pour chaque implémentation)

7.3.3 Analyse de performance de notre version parallèle de l'AG

Puisque notre implémentation de l'AG parallèle à gros-grains ne contient pratiquement aucune section intrinsèquement séquentielle et minimise grandement la fréquence des communications entre les processus, il n'est pas nécessaire d'estimer l'accélération possible, mais bien plus intéressant de la mesurer expérimentalement. Les mesures du temps de calcul, de l'accélération et de l'efficacité (ratio accélération sur nombre de processus) sont présentées aux figure 53, figure 54 et figure 55. Afin d'obtenir des résultats corrects, nous avons effectué chaque test dix fois et nous avons utilisé la médiane des temps mesurés pour construire les graphes. Ces tests de performance ont été effectués sur l'ordinateur multicœurs décrit au tableau 6 avec les paramètres listés au tableau 8.

Tableau 8. Paramètres de configuration de l'AG utilisés pour les tests de performance

Paramètres	Valeurs
Dimensions du terrain	500 x 500 unités
Nombre de chromosomes	128 et 256
Nombre de générations	100, 200 et 300
Nombres de migrations	5
Longueur minimum des chromosomes	8
Longueur maximum des chromosomes	8
Taux de mutations	80 %
Taux de survie (élitisme)	10 %
Vérifier le coût du lissage à partir de X% du processus évolution	80 %
Vérifier le coût du lissage pour les X% meilleurs chromosomes	25 %

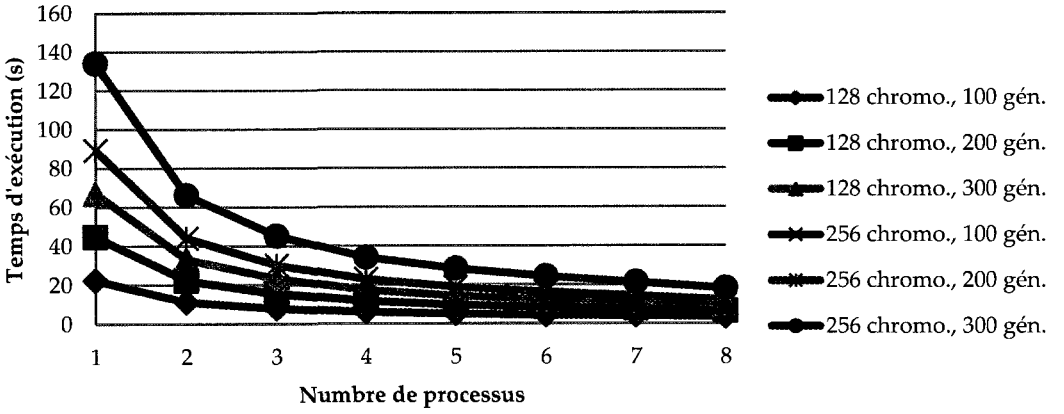


Figure 53. Temps d'exécution de l'AG parallèle à gros-grains pour différentes tailles de travail en fonction du nombre de processus utilisés

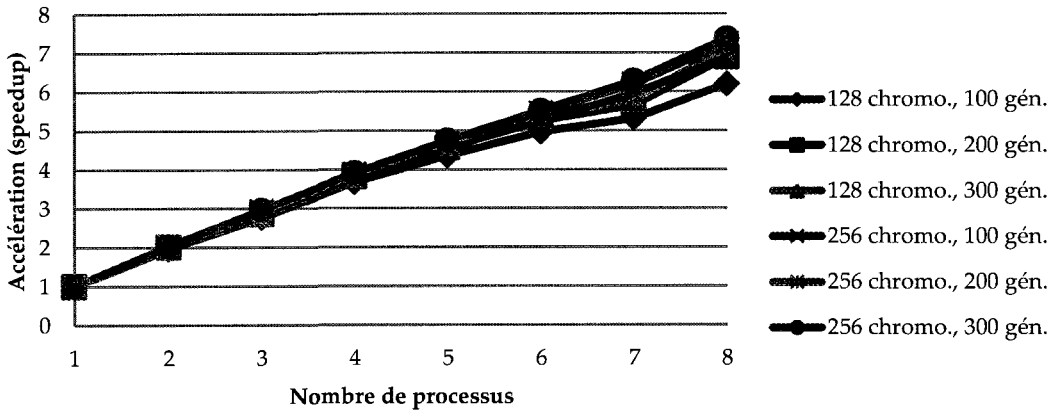


Figure 54. Accélération de l'AG parallèle à gros-grains pour différentes tailles de travail en fonction du nombre de processus utilisés

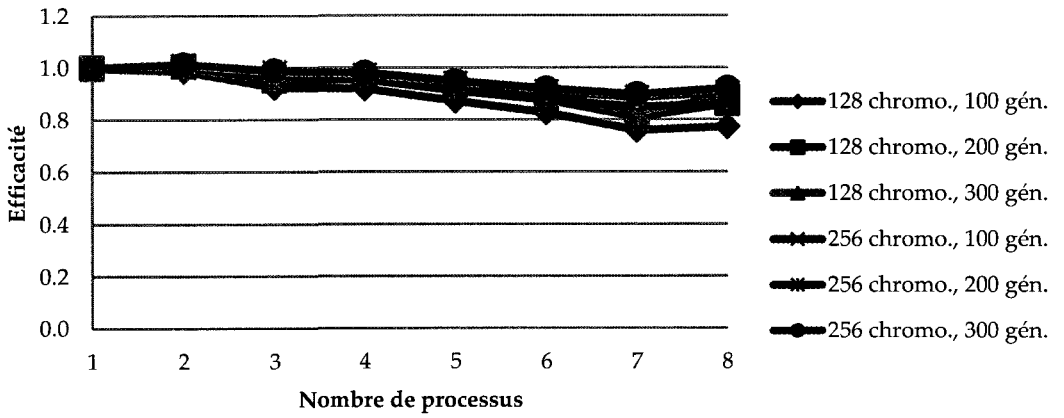


Figure 55. Efficacité de l'AG parallèle à gros-grains pour différentes tailles de travail en fonction du nombre de processus utilisés

Ces mesures démontrent une accélération impressionnante. Notre implémentation parallèle de l'AG permet d'effectuer le calcul de trajectoire avec une rapidité supérieure à 7 fois celle de la version séquentielle. Or, au chapitre 4, nous avons apporté plusieurs optimisations à la version séquentielle de l'AG afin d'accélérer son exécution. Si l'on comparait notre version parallèle à la version séquentielle naïve (colonne de gauche du tableau 3), nous obtiendrions une accélération de 23.6.

La loi d'Amdahl propose une approche plutôt pessimiste à la parallélisation. Cette loi assume que l'objectif de la parallélisation est de minimiser le temps de calcul pour un travail fixe. Or, comme nous l'avons vu, même avec une petite section intrinsèquement séquentielle (0.1744 dans le cas de l'AG parallèle maître-esclave), l'accélération possible est grandement limitée, peu importe le nombre de processeurs utilisés. Dans le cas de notre implémentation de l'AG parallèle à gros-grains, il existe aussi une section intrinsèquement séquentielle (l'entrée des données, l'initialisation des paramètres et la sortie des données) et l'accélération plafonnerait si l'on augmentait le nombre de processeurs. On se pose alors la question : est-il nécessaire de développer des systèmes massivement parallèles? La réponse est oui! Le but de la parallélisation n'est pas toujours de minimiser le temps de calcul, mais peut aussi être de maximiser la quantité de calcul effectué dans un temps fixe. Par exemple, dans le cas de la planification de trajectoires pour les drones, il n'est pas nécessaire de calculer une nouvelle trajectoire toutes les secondes; il est plus avantageux de calculer une meilleure trajectoire dans un temps relativement court (ex. : 10 s). Il existe en fait une seconde mesure de performance pour les programmes parallèles : l'accélération ajustée, aussi appelée accélération pour temps fixe. L'accélération ajustée ($S'_p(w,n)$) est définie comme suit [47]:

$$S'_p(w,n) = \frac{T(w,1)}{T(w,n)} \tag{45}$$

où $T(w,n)$ est le temps nécessaire pour effectuer le travail w sur n processeurs et $T(w,1)$ est le temps nécessaire pour effectuer le travail w sur 1 seul processeur.

Or, lorsque le surdébit est négligé, la loi de Gustafson dit que l'accélération ajustée est une fonction linéaire du nombre de processeurs n lorsque le travail w est ajusté pour maintenir un temps d'exécution fixe [47]. Dans le cas de notre implémentation de l'AG parallèle à gros-gains, on mesure l'accélération ajustée en comptant le nombre de générations simulées par l'AG

parallèle dans un temps fixe et on compare ce temps fixe à celui nécessaire à l'AG séquentielle pour simuler ce même nombre de générations. Les graphes du nombre de générations simulées et de l'accélération ajustée pour notre implémentation de l'AG parallèle à gros-grains se trouvent aux figure 56 et figure 57. On observe que l'accélération ajustée est semblable à l'accélération présentée ci-dessus. Cependant, si on exécutait l'AG parallèle sur un plus grand nombre de processeurs sans augmenter la quantité de travail, l'accélération serait très mauvaise puisque le temps d'exécution de la section intrinsèquement séquentielle et le surdébit dû à la communication seraient très grands comparé au temps d'exécution total. De son côté, l'accélération ajustée resterait bonne puisqu'on maintient un temps d'exécution fixe en augmentant la quantité de travail.

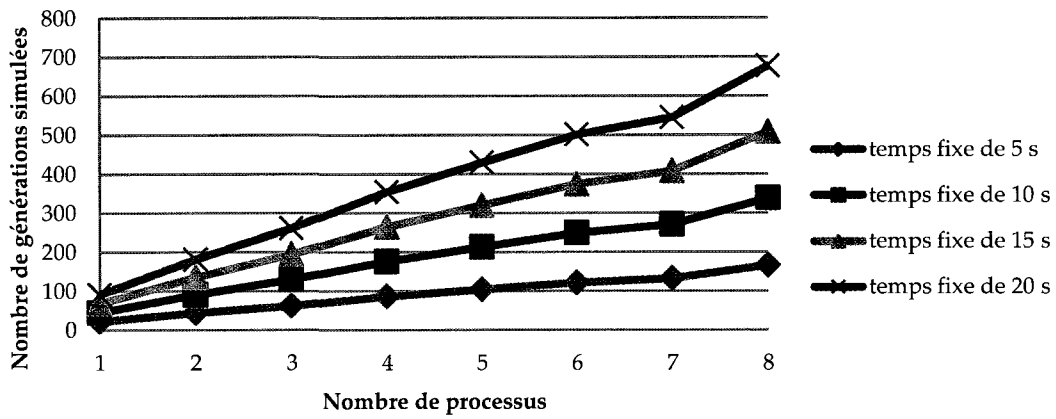


Figure 56. Nombre de générations simulées par l'AG parallèle à gros-grains pour différents temps fixes en fonction du nombre de processus utilisés

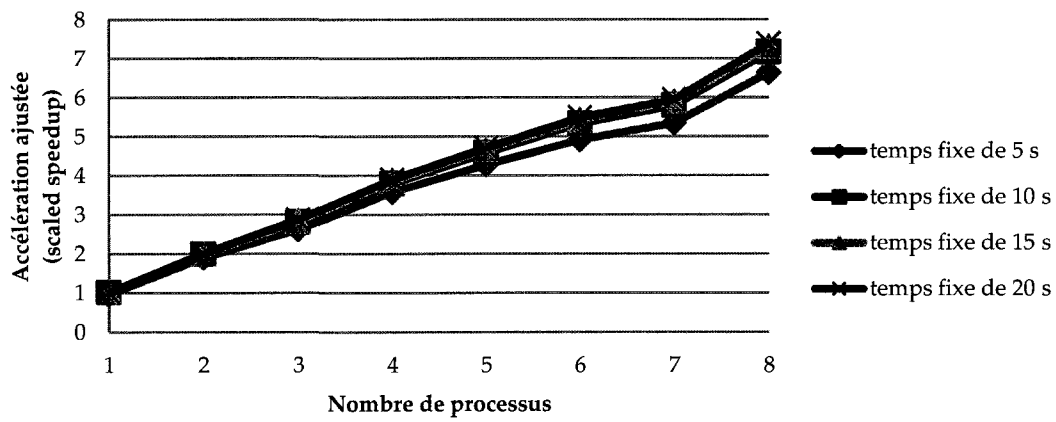


Figure 57. Accélération ajustée de l'AG parallèle à gros-grains pour différents temps fixes en fonction du nombre de processus utilisés

7.4 Parallélisation de l'optimisation par essaim de particules

Comme discuté précédemment, la parallélisation de l'OEP peut se faire similairement à celle de l'AG. Nous présentons dans cette section deux approches possibles et expliquons pourquoi la seconde est préférable. Puisque nous avons déjà longuement discuté du langage de programmation utilisé, des limitations à la parallélisation et de différentes mesures de performance, cette section est beaucoup plus brève.

7.4.1 OEP parallèle avec essaim global

Contrairement à l'AG, l'OEP est beaucoup plus simple à paralléliser. D'après les équations présentées au chapitre 5, à chaque itération, la position d'une particule est ajustée d'après sa meilleure position et la meilleure position de l'essaim. Pour paralléliser l'OEP, il suffit de

1) distribuer les particules sur différents processeurs pour former des essaims locaux; et
2) développer un processus de communication entre les processus pour identifier la meilleure position parmi tous les processus et distribuer cette meilleure position à tous les processus à chaque itération.

Dans notre cas, chaque processus utilise la commande `gcat` (équivalant à `MPI_All_Gather` de MPI) pour échanger leur meilleure position. Après cette commande, chaque processus détient les meilleures positions de tous les processus et utilise une boucle pour les comparer et sélectionner la meilleure position globale. Chaque processus peut ensuite mettre à jour la position de ses particules. Même si les particules sont distribuées sur plusieurs processus, il n'existe en fait qu'un essaim global puisque la communication entre chaque processus est effectuée à chaque itération. Cette approche ne modifie aucunement le comportement de l'OEP séquentielle, minimise la communication puisque seulement une particule par processus est échangée à chaque itération et minimise la section intrinsèquement parallèle puisque toutes les étapes de l'OEP sont parallélisées. Le diagramme de fonctionnement de notre implémentation est présenté à la figure 58.

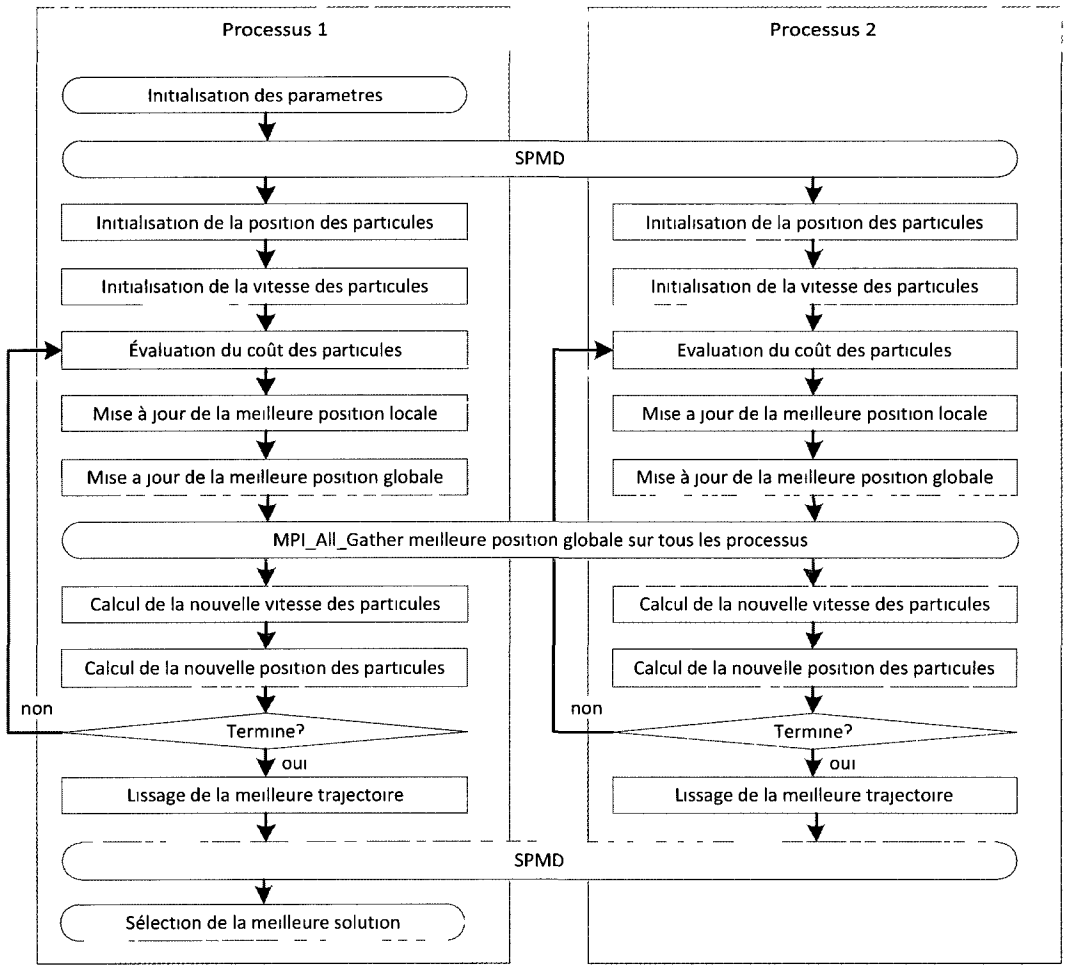


Figure 58. Diagramme de fonctionnement de notre implémentation de l'OEP parallèle avec essaim global

7.4.2 OEP parallèle avec essaims locaux et migrations

Bien que l'OEP soit par nature facile à paralléliser sans modifier son comportement, il est intéressant d'évaluer une deuxième approche : OEP parallèle avec essaims locaux et migrations. Le diagramme de fonctionnement de cette deuxième approche est présenté à la figure 59. Le principe est identique à celui développé pour l'AG parallèle à gros-grains : chaque processus contrôle le mouvement de son essaim indépendamment des autres processus et une migration aléatoire de toutes les particules entre tous les processus est appliquée à une fréquence fixe.

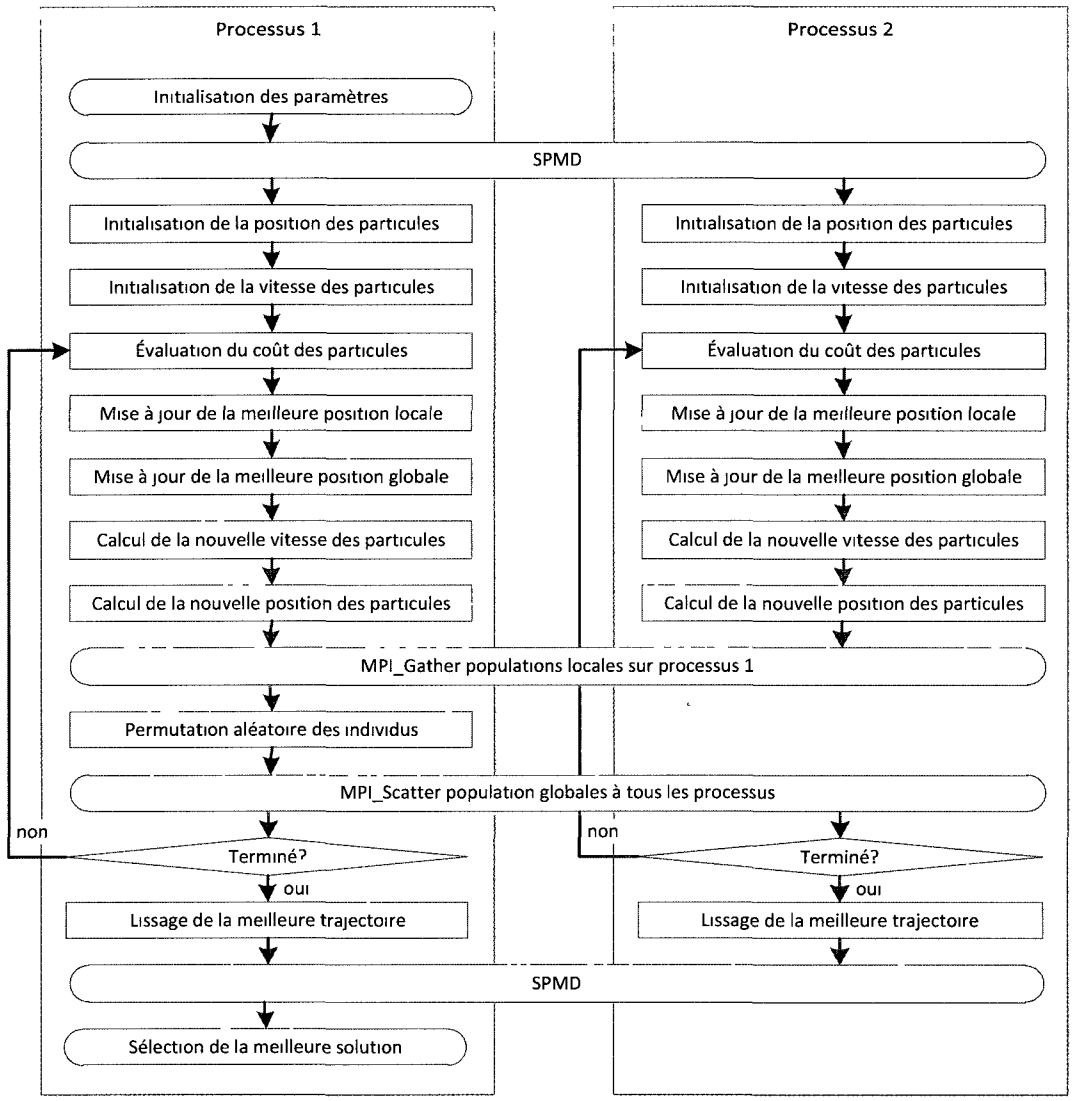


Figure 59. Diagramme de fonctionnement de notre implémentation de l'OEP parallèle avec essaims locaux et migrations

Puisque la section intrinsèquement séquentielle est minime et que la communication est limitée, la performance de cette deuxième approche devrait être semblable à la première. Tel qu'illustré à la figure 60 à l'aide de la fonction « boxplot » de MATLAB [57], il est intéressant d'observer que la qualité des trajectoires calculées par l'OEP parallèle avec essaims locaux et 5 migrations est supérieure à celle obtenue par la version parallèle globale de l'OEP. En fait, la moyenne du coût des trajectoires obtenues par l'OEP parallèle local avec 5 migrations est de 0.9483 tandis que celle de l'OEP global est de 5.9564. Le test-T [58] confirme que cette différence est statistiquement significative avec une valeur p de 4.30^{E}-22. Tous comme pour

l'AG, l'utilisation d'essaims locaux permet une meilleure exploration tandis que les migrations améliorent la convergence. L'OEP parallèle avec essaims locaux et migrations est préférable et donc la méthode que nous adoptons.

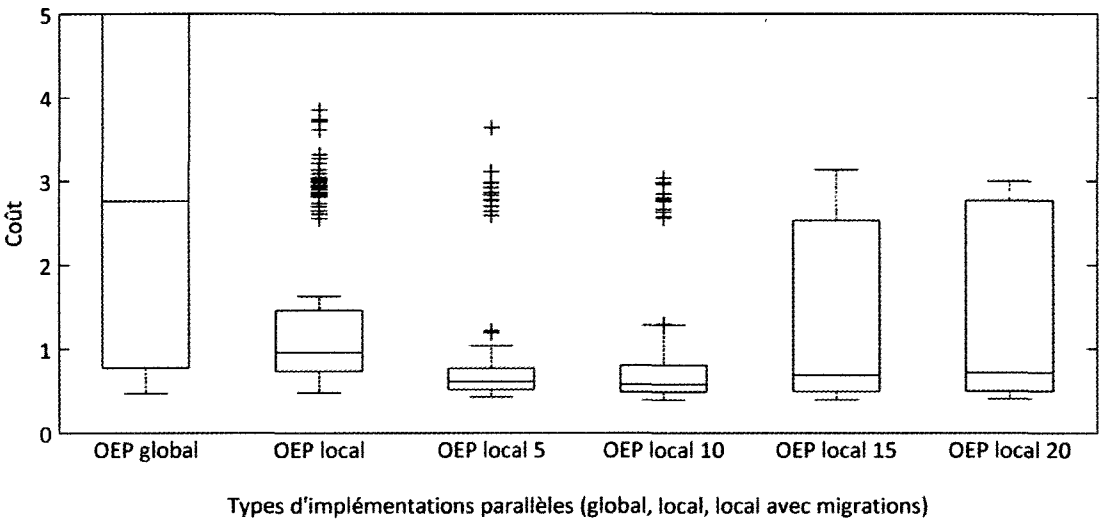


Figure 60. Coût des trajectoires calculées par différentes implémentations parallèles de l'algorithme d'OEP (128 particules, 350 itérations, 6 processus, terrain 1, scénario 3, trajectoires calculées 200 fois pour chaque implémentation)

7.4.3 Analyse de performance de notre version parallèle de l'OEP

Comme pour l'AG, nous soumettons notre implémentation de l'OEP parallèle à une analyse complète de performance. Le temps d'exécution est illustré à la figure 61, l'accélération à la figure 62, l'efficacité à la figure 63, le nombre d'itérations pour un temps fixe à la figure 64 et l'accélération ajustée à la figure 65. Ces tests ont été effectués sur l'ordinateur multicœur décrit au tableau 6 avec les paramètres listés au tableau 9. Tout comme l'AG, l'OEP parallèle permet d'effectuer le calcul de trajectoire avec une rapidité supérieure à 7 fois celle de la version séquentielle. Il est aussi intéressant d'observer que pour un temps fixe, avec 8 processus, l'AG est capable de simuler 50 % plus d'itérations que l'OEP pour le même nombre de particules.

Tableau 9. Paramètres de configuration de l'OEP utilisé pour les tests de performance

Paramètres	Valeurs
Dimensions du terrain	500 x 500 unités
Nombre de particules	128 et 256
Nombre d'itérations	100, 200 et 300
Nombres de migrations	5
Longueur des chromosomes	8
Vérifier le coût du lissage à partir de X% du processus itératif	80 %
Vérifier le coût du lissage pour les X% meilleures particules	25 %

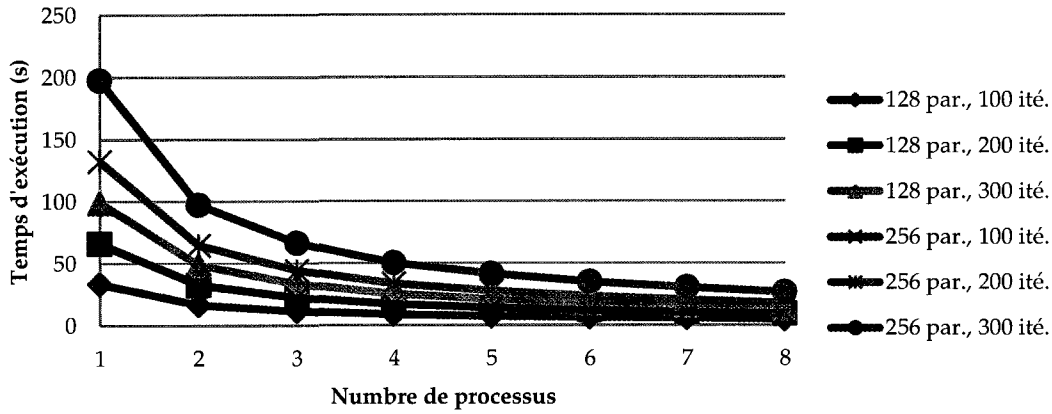


Figure 61. Temps d'exécution de l'OEP parallèle local avec migrations pour différentes tailles de travail en fonction du nombre de processus utilisés

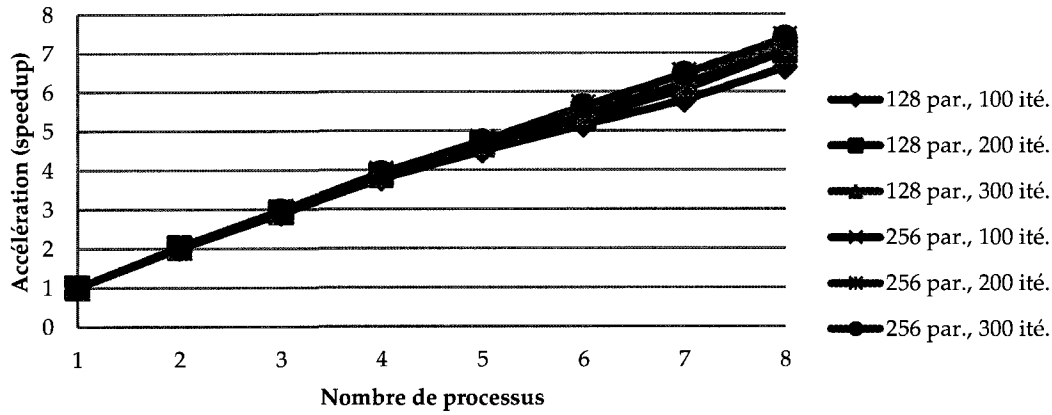


Figure 62. Accélération de l'OEP parallèle local avec migrations pour différentes tailles de travail en fonction du nombre de processus utilisés

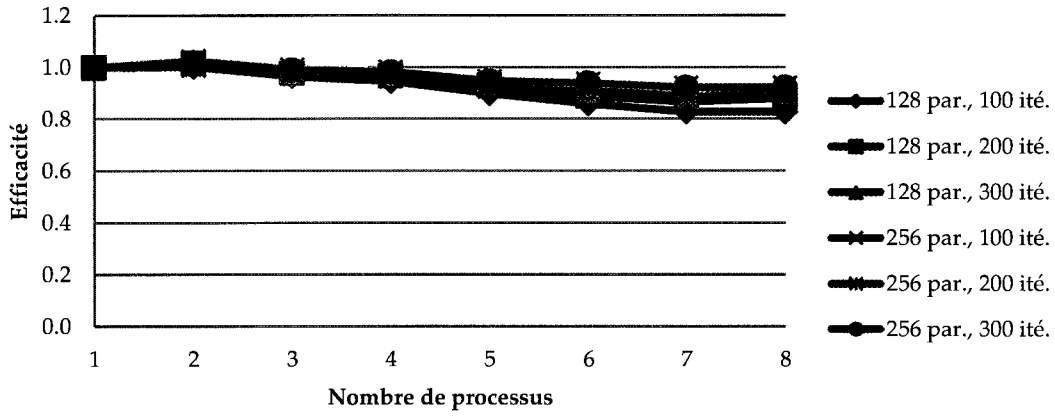


Figure 63. Efficacité de l'OEP parallèle local avec migrations pour différentes tailles de travail en fonction du nombre de processus utilisés

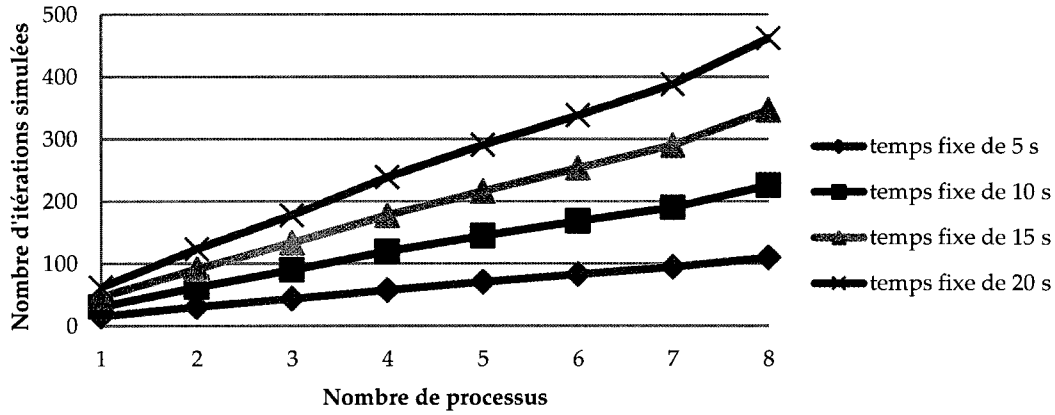


Figure 64. Nombre d'itérations exécutées par l'OEP parallèle local avec migrations pour différents temps fixes en fonction du nombre de processus utilisés

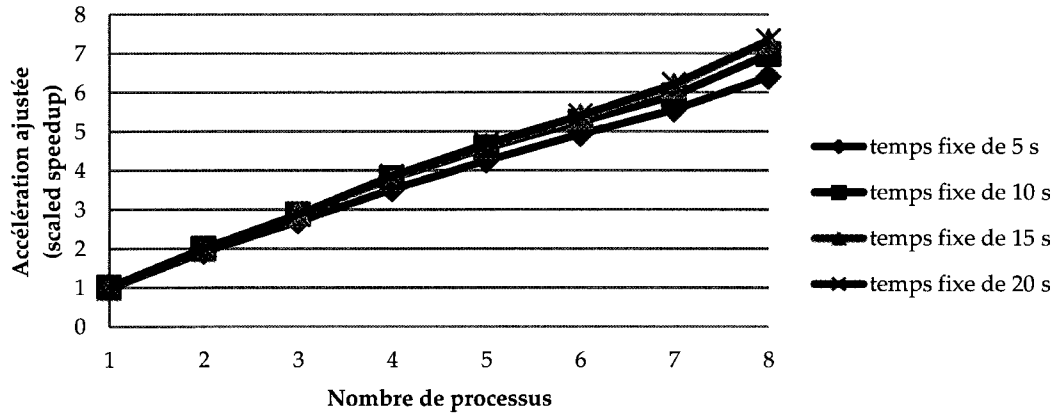


Figure 65. Accélération ajustée de l'OEP parallèle local avec migrations pour différents temps fixes en fonction du nombre de processus utilisés

7.5 Application en temps réel

Dans ce mémoire, nous avons expliqué la nécessité d'utiliser des algorithmes d'optimisation non déterministes afin d'aborder la complexité de la planification de trajectoires pour les drones. Un algorithme non déterministe ne permet pas de trouver la meilleure solution à un problème, mais une très bonne solution dans un temps raisonnable. Or, dans le cas d'un système temps réel, ce temps raisonnable est peut-être trop long. Dans ce chapitre, nous avons démontré que le paradigme de programmation parallèle « Instruction simple, données multiples » peut être utilisé afin d'exploiter pleinement la technologie multicœur des processeurs actuels et réduire le temps d'exécution par un facteur de 7. Dans le cas de notre AG parallèle, le temps nécessaire pour la planification de trajectoires est de 3.63 s (système à 8 cœurs, terrain de 500 x 500 unités, 128 chromosomes, 100 générations, 8 points de passage) et dans le cas de notre OEP parallèle, le temps est de 5.10 s (système à 8 cœurs, terrain de 500 x 500 unités, 128 particules, 100 itérations, 8 points de passages).

Il est difficile de comparer le temps d'exécution de nos deux algorithmes avec ceux récemment publiés. La plupart des auteurs dans le domaine de la planification de trajectoires discutent de la qualité de leur approche, mais plus rarement de la performance. De plus, il faut que les auteurs utilisent une fonction de coût et un terrain d'une complexité similaire à la nôtre pour que la comparaison soit valable. Notre comparaison de performance est donc restreinte à l'implémentation de Bélanger [15] basé sur l'AG et à celle d'Eslam Pour [16] basée sur l'OEP. Pour la comparaison, nous avons reproduit le scénario (terrain, point de départ, point d'arrivée et les zones dangereuses) utilisé par les deux auteurs et ajusté la grosseur de la population et le nombre d'itérations afin d'obtenir une solution qualitativement comparable. Le scénario utilisé est illustré à la figure 66 et la comparaison des temps d'exécution est présentée au tableau 10.

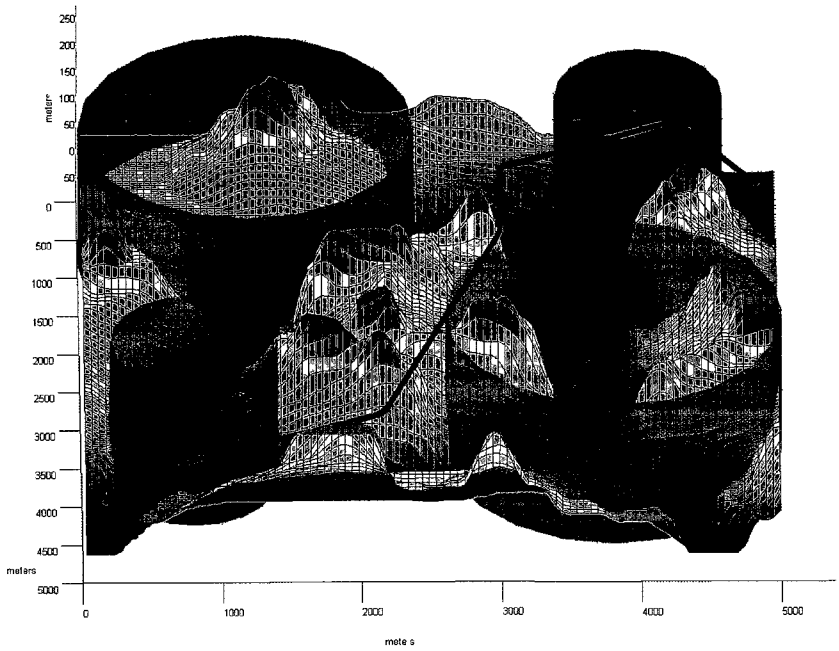


Figure 66. Scénario utilisé pour la comparaison de performance de notre implémentation de l'AG et l'OEP à celles d'autres auteurs.

Tableau 10. Comparaison de la performance de nos implémentations avec celles d'autres auteurs

Planification de trajectoires basée sur l'AG	
Temps d'execution de notre implementation	3 63 s
Temps d'execution de l'implementation de Belanger [15]	53 68 s
Gain de performance apporte par notre implementation	1 480 %
Planification de trajectoires basée sur l'OEP	
Temps d'execution de notre implementation	5 10 s
Temps d'execution de l'implementation de Eslam Pour [16]	75 s
Gain de performance apporte par notre implementation	1 470 %

Lors de la comparaison ci-dessus, tout comme les deux autres auteurs, nous avons spécialement choisi la grosseur de la population et le nombre d'itérations nécessaire afin d'obtenir un bon résultat pour le scénario utilisé tout en minimisant le calcul requis. Or, pour un scénario plus complexe, plus réaliste, tel que ceux utilisés au chapitre suivant, un nombre plus grand d'individus et d'itérations est nécessaire. Par expérimentation, nous avons déterminé qu'un temps d'exécution fixe de 10 s est suffisant et permet de générer des trajectoires faisables et de

très bonne qualité pour des scénarios réels très complexes. Or, est-ce qu'un temps d'exécution de 10 s est considéré un calcul en temps réel?

La réponse est relative à l'application. En fait, un système temps réel est un système dont l'exécution doit respecter une ou plusieurs contraintes temporelles [60]. Ces contraintes peuvent être strictes ou non, courtes ou plus longues. Incorrectement, on associe souvent un système à réponse rapide à un système temps réel. L'exécution du système n'a pas besoin d'être rapide, mais doit respecter les contraintes temporelles. Dans le cas de la planification de trajectoires, cette contrainte pourrait être que l'algorithme de planification de trajectoires doit générer une nouvelle trajectoire dans un temps plus petit ou égale à 10 s. A priori, ce temps de réaction semble trop long. Or, on peut se baser sur la vitesse de croisière du drone pour déterminer la distance parcourue en 10 s. Par exemple, prenons le drone du tableau 11 dont la vitesse est la plus grande : le Global Hawk avec une vitesse de croisière de 635 $km \cdot h^{-1}$. Les trajectoires générées doivent être d'une longueur supérieure ou égale à 1 764 m pour que le module de planification de trajectoires ait le temps de générer la trajectoire suivante avant que le drone ait parcouru la trajectoire courante. Comme présentés au chapitre suivant, nos algorithmes de planification de trajectoires génèrent des routes de 5 à 60 km sans difficulté. Ainsi, on peut considérer qu'un temps de 10 s est suffisamment court pour permettre l'utilisation de nos algorithmes en temps réel même pour un avion rapide comme le Global Hawk. Ce sera également le cas pour des avions moins rapides, comme on peut le voir dans le tableau 11.

Tableau 11. Vitesses de croisière et distances parcourues en 10 s pour différents drones utilisés par les forces armées canadiennes ou américaines

Drones	Fabricants	Vitesses de croisière ($km \cdot h^{-1}$)	Distances parcourues en 10 secondes (m)
Predator [61]	General Atomic	130	361
Global Hawk [61]	Northrop Grumman	635	1 764
Meveric [62]	Prioria	48	133
ScanEagle [63]	Boeing, Insitu	111	308
Heron [64]	Malat (division de Israel Aerospace Industries)	207	575
Silver Fox [65]	Alix, BAE Systems	83	230

Chapitre 8

Comparaison des deux algorithmes d'optimisation

Les résultats d'une étude de plus de 1 400 publications scientifiques dans le domaine de planification de trajectoires sont présentés dans [19]. L'étude souligne que depuis 1990, environ 35 % des auteurs dans le domaine utilisent une approche basée sur l'AG, ce qui en fait l'approche la plus répandue. L'article mentionne aussi que les solutions basées sur l'OEP sont de plus en plus fréquentes. Ces statistiques sont très intéressantes, mais ne peuvent être utilisées pour identifier une approche préférable. Dans son mémoire de maîtrise [15], Bélanger compare une implémentation basée sur l'AG avec une basée sur l'optimisation par colonie de fourmis (OCF). Bien que ses résultats soient intéressants, sa comparaison n'est pas assez rigoureuse pour être vraiment concluante. Il n'utilise que deux terrains fictifs, une résolution différente pour chacun des algorithmes (500 x 500 pour l'AG et 20 x 20 pour l'OCF) et une fonction de coût différente. De plus, ses conclusions sont basées sur uniquement 7 trajectoires générées par chacun des algorithmes. Un second auteur, Eslam Pour présente dans son mémoire de maîtrise [16] une comparaison entre l'AG et l'OEP. Tout comme Bélanger, sa comparaison est qualitative, basée sur les mêmes deux terrains fictifs et utilise aussi des fonctions de coût différentes. En fait, plusieurs auteurs utilisent une méthode semblable et basent leurs conclusions sur une comparaison visuelle de quelques trajectoires générées par différents moteurs d'optimisation sur un nombre limité de terrains.

Dans ce chapitre, nous abordons cette lacune et présentons une comparaison quantitative et statistiquement significative des solutions générées par nos algorithmes de planification de trajectoires pour les drones basée sur l'AG et l'OEP. Nous utilisons 40 scénarios sur 8 terrains différents (les deux précédemment utilisés par Bélanger [15] et Eslam Pour [16] et six terrains réels provenant de la base de données canadienne de cartes d'élévation digitales GeoBase [5]). Nos deux algorithmes de planification de trajectoires utilisent la même fonction de coût et un temps de calcul fixe de 10 s permettant ainsi une comparaison valide.

8.1 Optimisation des paramètres

Avant de comparer la qualité des solutions produites par chacun des algorithmes, il est important d'ajuster leurs paramètres. Il serait incorrect de conclure qu'une approche est meilleure que l'autre si l'une était bien paramétrée et l'autre non. Tel qu'expliqué dans [17] et [66], il existe quelques règles de base pour l'ajustement des paramètres, mais aucune technique infaillible. Par exemple, il est proposé d'utiliser une heuristique telle que l'AG pour paramétrer un autre AG (ce qui serait beaucoup trop long dans notre cas) [17]. Ces deux ouvrages recommandent aussi un ajustement manuel par essais et erreurs spécifique à l'application. Dans notre cas, nous utilisons les paramètres de bases suggérés dans ces deux livres et déterminons expérimentalement que la taille de la population (ou de l'essaim) et le nombre de générations (ou d'itérations) sont les deux paramètres qui influencent le plus la qualité des trajectoires produites par l'AG et l'OEP. La taille de la population influe surtout sur l'exploration de l'espace de recherche et le nombre de générations, sur le raffinement de la solution finale. Lorsque l'espace de recherche a un grand nombre de dimensions (c.-à-d. lorsqu'on désire une trajectoire finale avec plusieurs points de passage) où lorsque l'espace est compliqué (p. ex. lorsqu'on utilise un terrain montagneux), il est nécessaire d'utiliser une grande population pour que le moteur d'optimisation puisse trouver une trajectoire faisable. Après avoir choisi la taille de la population, il est possible d'ajuster le nombre de générations pour faire varier la qualité de la trajectoire finale. Évidemment, une très grande population couplée avec un très grand nombre de générations produit les meilleures solutions. Cependant, puisque nous voulons effectuer la planification de trajectoires en temps réel, nous devons sélectionner une taille de population et un nombre de générations qui produisent de très bonnes solutions tout en minimisant le temps de calcul.

Afin de respecter la contrainte temporelle de la planification de trajectoires en temps réel, nous avons établi au chapitre précédent un temps de calcul fixe de 10 s. Or, puisque la complexité de l'AG est définit par $O(taille\_pop * nombre\_gen)$ et celle de l'OEP par $O(taille\_essaim * nombre\_ité)$, seule la taille de la population doit être ajustée. Pour déterminer la valeur optimale, nous exécutons les deux algorithmes avec différentes tailles de populations et calculons 20 trajectoires pour chacun des 40 scénarios. Nous affichons ensuite les distributions des coûts des trajectoires générées à la figure 67 (en utilisant la fonction « boxplot » de MATLAB [57]). Dans le cas de l'AG, il semble qu'une population de 128 chromosomes soit idéale. Si l'on utilise une population plus grande, le coût médian augmente d'abord légèrement

pour des populations de 192 et 256 puis significativement pour une population de 320. Dans le cas de l'OEP, la qualité des solutions est optimale pour un essaim de 192 particules et ne s'améliore pas avec un plus grand essaim. Pour notre comparaison, nous utilisons donc une population de 128 chromosomes pour l'AG et de 192 particules pour l'OEP.

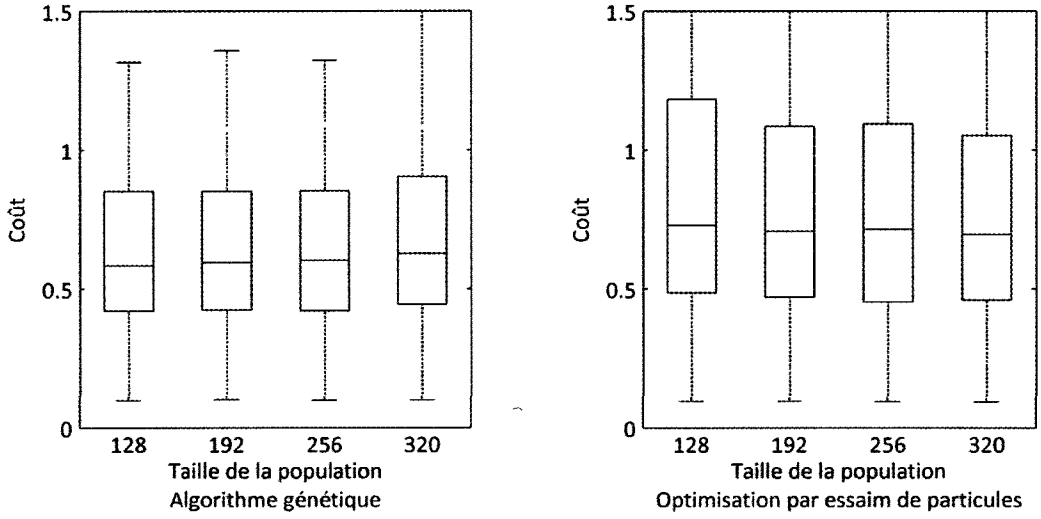


Figure 67. Distribution des coûts des trajectoires calculées par l'AG et l'OEP en fonction de la taille de la population (temps de calcul fixe de 10 secondes, 8 processus, 40 scénarios sur 8 terrains différents, 20 trajectoires générées pour chaque scénario)

8.2 40 scénarios différents

Nous présentons dans cette section les huit terrains utilisés pour la comparaison de l'AG et l'OEP. Les cartes d'élévation digitales (CED) 1 et 2 représentent des terrains fictifs et proviennent de [16] et [15]. Les CED 3 à 8 représentent des terrains réels à l'échelle de 1:50 000 suivant la projection NAD63 et proviennent de la base de données GeoBase [5]. Pour faciliter la visualisation, nous superposons à chaque CED une photographie satellite suivant la projection WGS84 et provenant du site internet Google Maps [67]. Puisque la projection NAD63 est plus ancienne et que la projection WGS84 est beaucoup plus répandue (utilisé par les systèmes de positionnement global portables), nous avons re-projeté les CED à l'aide du logiciel Global Mapper [68] avant de les superposer aux images satellites WGS84. Contrairement aux terrains simples utilisés dans plusieurs articles récents [69], [13], [15], [16], [70], [71], [72], [73], [74], [22], [75] et [76], ces six terrains ont été choisis avec soin pour leur grande complexité afin que nous puissions démontrer la performance réelle de nos algorithmes. Dans le but d'augmenter le

nombre de tests sur lesquels nous basons notre comparaison, nous développons cinq scénarios pour chacun des terrains. Un scénario consiste en un agencement unique du point de départ, du point d'arrivée et des zones dangereuses. Les cartes 2D et 3D des 8 terrains sont présentées aux figure 68 à figure 83. Les trajectoires montrées sur ces figures ont été générées par l'AG. L'OEP produit des solutions visuellement aussi bonnes. Une description de chacun des terrains se trouve ci-dessous.

La carte 1 représente un canyon sur 5 km par 5 km. Ce terrain est fictif et a précédemment été utilisé par les auteurs de [16] et [15]. Le cours d'eau se situe à une altitude de 0 m ASL (au-dessus du niveau de la mer) et le sommet le plus haut, à 255 m ASL. Lorsque l'on place le point de départ au coin inférieur gauche et le point d'arrivée au coin supérieur droits, nos algorithmes génèrent des trajectoires faisables qui suivent le cours d'eau et passent d'un côté ou l'autre de l'île minimisant ainsi l'altitude moyenne et la longueur du vol. On peut aussi forcer la trajectoire à passer au-dessus de l'île en ajoutant des zones dangereuses. Comme on le remarque à la figure 68, une trajectoire qui traverse l'île inclut certaines courbes afin de minimiser l'altitude moyenne et respecter la puissance maximale du drone.

La carte 2 est elle aussi fictive et provient de [16] et [15]. Elle représente une rivière sur 5 km par 5 km. L'altitude du terrain varie de 0 à 255 m ASL. Lorsque les points de départ et d'arrivées sont placés aux extrémités de la rivière, les trajectoires générées suivent l'eau minimisant l'altitude et la longueur du vol comme on le voit à la figure 70. Si l'on place les points de départ dans l'autre diagonale de la carte, il est intéressant de remarquer que les trajectoires descendent, traversent la rivière près de sa surface et remontent ensuite sur l'autre rive minimisant toujours l'altitude moyenne et la longueur du trajet.

La carte 3 représente la région de St-Jean de Terre-Neuve. L'altitude varie de 0 à 246 m ASL et sa superficie est de 10 km par 10 km. À la figure 72, nous plaçons le point de départ dans le port de St-Jean et le point d'arrivée à l'est de Cap Spear. Nous bloquons l'entrée du port à l'aide de deux zones dangereuses situées sur les anciennes fortifications militaires et disposons cinq zones dangereuses dans l'océan représentant les radars de navires ennemis. On remarque que la trajectoire générée évite l'entrée du port et passe par une seconde ouverture plus au nord minimisant l'altitude. Le trajet survole ensuite la surface de l'océan sans pénétrer aucune zone dangereuse.

La carte 4 couvre une région de 192 km^2 aux alentours du Lac Louise, Alberta. L'altitude du terrain varie de 1 515 à 3 491 m ASL. À la figure 74, nous avons placé le point de départ dans une vallée au coin supérieur droit et le point d'arrivée, dans une vallée au coin inférieur gauche. Nos algorithmes de planifications génèrent habituellement des trajectoires qui suivent la vallée minimisant l'altitude moyenne du trajet. Cependant, dans cet exemple, nous forçons la trajectoire à traverser la montagne en bloquant la vallée à l'aide de zones dangereuses. Puisque le drone utilisé dans le calcul ne possède pas la puissance nécessaire pour monter en ligne droite vers le sommet de la montagne, nos algorithmes génèrent des hélices verticales afin de minimiser le taux d'ascension requis et produire une trajectoire faisable (il est important de noter que le taux d'ascension maximal du drone diminue à mesure qu'il gagne de l'altitude).

Le cinquième terrain a une superficie de 348 km^2 et représente en partie le Lac Kinbasket en Colombie-Britannique. L'altitude du sol varie de 750 à 2 746 m ASL. Cet exemple est plus complexe que les précédents puisque les points de départ et d'arrivée sont séparés par plusieurs vallées et montagnes. On remarque à la figure 76 que la trajectoire générée sillonne les vallées pour se rendre à l'objectif en minimisant l'altitude et la distance parcourue.

La carte 6 couvre une région de 269 km^2 autour du Glacier Columbia en Alberta. L'altitude du terrain varie de 1 621 à 3 486 m, soit la plus haute parmi tous nos exemples. À la figure 78, le point de départ se situe au coin supérieur gauche à une altitude de 2 084 m et le point d'arrivé, au coin inférieur droit à une altitude de 2 350 m. Si l'on télécommandait manuellement le drone, on serait tenté de le faire descendre dans la vallée (tout comme à la figure 78) pour contourner le glacier de gauche et ensuite suivre la coulée de glace et rejoindre le plateau (à environ 2 800 m ASL) au bas à gauche de la carte d'où l'on pourrait facilement atteindre l'objectif. Cette trajectoire serait faisable par le drone utilisé dans l'exemple si ce terrain se trouvait au niveau de la mer. Or la puissance requise par cette trajectoire est beaucoup plus grande que celle disponible due à la haute altitude. Dans le cas d'un avion ordinaire, le pilote remarquerait immédiatement ce manque de puissance par le bruit du moteur, les vibrations, le manque de portance, etc. Dans le cas d'un drone télécommandé, il serait beaucoup plus difficile de remarquer ce manque de puissance. Cet exemple démontre bien la valeur de nos algorithmes de planification de trajectoires qui réussissent à trouver une trajectoire faisable en traversant l'étroit passage entre le glacier de droite et la limite de la carte. La trajectoire générée minimise la distance parcourue, l'altitude moyenne et respecte la puissance maximale disponible du drone.

La septième carte utilisée (figure 80) couvre une superficie de 1 360 m² et une dénivellation de 1 290 à 3 079 m dans la région de Banff, Alberta. Cet exemple contient de nombreuses montagnes et vallées ce qui représente un espace de recherche complexe et discontinu. Nos algorithmes d'optimisation réussissent toutefois à générer une très bonne trajectoire faisable tout en évitant les zones dangereuses.

Le terrain 8 s'étend sur 3 717 m² dans la région de Jasper, Alberta, et son altitude varie entre 1 020 et 3 308 m. Son relief est extrêmement complexe et teste la limite de nos algorithmes de planification de trajectoires. En fait, il serait difficile de trouver un terrain plus compliqué. À la figure 82, nous plaçons le point de départ à gauche et le point d'arrivée à droite. La trajectoire générée suit d'abord une vallée et gagne ensuite l'altitude nécessaire pour survoler la région montagneuse. Le détour visible à la fin du trajet n'est pas superflu, mais nécessaire pour que le drone descende au point d'arrivée sans dépasser son taux maximum de descente d'après les spécifications du drone (tableau 15 au chapitre 9).

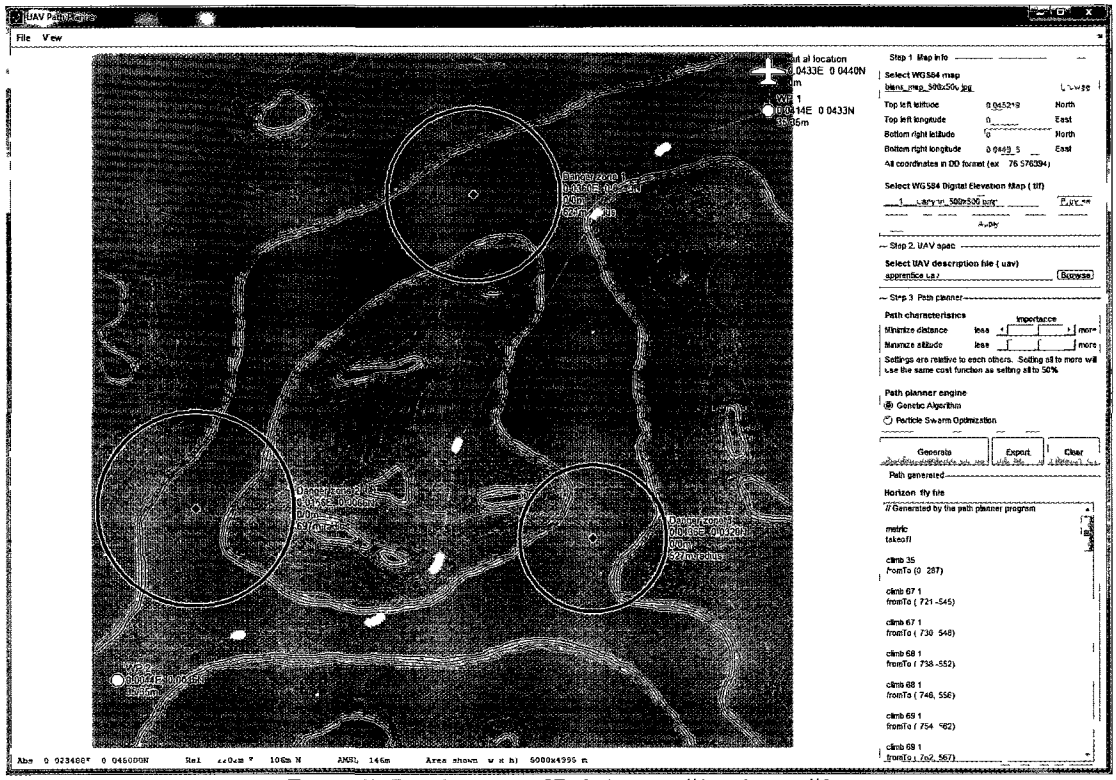


Figure 68. Représentation 2D de la carte #1, scénario #3

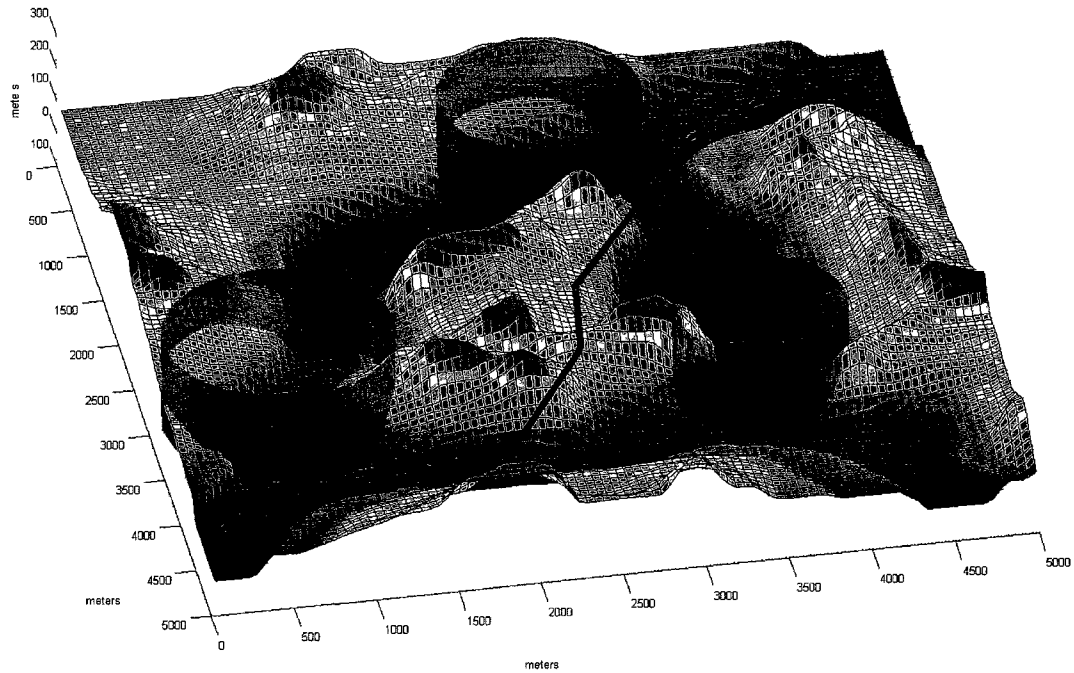


Figure 69. Représentation 3D de la carte #1, scenario #3

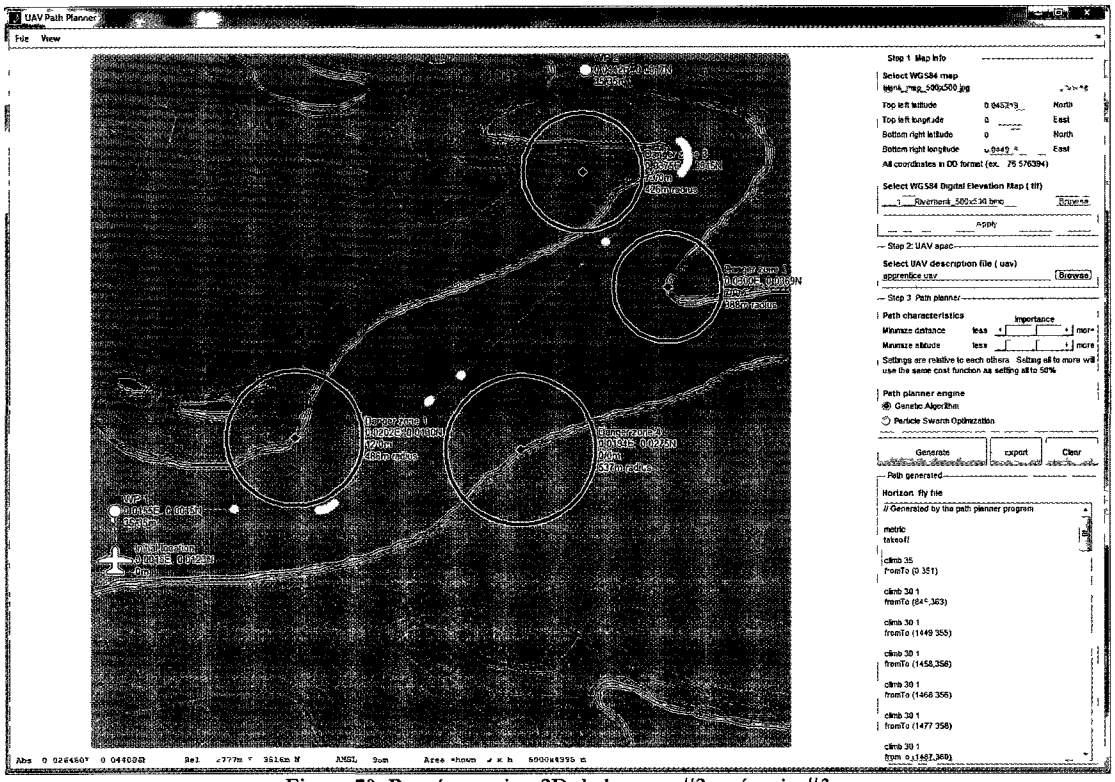


Figure 70. Représentation 2D de la carte #2, scénario #3

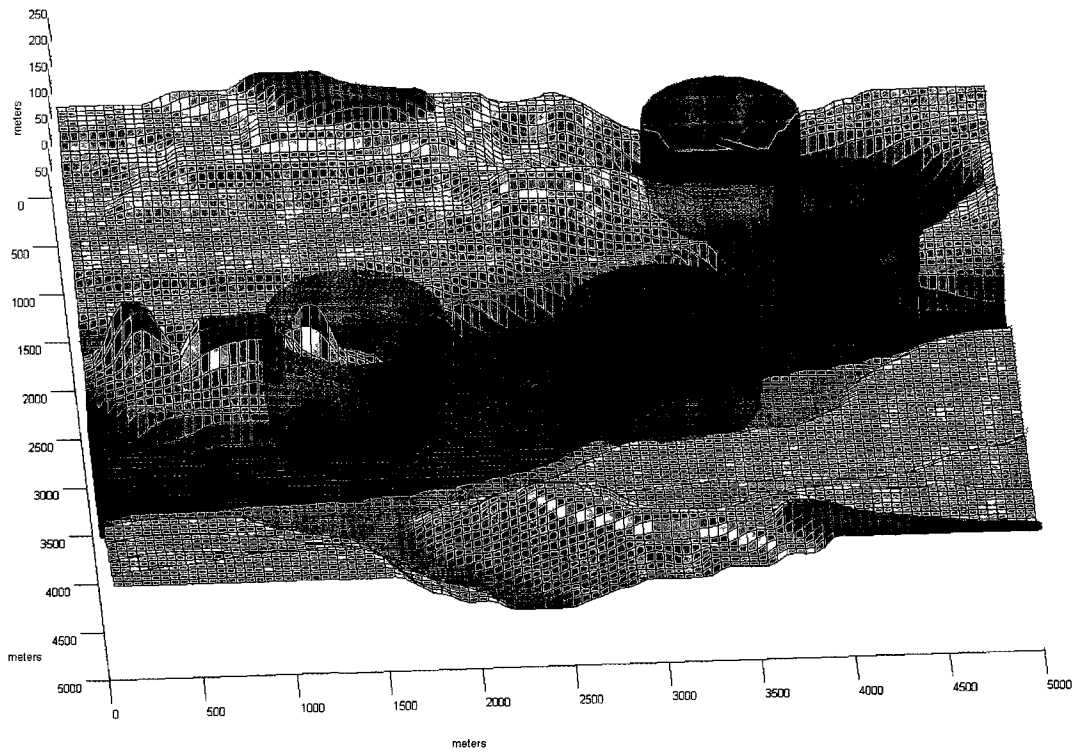


Figure 71. Représentation 3D de la carte #2, scénario #3

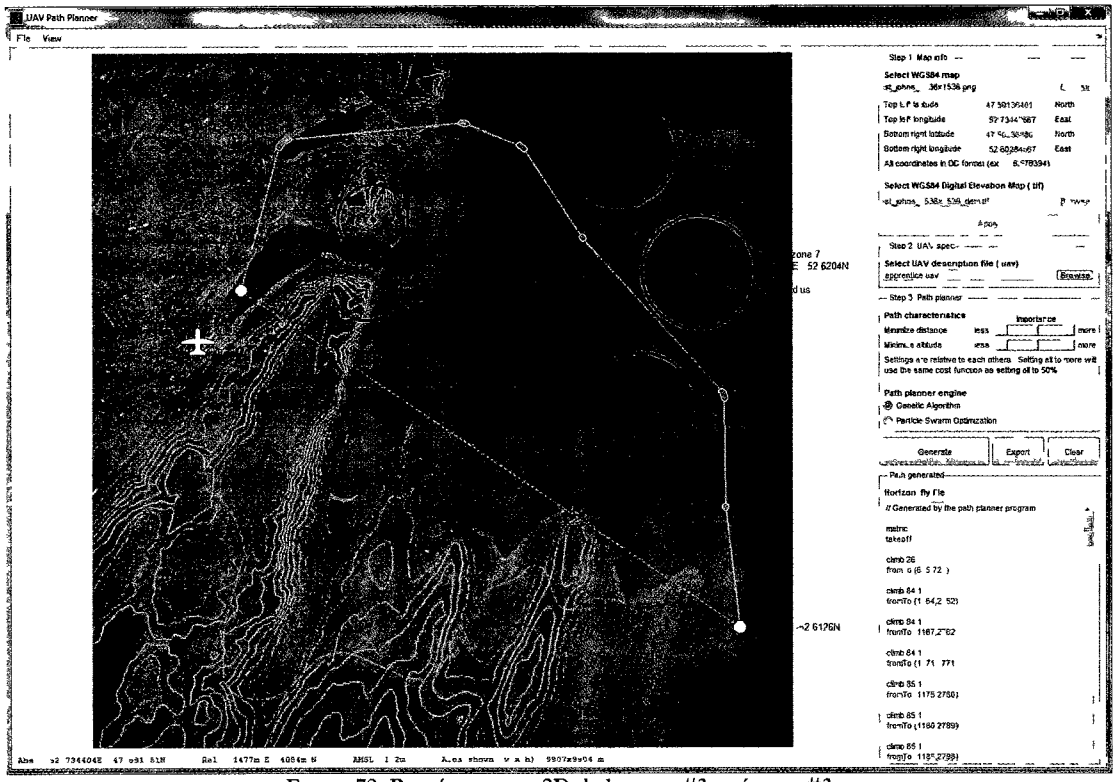


Figure 72. Représentation 2D de la carte #3, scénario #3

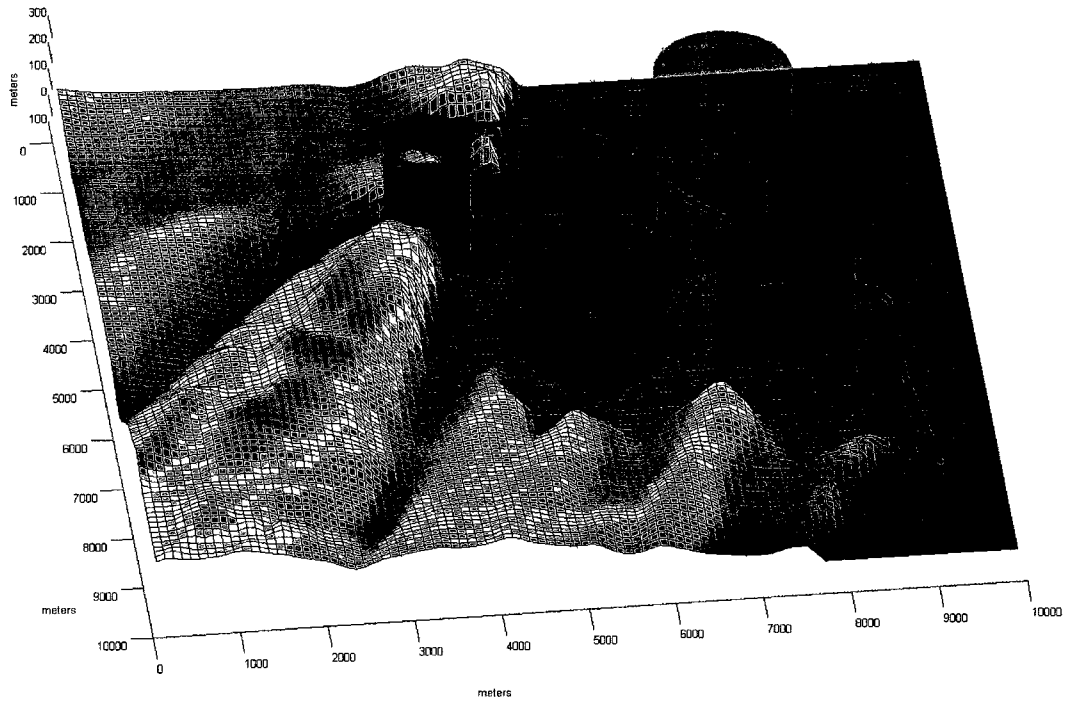


Figure 73 Representation 3D de la carte #3, scenario #3

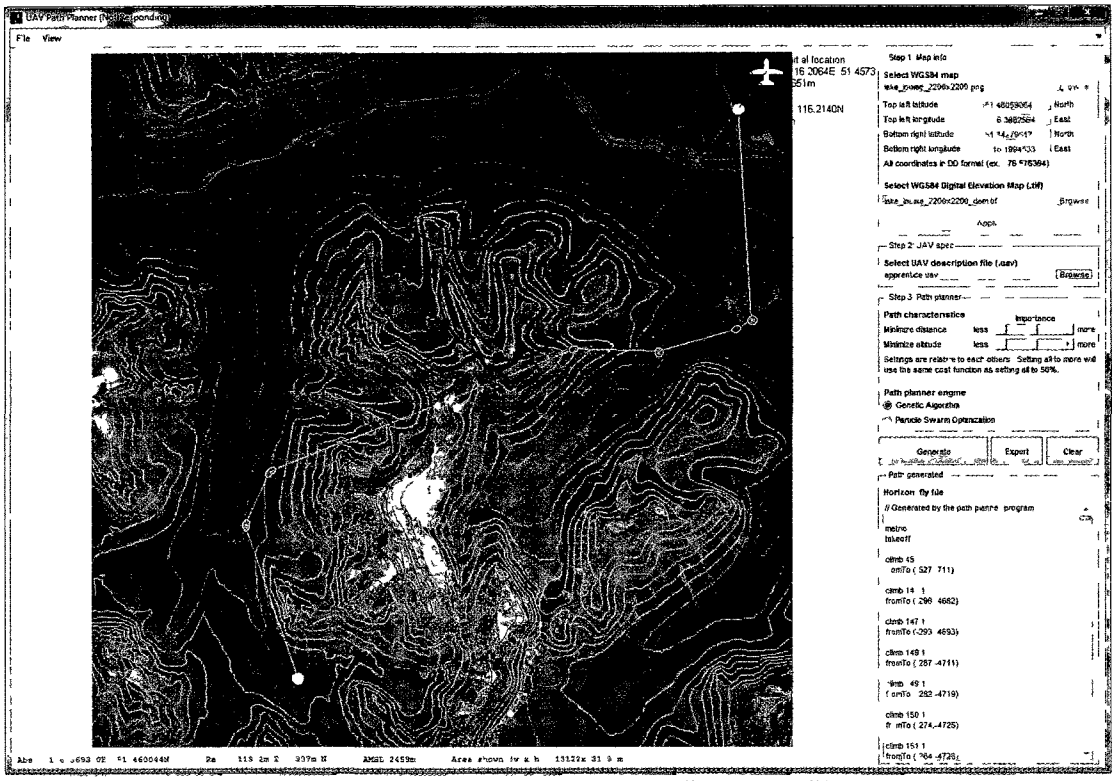


Figure 74. Représentation 2D de la carte #4, scénario #3

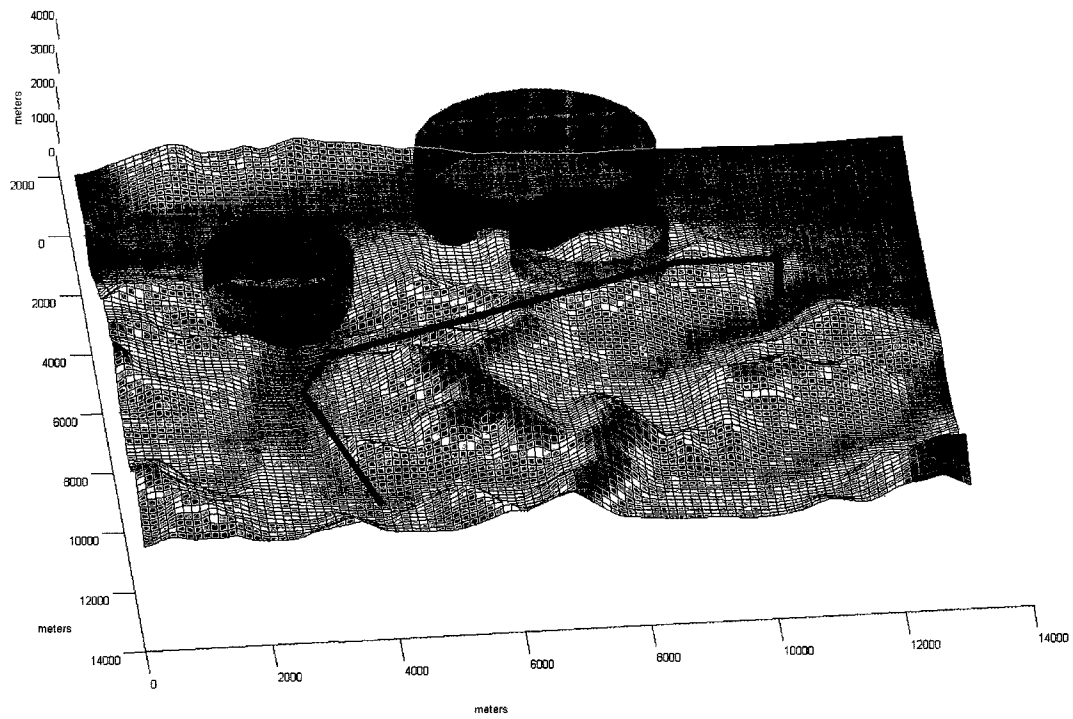


Figure 75. Representation 3D de la carte #4, scenario #3

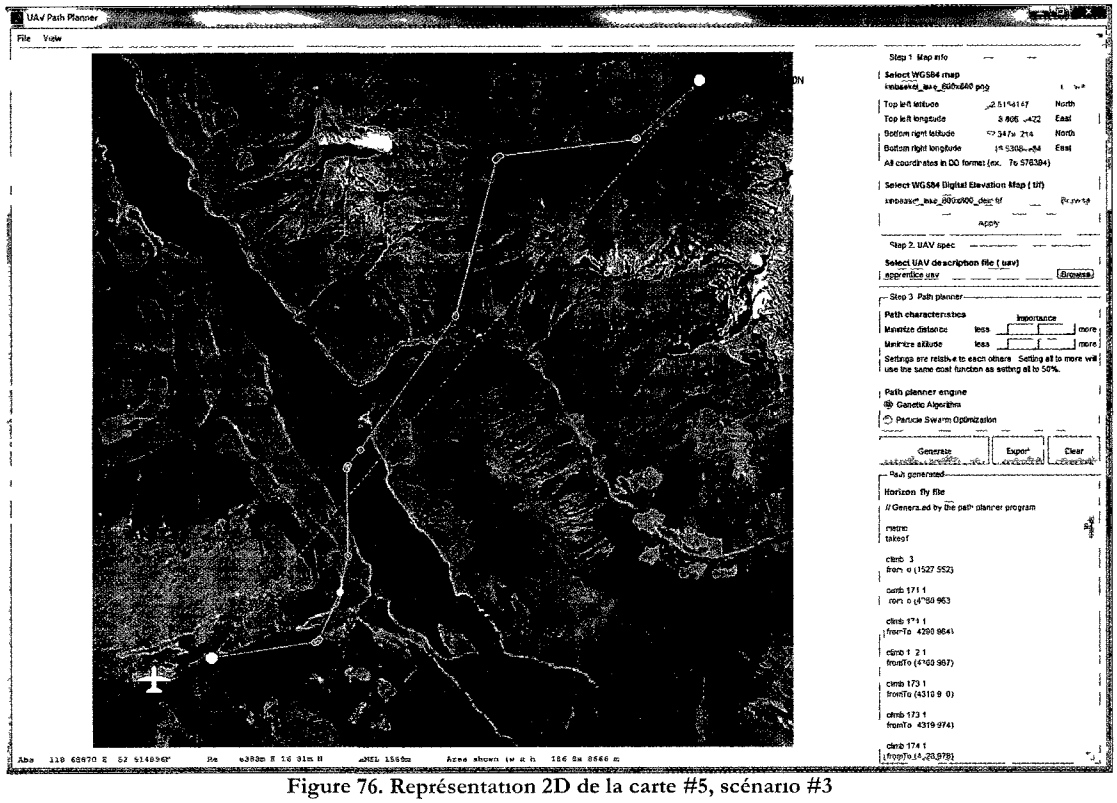


Figure 76. Représentation 2D de la carte #5, scénario #3

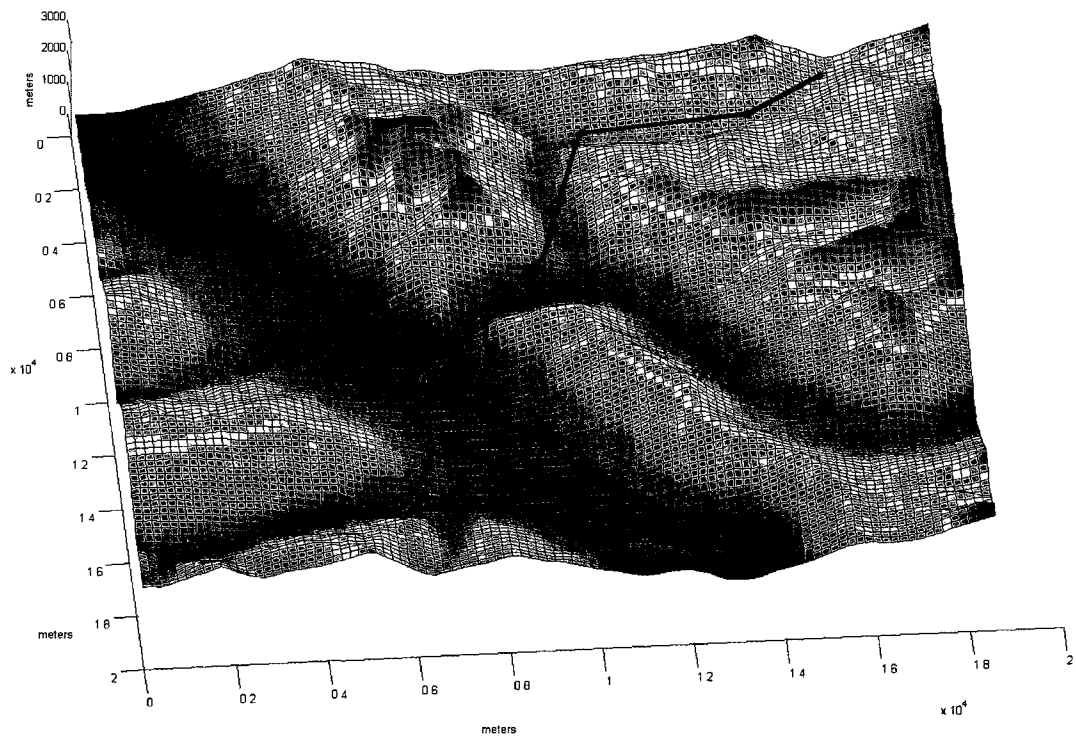


Figure 77. Representation 3D de la carte #5, scénario #3

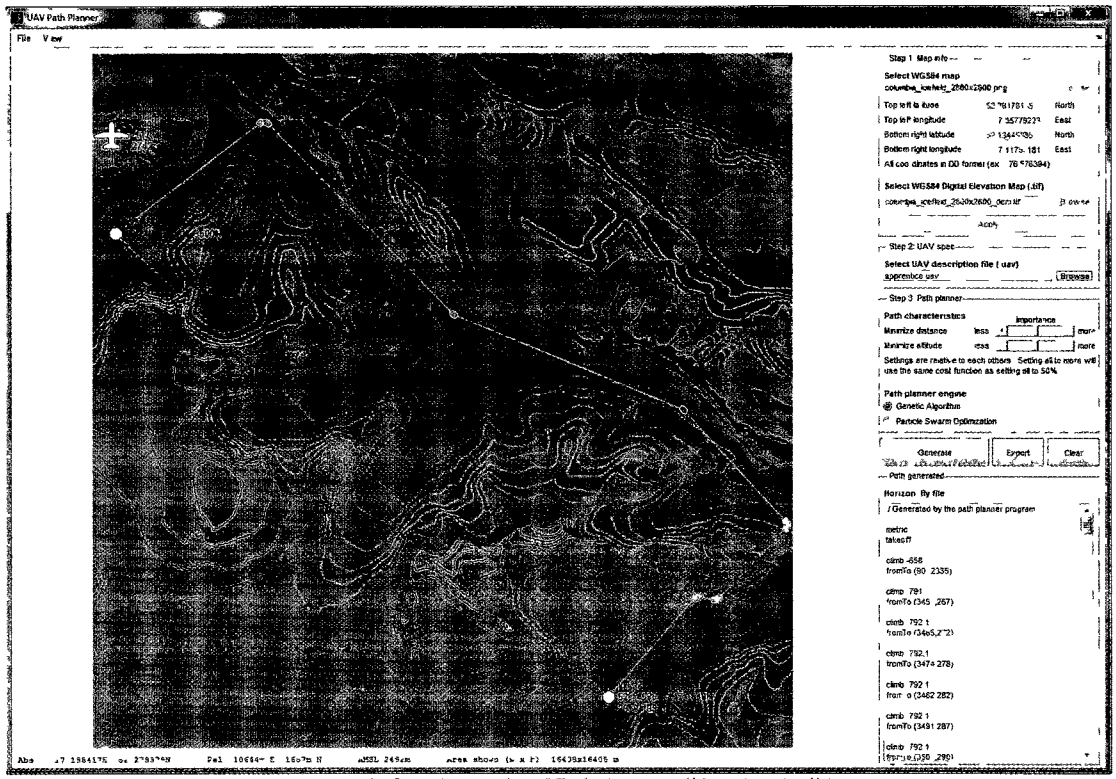


Figure 78. Représentation 2D de la carte #6, scénario #5

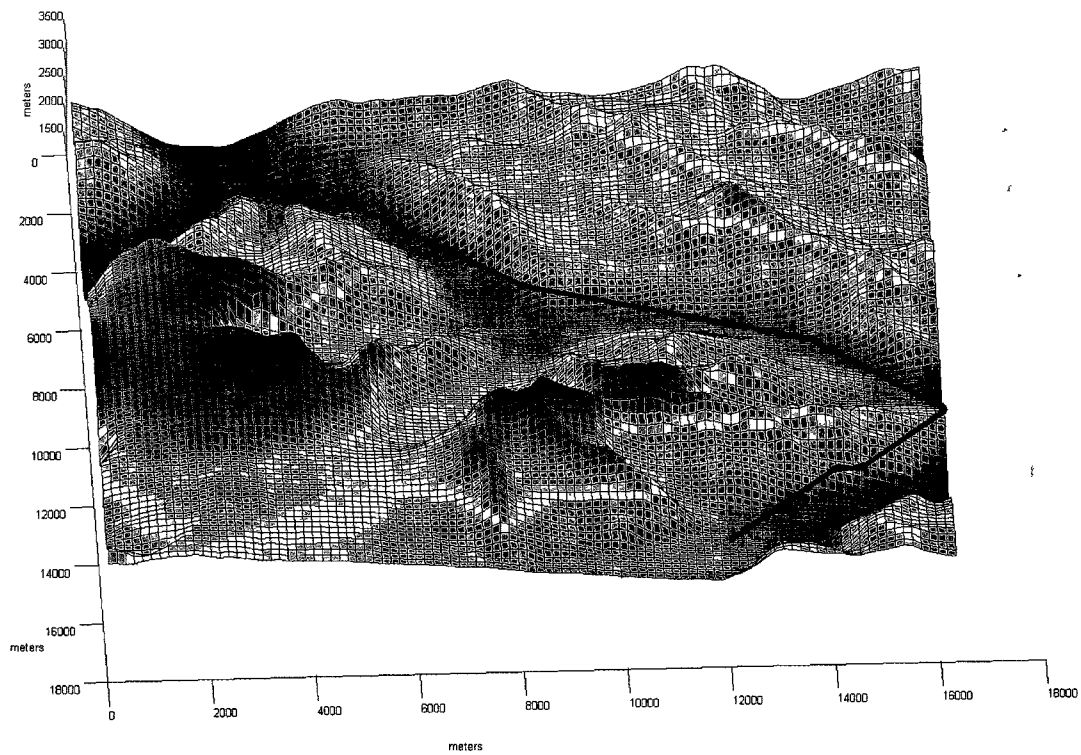


Figure 79. Représentation 3D de la carte #6, scénario #5

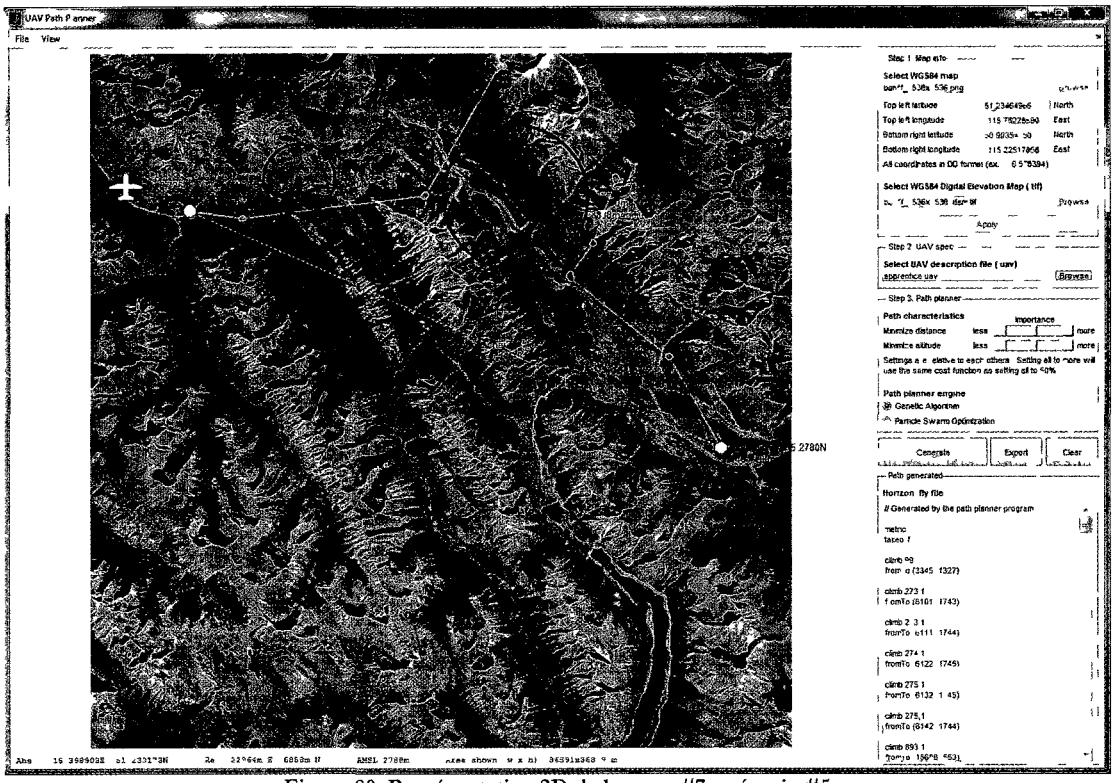


Figure 80. Représentation 2D de la carte #7, scénario #5

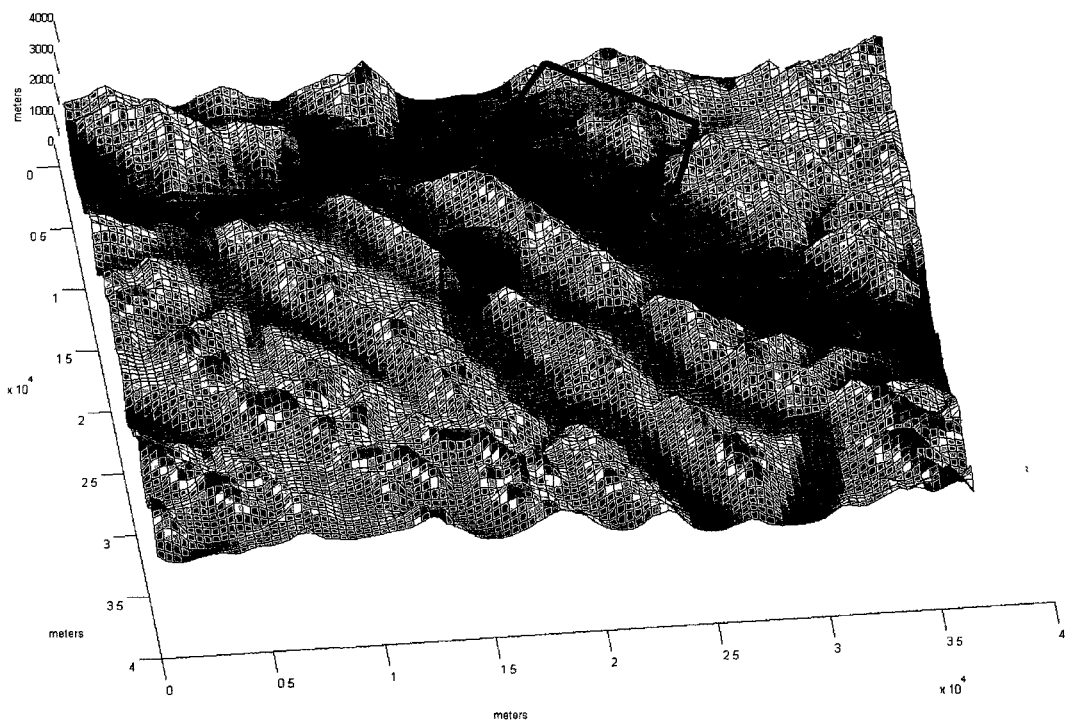


Figure 81. Représentation 3D de la carte #7, scénario #5

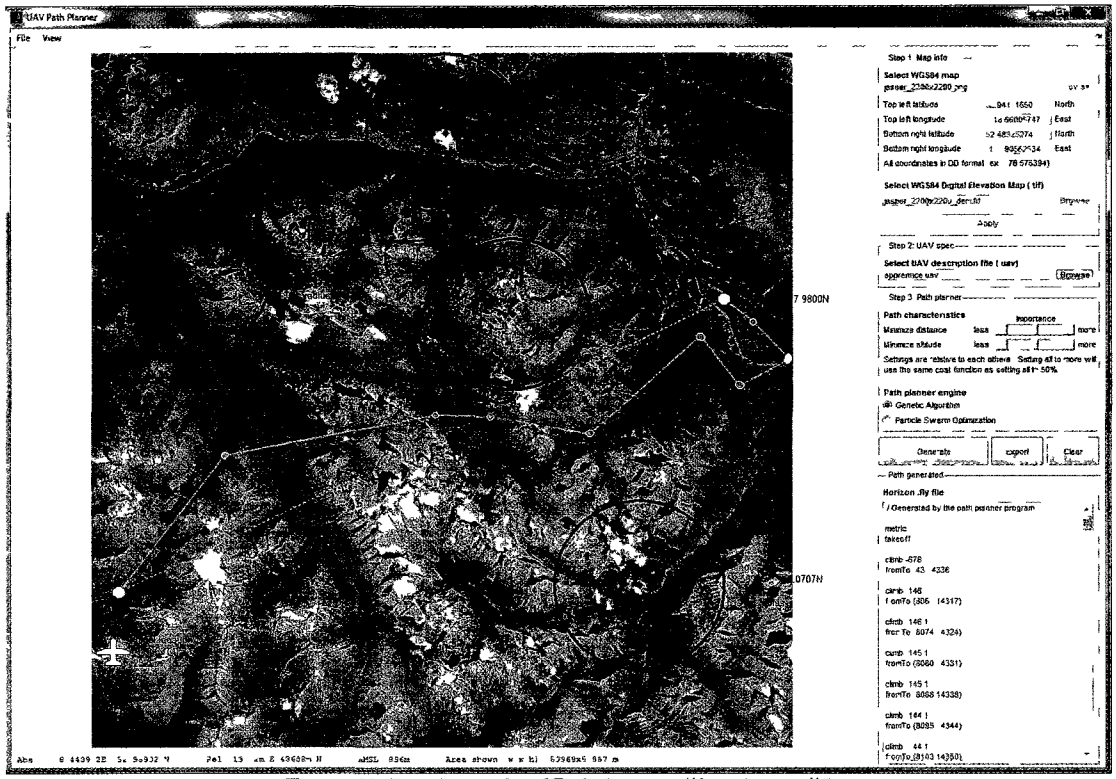


Figure 82. Représentation 2D de la carte #8, scénario #5

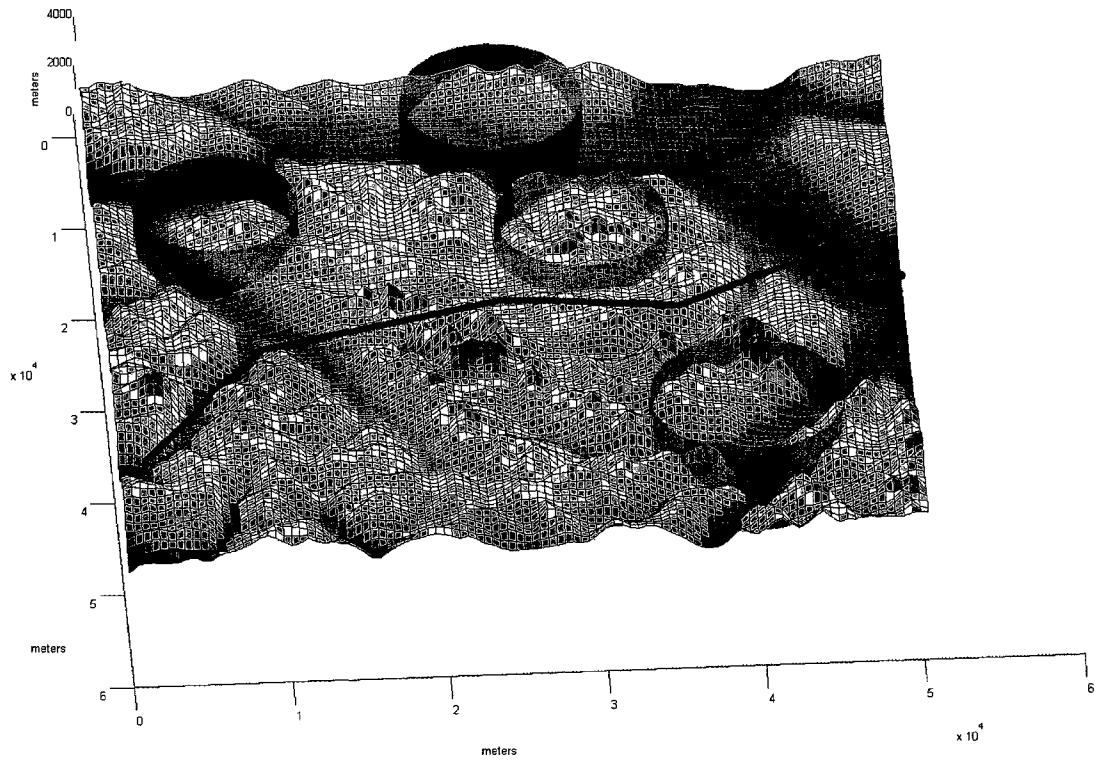


Figure 83 Représentation 3D de la carte #8, scenario #5

8.3 Présentation des résultats

Afin de comparer la qualité des trajectoires produites par notre AG et notre OEP, nous développons un programme test qui calcule 60 trajectoires pour chacun des 40 scénarios utilisant d'abord l'AG et ensuite l'OEP. Les paramètres utilisés pour chacun des algorithmes sont listés au tableau 12 et les caractéristiques du drone, au tableau 15 (voir chapitre 9). Nos deux algorithmes utilisent la même fonction de coût. Les distributions du coût des trajectoires générées sont affichées aux figure 84 et figure 85. Il est important de noter que le coût d'une trajectoire est une mesure quantitative de la qualité de la trajectoire d'après la formule présentée au chapitre 3. On ne peut pas comparer le coût de deux trajectoires pour deux scénarios différents puisque le trajet optimal a un coût différent pour chaque scénario. Prenons l'exemple d'un terrain plat sans zone dangereuse, la trajectoire optimale serait la ligne droite et aurait un coût de 0. Dans le cas d'un terrain montagneux, la ligne droite serait impossible et la trajectoire optimale serait obligatoirement plus longue et plus élevée résultant en un coût supérieur à 0. En d'autres mots, une trajectoire avec un coût de 0.5 peut-être quasi optimale pour un scénario, mais beaucoup moins bonne dans un autre scénario. Pour cette raison, nous présentons aux figure 84 et figure 85 la distribution des coûts pour chacun des scénarios individuellement en utilisant la fonction « boxplot » de MATLAB [57].

Tableau 12. Paramètres de configuration de l'AG et de l'OEP utilisés pour la comparaison

Paramètres	Valeurs	
	AG	OEP
Implémentation de l'algorithme d'optimisation	AG parallele à gros grains avec migrations	OEP parallèle avec essaims locaux avec migrations
Ordinateur multicœur utilisé	Tel que défini au tableau 6	Tel que défini au tableau 6
Outil de développement utilisé	Windows 7 entreprise, 64 bits MATLAB 2010b, 64 bits, PCT v5.0	Windows 7 entreprise , 64 bits MATLAB 2010b, 64 bits, PCT v5.0
Nombre de processus utilisés	8	8
Temps de calcul fixe	10 s	10 s
Nombre de chromosomes/particules	128	192
Nombre de générations/itérations	Déterminé par le temps de calcul	Déterminé par le temps de calcul
Nombres de migrations	5	5
Longueur des chromo./particules	8	8
Vérifier le coût du lissage à partir de X% du processus itératif	80 %	80 %
Vérifier le coût du lissage pour les X% meilleurs chromosomes/particules	25 %	25 %
Résolution du terrain	* 500 x 500 unités	* 500 x 500 unités
* Nous utilisons ici une résolution de terrain de 500 x 500 unités afin d'être consistent avec les résolutions utilisées dans [15][16][14]. Cependant, nos implémentations de l'AG et de l'OEP utilisent des nombres réels et ne sont pas limitées par la discrétisation du terrain.		

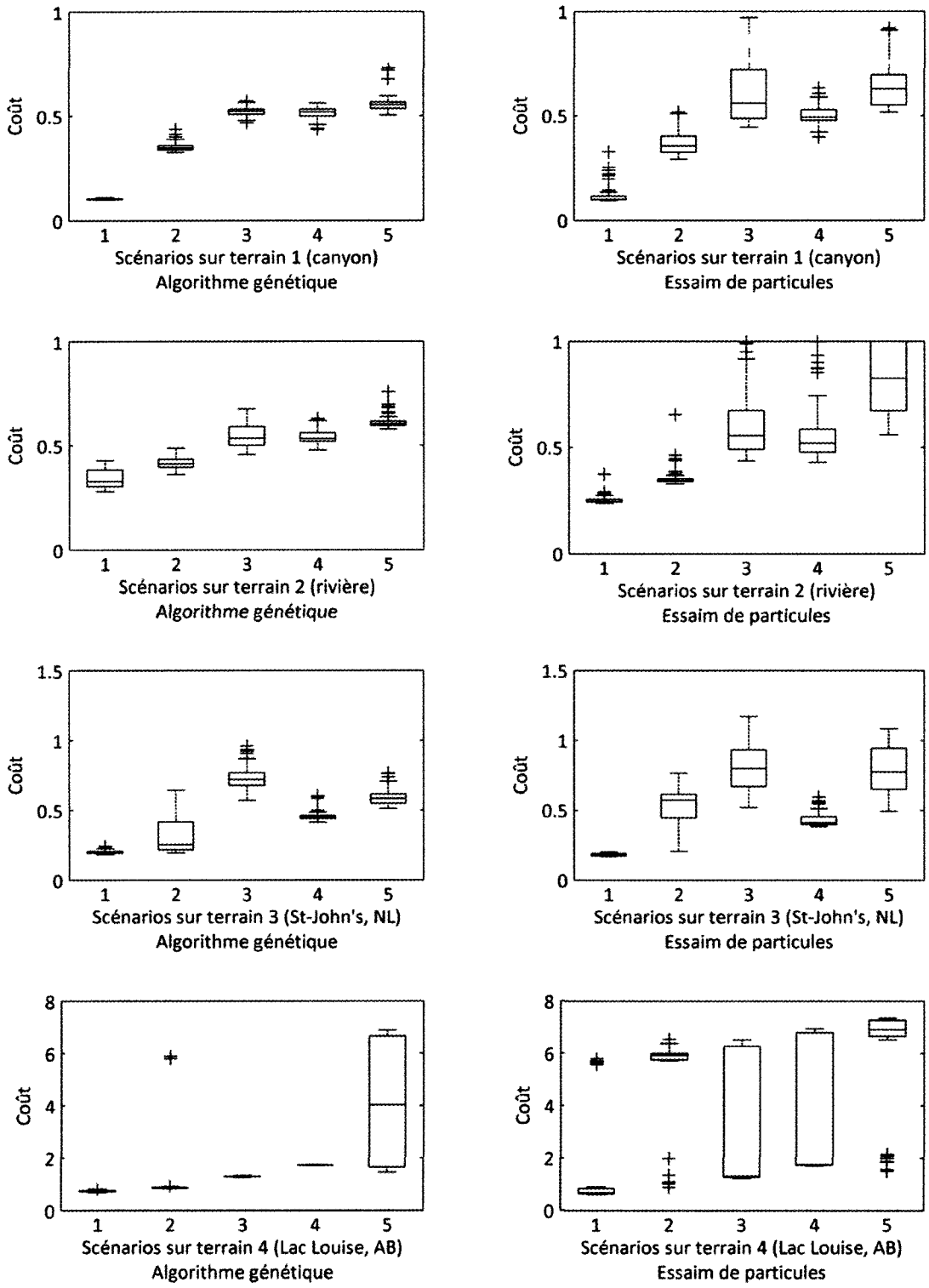


Figure 84. Coûts des trajectoires calculées par l'AG et l'OEP pour les 5 scénarios sur les terrains 1 à 4 (temps de calcul fixe de 10 secondes, 8 processus, chaque trajectoire a été calculée 60 fois)

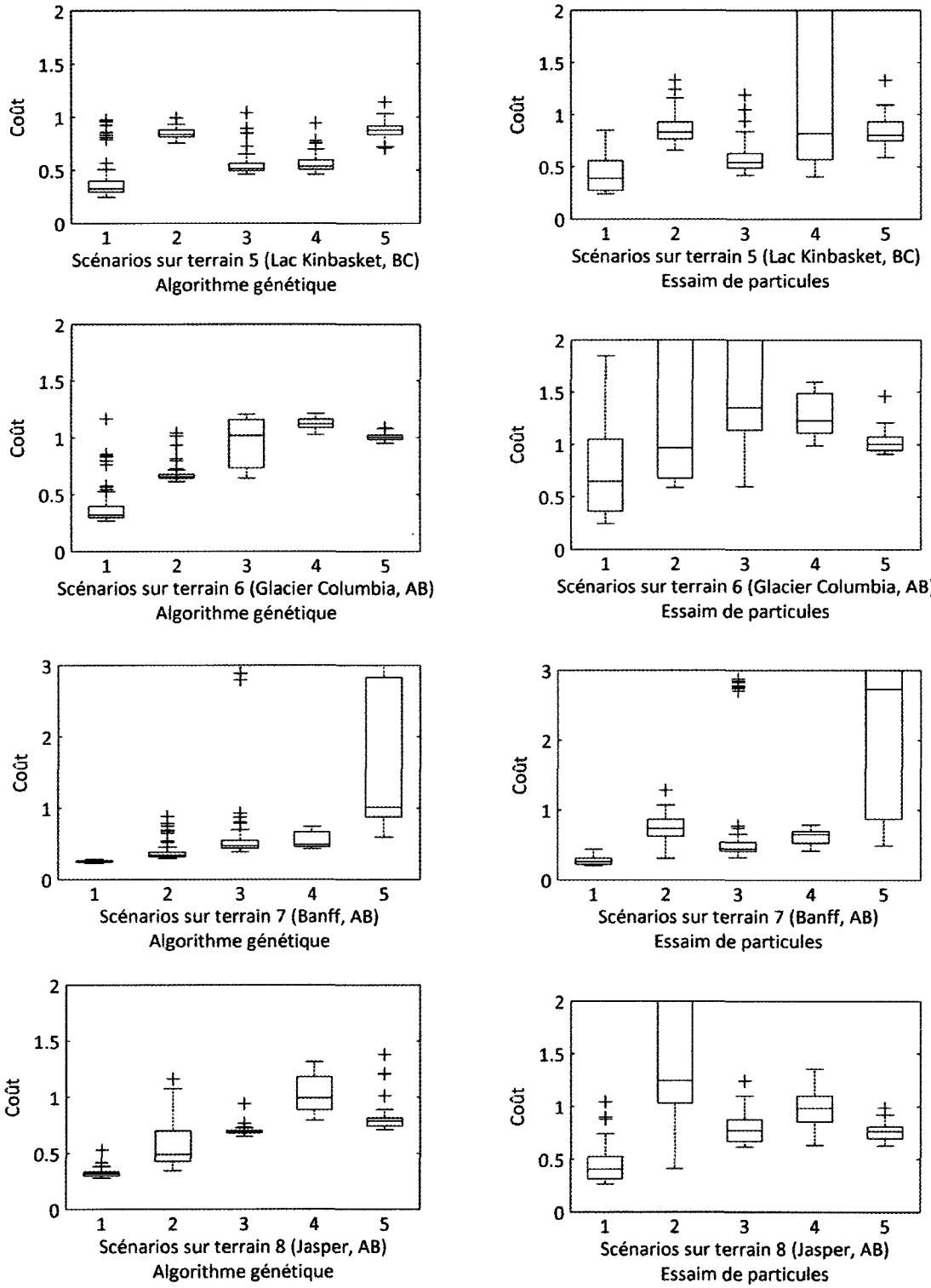


Figure 85. Coûts des trajectoires calculées par l'AG et l'OEP pour les 5 scénarios sur les terrains 5 à 8 (temps de calcul fixe de 10 secondes, 8 processus, chaque trajectoire a été calculée 60 fois)

Les graphiques présentés aux figure 84 et figure 85 permettent de comparer les distributions des coûts des trajectoires produites par les deux algorithmes pour chacun des 40 scénarios, un à la fois. On remarque que, en général, la dispersion des coûts est moins grande pour l'AG que pour l'OEP ce qui semble signifier une meilleure fiabilité de la part de l'AG. On remarque aussi que la médiane des coûts obtenus par l'AG est habituellement plus basse que pour l'OEP ce qui semble démontrer la supériorité des solutions produites par l'AG. Afin que la comparaison soit statistiquement significative, nous calculons au tableau 13 la moyenne des coûts des trajectoires générées par les deux algorithmes pour chacun des scénarios. Nous utilisons ensuite le test-T [58] pour vérifier si la différence des moyennes est significative ou non. Lorsque la différence est significative, nous comparons les deux moyennes et identifions l'algorithme gagnant, soit celui qui produit des solutions avec un coût moyen inférieur. Pour le test-T, nous utilisons un seuil de probabilité de 5 %. En d'autres mots, lorsque la probabilité que la différence des moyennes soit attribuable au hasard (soit la valeur *p* au tableau 13) est moins de 5 %, nous considérons cette différence comme étant significative. Notre analyse démontre que :

1) l'AG a produit des solutions de qualité significativement supérieure à celle de l'OEP pour 25 des 40 scénarios;
2) l'OEP a produit des solutions de qualité significativement supérieure à celle de l'AG pour 3 des 40 scénarios; et
3) l'AG et l'OEP ont produit des solutions de qualité similaire pour 12 des 40 scénarios.

En conclusion, par comparaison de la moyenne lorsque la différence est statistiquement significative, nous démontrons que notre implémentation parallèle de l'AG pour la planification de trajectoires produit de meilleurs résultats que notre implémentation parallèle de l'OEP pour une utilisation en temps réel avec un temps de calcul fixe de 10 s.

Tableau 13. Comparaison statistiquement significative par le test-T de la qualité des solutions produites par l'AG et l'OEP pour les 5 scénarios sur chacun des 8 terrains

Terrain	Scénario	Valeur moyenne des coûts des trajectoires produites		Valeur p (test-T)	Différences statistiquement significatives	Gagnant	
		AG	OEP			AG	OEP
1	1	**0.103**	0.119	0.011	oui	X	
	2	**0.353**	0.368	0.064	non		
	3	**0.519**	0.988	0.025	oui	X	
	4	0.512	**0.503**	0 184	non		
	5	**0.559**	0 718	0.003	oui	X	
2	6	0.342	**0.253**	0.000	oui		X
	7	0.414	**0.357**	0.000	oui		X
	8	**0.550**	0.597	0.027	oui	X	
	9	**0.543**	0.571	0.157	non		
	10	**0.612**	0.917	0.000	oui	X	
3	11	**0.200**	0.740	0 093	non		
	12	**0.327**	0.529	0.000	oui	X	
	13	**0.733**	0.815	0.001	oui	X	
	14	0 460	**0.437**	0.014	oui		X
	15	**0.595**	0.772	0.000	oui	X	
4	16	**0.731**	1.442	0.003	oui	X	
	17	**1.029**	5.745	0.000	oui	X	
	18	**1.292**	3.049	0 000	oui	X	
	19	**1.734**	3.119	0.000	oui	X	
	20	**4.156**	8.670	0.000	oui	X	
5	21	**0.399**	0.434	0 314	non		
	22	**0.842**	0.867	0.195	non		
	23	**0.546**	0.758	0.252	non		
	24	**0.730**	3.318	0 000	oui	X	
	25	0.873	**0.831**	0.060	non		
6	26	**0.484**	1.631	0.000	oui	X	
	27	**0.679**	2.707	0.000	oui	X	
	28	**1.486**	3.197	0.000	oui	X	
	29	**1.125**	2.232	0 000	oui	X	
	30	**1.007**	1.261	0.079	non		
7	31	**0.250**	0.273	0 003	oui	X	
	32	**0.393**	1.598	0.024	oui	X	
	33	**0.625**	0.998	0.106	non		
	34	**0.542**	0.619	0.000	oui	X	
	35	**1.753**	3.317	0.001	oui	X	
8	36	**0.325**	0.691	0.013	oui	X	
	37	**0.590**	3.023	0.000	oui	X	
	38	**0.697**	0.796	0.000	oui	X	
	39	1.020	**0.984**	0 242	non		
	40	**0.810**	1.383	0.079	non		
				TOTAL	**28/40**	**25/40**	**3/40**

Chapitre 9

Intégration à un pilote automatique commercial

Dans le domaine de recherche sur la planification de trajectoires pour les drones, il est souvent trop coûteux ou peu commode d'effectuer des essais de vol. Plusieurs auteurs se limitent donc à l'aspect théorique de leurs solutions. Par exemple, trois étudiants du Collège militaire royal du Canada ont récemment publié des mémoires de maîtrise dans le domaine de la planification de trajectoires pour les drones sans pouvoir effectuer de validation par vols d'essai [14], [15] et [16]. Une validation demanderait l'effort d'une équipe de recherche et non de quelques individus.

Dans le cadre de ce travail, nous abordons cette lacune et développons une plateforme de validation. Cette plateforme pourra être utilisée pour le développement futur d'un module de suivi de trajectoire (soit le module complémentaire à la celui de la planification de trajectoires) et la validation possible de nos algorithmes. Dans ce chapitre, nous traitons de l'installation du pilote automatique MP2128g de MicroPilot [3] dans un avion E-Flite Apprentice [4]. Nous décrivons ensuite le logiciel que nous avons conçu pour interfacer nos algorithmes au pilote automatique. Nous présentons finalement les résultats des tests où nous vérifions le fonctionnement du MP2128g, l'exécution de notre logiciel et la compatibilité de l'interface entre notre logiciel et le MP2128g.

9.1 Installation et calibration du pilote automatique

L'Apprentice de E-Flite [4] est un avion téléguidé de grosseur moyenne, fabriqué en styromousse et tiré par un moteur électrique sans balais. Cet avion s'avère parfait pour notre utilisation puisqu'il est facile à piloter, résistant, léger et offre une protection accrue pour le pilote automatique qui se retrouve entouré de styromousse. Le tableau 14 liste quelques caractéristiques de l'Apprentice dont la photo se trouve à la figure 86.

Tableau 14. Caractéristiques de notre avion E-Flite Apprentice

Caractéristiques	Valeurs
Longueur	94.0 cm
Envergure	147.5 cm
Surface de l'aile	33.7 dm²
Poids sans batterie	1275 g
Batterie	TP8000-3SSR, 11.1 V, 8000 mA, 3 cellules, 531 g
Charge alaire avec batterie	54 g·dm ²
Calibre du moteur	0.25
Calibre du variateur de vitesse électronique	40 A
Calibre de l'hélice	11 x 8
Télécommande	Futaba 8FG 2.4 GHz

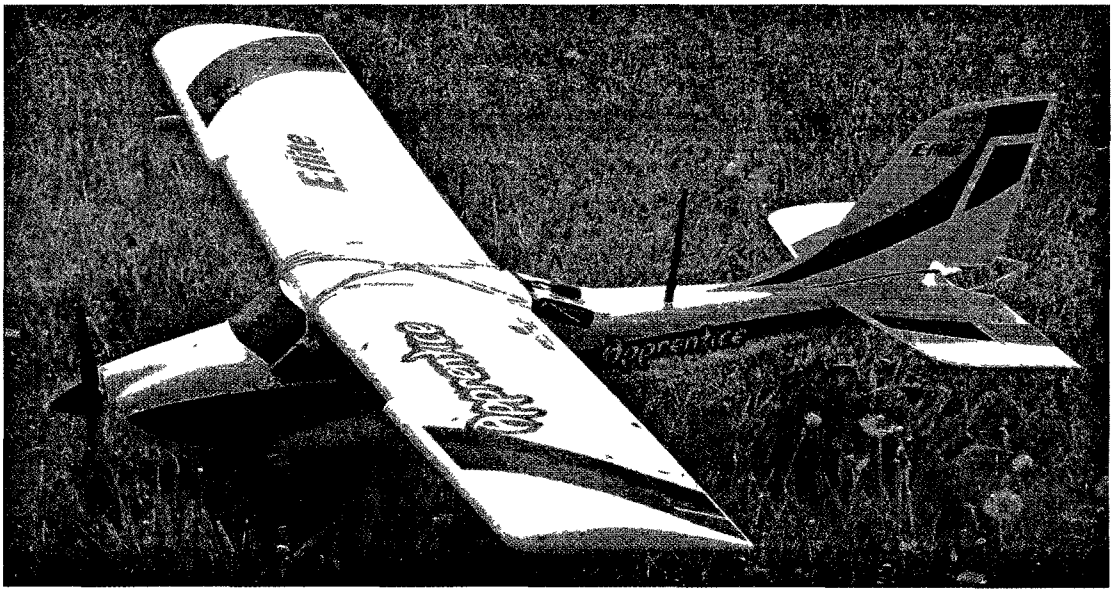


Figure 86. Avion E-Flite Apprentice

Le MP2128g est un pilote automatique développé par MicroPilot [3] et spécialement conçu pour les drones. Le système consiste en un gyroscope à trois axes, un accéléromètre, un GPS, un altimètre barométrique et une prise dynamique (pour un tube de Pitot), le tout intégré sur une carte de 10 cm par 4 cm. Le fabricant prétend qu'il s'agit du plus petit pilote automatique pour les drones au monde [3]. Le MP2128g permet la navigation GPS par points de passages tout en maintenant l'altitude et la vitesse. Le système inclut le logiciel de station de contrôle Horizon [77], aussi développé par MicroPilot. Ce logiciel permet de programmer et d'analyser

l'opération du pilote automatique Avec l'ajout d'un modem au sol et à bord du drone, Horizon peut être utilisé pour contrôler et surveiller les opérations en vol.

Comme il est visible à la figure 87, l'Apprentice offre suffisamment d'espace pour l'installation du pilote automatique Afin de minimiser l'interférence, nous installons le MP2128g dans la section principale, positionné au centre de gravité de l'avion (figure 88) et le modem sans fil, dans la queue (figure 89). L'antenne GPS est positionnée sur le dessus du fuselage, derrière les ailes et le tube de Pitot, sur l'aile gauche loin du corps de l'avion. On retrouve à la figure 90, le schéma des connexions entre le pilote automatique, le modem et les composantes électroniques de l'avion

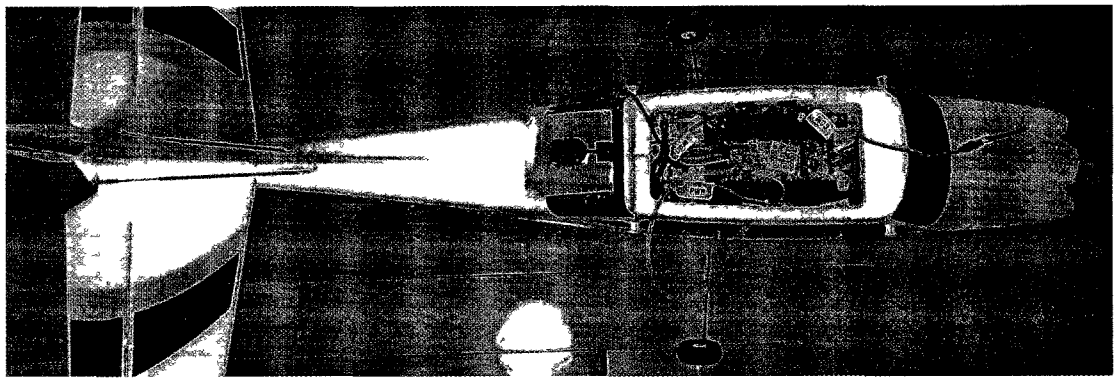


Figure 87 Vue de dessus de l'avion Apprentice equipe du pilote automatique MP2128g

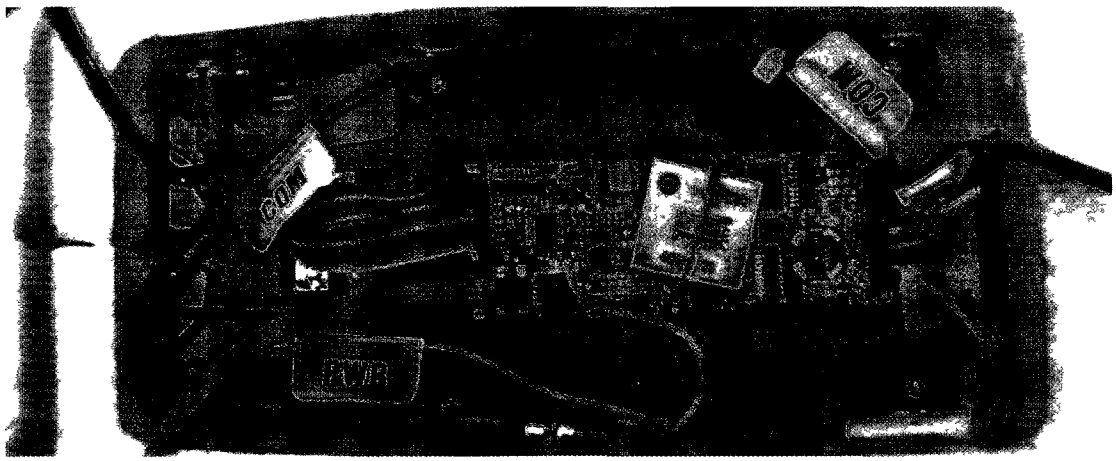


Figure 88 Le MP2128g installé dans le corps de l'Apprentice

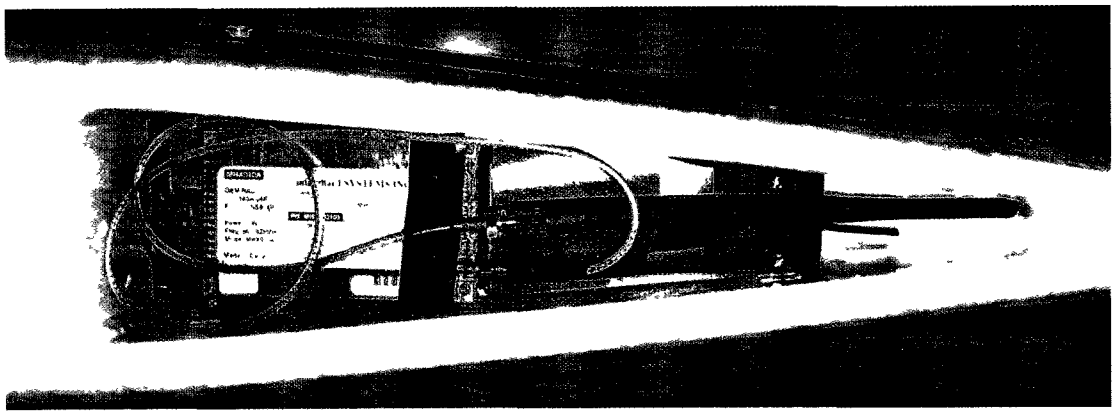


Figure 89. Le modem sans fil installé dans la queue de l'Apprentice

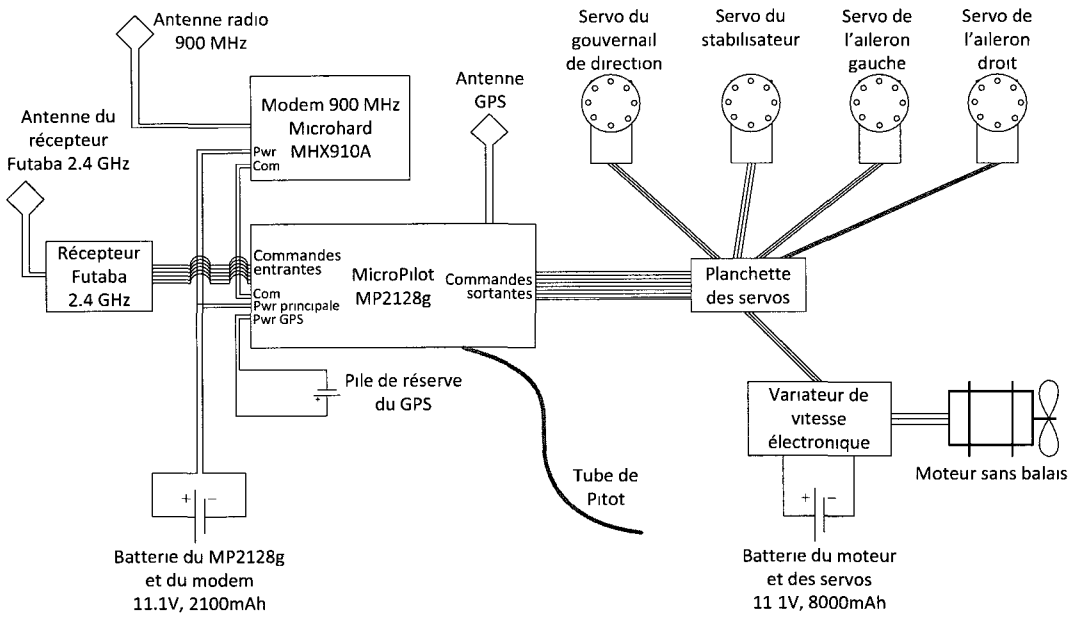


Figure 90. Schéma d'installation du pilote automatique MicroPilot MP2128g

Le MP2128g manœuvre l'avion à l'aide de 13 régulateurs PID (proportionnel, intégral, dérivée). Certains servent à contrôler la dynamique du drone et d'autres, la navigation. Après l'installation du MP2128g, il est nécessaire de calibrer ces régulateurs. La méthode proposée par le fabricant demande plusieurs vols d'essai et nécessite une expertise spécialisée (c'est pourquoi MicroPilot offre un service de calibration). Dans notre cas, puisque nous sommes intéressés à interfacer nos algorithmes au pilote automatique et non à développer le module de suivi de trajectoires, nous avons uniquement calibré les régulateurs contrôlant la dynamique du drone.

9.2 Développement du logiciel de planification de trajectoires

Similairement à plusieurs autres auteurs [14][15][16], nos algorithmes de planification de trajectoires utilisent une représentation de l'environnement sous forme de matrice 2D où chaque cellule contient l'élévation du terrain. Les points de passage d'une trajectoire sont alors représentés sous forme de coordonnées orthogonales (x, y, z) où x est une rangé de la matrice, y, une colonne et z, une altitude. Malgré que ce système de coordonnées soit approprié pour l'algorithme d'optimisation, il n'est pas celui utilisé dans les cartes d'élévation digitales ni par le pilote automatique. Ces deux derniers utilisent plutôt un système de coordonnées sphériques sous forme de latitude et longitude (NAD63 pour les cartes d'élévation digitales GeoBase [5] et WGS84 pour le MP2128g [3]). Or, un tel système n'est pas orthographique : la distance entre les deux coins inférieurs de la carte et les deux coins supérieurs diffèrent de quelques mètres. Pour utiliser nos algorithmes de planification de trajectoires avec de vraies cartes d'élévation digitales et un pilote automatique commercial, nous avons développé un logiciel en MATLAB qui convertit d'abord la carte du terrain dans un système de coordonnée orthogonale, exécute ensuite nos algorithmes et transfère finalement la trajectoire calculée dans le système de coordonnées sphériques utilisé par le pilote automatique.

9.2.1 Fonctionnement du logiciel de planification de trajectoires

La figure 91 montre l'interface graphique du logiciel que nous avons développé. Cette interface contient deux sections principales : la carte interactive à gauche et le panneau de configurations à droite. Il y a aussi un menu dans le haut de la fenêtre, une barre d'état sous la carte interactive et les commandes de vol générées pour le MP2128g dans le bas à droite. L'usager utilise le panneau de configuration pour :

1) afficher une carte satellite provenant de Google Maps [67];
2) y superposer une carte d'élévation digitale provenant de la base de données canadienne GeoBase [5];
3) charger un fichier contenant les caractéristiques du drone (le format de ce fichier est présenté au tableau 15);
4) choisir les propriétés de la trajectoire voulue (ajustement relatif du poids des termes de la fonction de coût associés à la longueur et à l'altitude de la trajectoire); et
5) choisir l'algorithme de planification de trajectoires à utiliser (AG ou OEP).

L'usager peut ensuite se servir de la carte interactive pour :

1) ajouter, modifier et effacer des zones dangereuses;
2) identifier ou modifier la position de la station de contrôle (nécessaire si l'utilisateur veut générer la trajectoire finale en utilisant des coordonnées relatives à la position de la station de contrôle, ce qui est un mode d'opération du MP2128g);
3) ajouter, modifier et effacer des points de passages; et
4) sélectionner les segments du trajet à remplacer par une trajectoire calculée. Par exemple, une mission à objectifs multiples nécessite plusieurs points de passage défini par l'utilisateur. L'algorithme d'optimisation calcule ensuite les trajectoires entre ces objectifs.

Puisque le processus de configuration peut être long, nous avons inclus une fonction qui permet de sauvegarder et charger une configuration à n'importe quel moment. Lorsque l'utilisateur a terminé de configurer un scénario et clique sur le bouton « Generate », notre logiciel convertit la carte d'élévation, les zones dangereuses et les points de passage dans un système de coordonnée orthogonale avant d'exécuter l'algorithme de planification de trajectoires. La trajectoire finale est ensuite transférée dans le système de coordonnées WGS84 pour être affichée en jaune sur la carte interactive. Le logiciel utilise aussi cette trajectoire finale pour générer un fichier de commandes (ou fichier de vol) pour le MP2128g.

Afin de bien expliquer le fonctionnement de notre logiciel, nous incluons à la figure 92 un diagramme d'activité montrant les étapes nécessaires pour définir un scénario et exécuter nos algorithmes de planification de trajectoires.

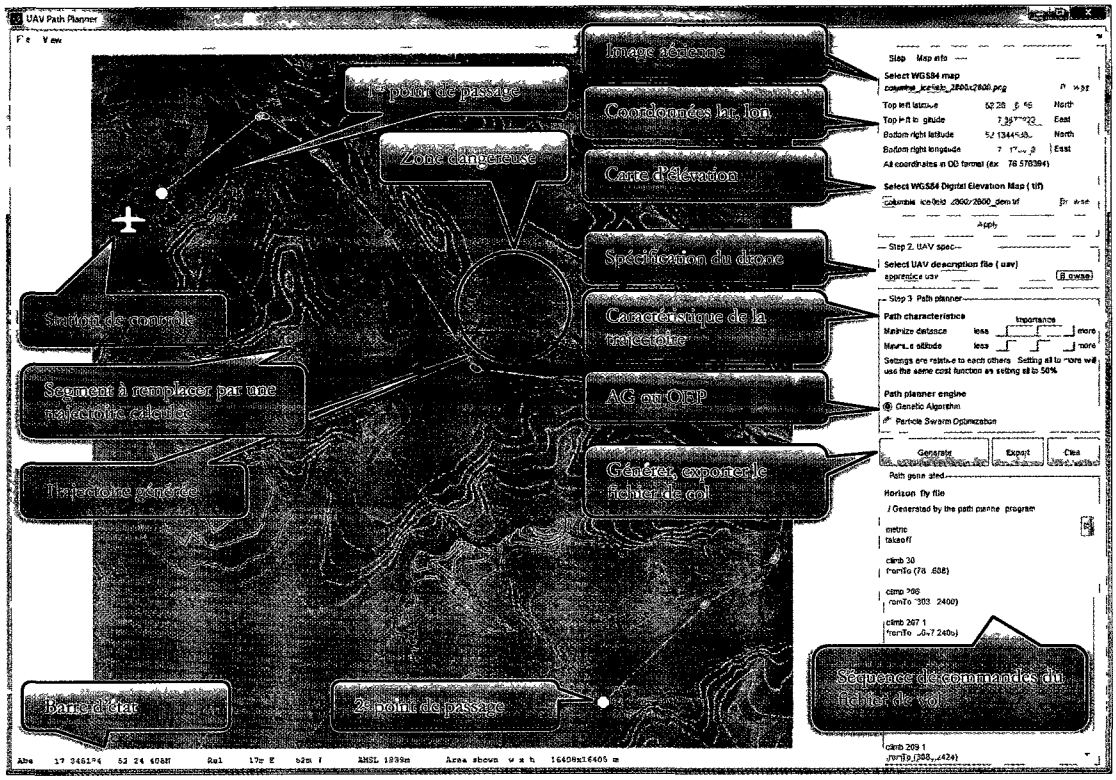


Figure 91 Interface graphique de notre logiciel de planification de trajectoires

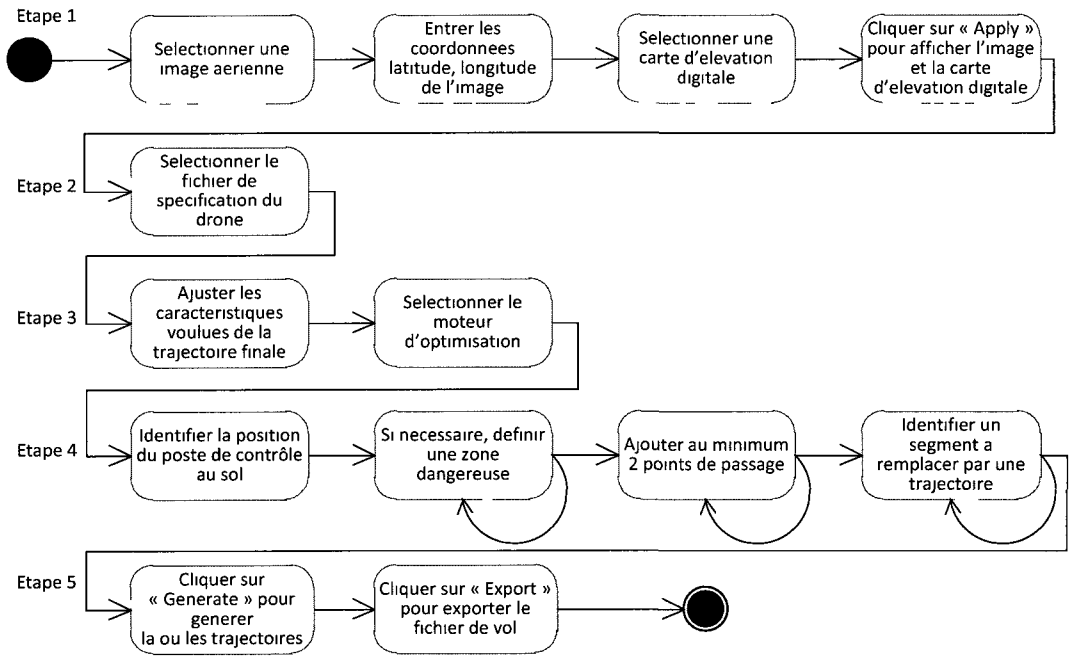


Figure 92 Diagramme d'activites pour la generation d'une trajectoire utilisant notre logiciel

Lorsque l'utilisateur clique sur le bouton « Export », le logiciel sauvegarde le fichier de vol directement dans la structure de fichiers de la station de contrôle Horizon. L'usager peut ensuite utiliser Horizon pour transférer le nouveau fichier de vol au MP2128g suivant les étapes décrites à la figure 93. Si le drone est déjà en vol, le transfert se fait par le modem sans fil et le drone change immédiatement sa trajectoire pour adopter la nouvelle.

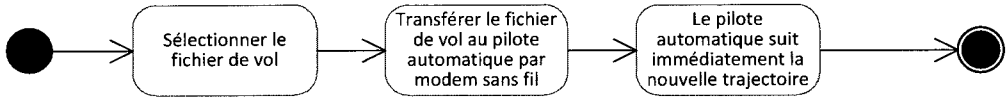


Figure 93. Diagramme d'activités pour le transfert du fichier de vol vers le MP2128g

9.2.2 Structure du logiciel de planification de trajectoires

On retrouve à la figure 94 le diagramme de classes de notre logiciel de planification de trajectoires. Cette implémentation modulaire permet l'ajout ou la modification d'une composante sans affecter le fonctionnement ou la structure du logiciel. Par exemple, il serait facile d'ajouter un troisième algorithme de planification de trajectoires qui utiliserait aussi la même fonction de coût. Notre implémentation utilise le paradigme de programmation orientée objet qui offre un niveau d'abstraction supérieure et permet de mieux gérer la complexité du programme.

Tel qu'illustré sur le diagramme de classes, nous avons défini une classe pour l'interface graphique. Une instance de l' « Interface graphique » possède une « Configuration » qui contient plusieurs éléments, dont les caractéristiques du drone et de la trajectoire. L'utilisation d'une classe « Configuration » permet de sérialiser une instance et donc de sauvegarder une configuration sur le disque dur. L'usager utilise l'interface graphique pour créer un scénario de vol. Lorsqu'il clique sur le bouton « Generate », la classe « Interface graphique » utilise la classe « Planification de trajectoires » qui coordonne l'exécution du moteur d'optimisation. La classe « Planification de trajectoires » utilise ensuite la classe « Fichier de vol Horizon » pour générer un fichier de vol compatible avec le logiciel Horizon. Les coordonnées de la trajectoire générée et la séquence de commandes de vol sont ensuite retournées à l'« Interface graphique » pour être présentées à l'utilisateur.

Dues aux limitations de MATLAB, les algorithmes de planification de trajectoires, la fonction de coût et la fonction de lissage ne sont pas programmés sous forme de classes, mais bien de fonctions. Le PCT de MATLAB ne permet pas d'effectuer des calculs parallèles en

utilisant des classes, mais uniquement des fonctions. Afin de suivre notre approche modulaire pour ces quatre composantes, nous avons pris soin de les subdiviser logiquement en plusieurs sous-fonctions suivant le paradigme de programmation structuré.

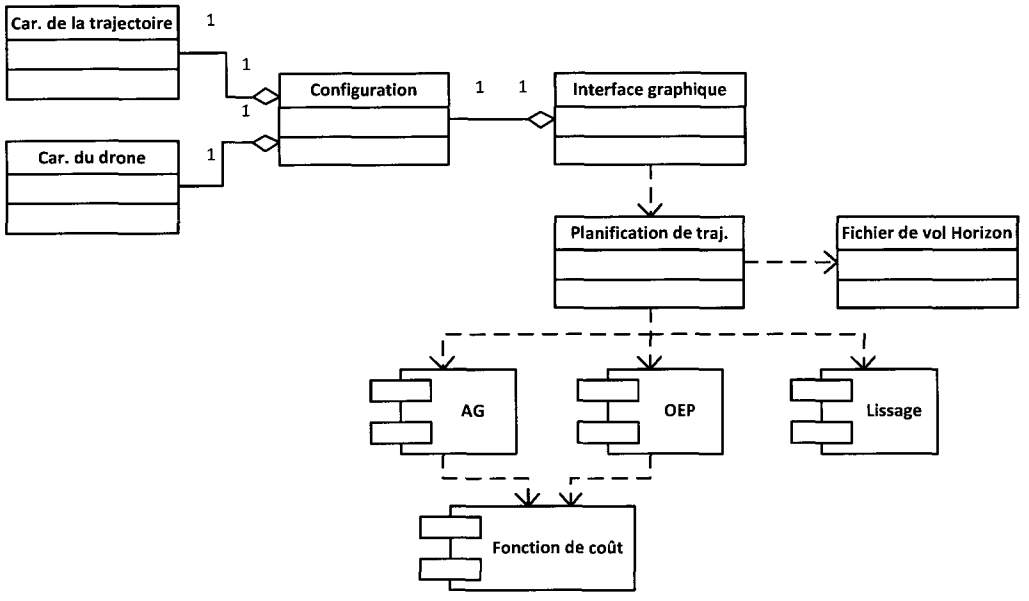


Figure 94. Diagramme de classes de notre logiciel de planification de trajectoires

9.3 Validation du logiciel de planification de trajectoires

Après avoir installé le pilote automatique et développé notre logiciel de planification de trajectoires, nous avons effectué la validation de notre système. Cette validation consiste à utiliser une carte d'élévation digitale provenant du site GeoBase [5], à utiliser notre logiciel de planification de trajectoires pour générer un itinéraire faisable par notre drone et, finalement, de le faire voler sur cette trajectoire en utilisant le pilote automatique. Ce test nous permet de vérifier le bon fonctionnement de notre logiciel, du pilote automatique et de l'interface entre les deux. Une fois validé, ce système pourrait être utilisé pour le développement futur d'un module de suivi de trajectoire et, possiblement, une validation de nos algorithmes de planifications de trajectoire.

Afin de minimiser les risques, nous avons d'abord effectué le test en simulation et ensuite en vol d'essai. Nous présentons dans cette section les résultats obtenus ainsi que plusieurs captures d'écran montrant le fonctionnement de notre système.

9.3.1 Vol en simulation

Notre premier test se fait en simulation et permet de vérifier le bon fonctionnement de notre logiciel et de l'interface entre notre logiciel et Horizon. Le simulateur utilisé est celui inclus dans Horizon. La première étape consiste à effectuer la procédure définie précédemment à la figure 92. Celle-ci inclut la sélection de la carte satellite provenant de Google Map [41]; l'entrée des coordonnées latitudes et longitudes de la carte; la sélection de la carte d'élévation digitale GeoBase [5]; le chargement du fichier contenant les spécifications de notre drone (voir tableau 15); et la création du scénario de vol en utilisant la fenêtre interactive de notre logiciel. L'utilisateur clique ensuite sur le bouton « Generate » pour faire appel à nos algorithmes de planification de trajectoires et générer le fichier de vol. On voit à la figure 95 le trajet généré en jaune ainsi que les commandes de vol affichées dans la boîte de texte en bas à droite. La carte utilisée pour ce test est celle du Glacier Columbia en Alberta.

Tableau 15. Fichier de description de notre drone tel qu'utilisé pour la planification de toutes les trajectoires dans ce document

```
% E-Flite Apprentice with 1x TP8000-3SSR battery, 1x TP2100-3SPL and MicroPilot
% Note: this file is evaluated by MATLAB using the eval() function.
%       It is important to respect the format.

% UAV physical characteristics
maxGs = 2;                          % max load factor (max g force)
emptyWeight = 2.000;                % weight empty (kg)
fuelWeight = 0.5;                   % weight of fuel (kg)
wingSpan = 1.47;                    % wing span (m)
wingArea = 0.337;                   % wing area (m^2)
globalDragCoefficient = 0.0129;     % global drag coefficient at 0 lift
oswaldEfficiency = 0.8607;          % Oswald efficiency number

% UAV flying characteristics
crusingSpeed = 72;                  % cruising speed (km/h)
minFlyAltitude = 30;                % minimum flying height (m)
maxFlyAltitude = 8000;              % max altitude (m) (8850 m for max
                                    % ascent rate of 0.000779 m/s)

% MicroPilot MP2128g flying characteristics
turnRadius = 150;                   % minimum radius when the autopilot is
                                    % changing direction (m)
minDistanceBetweenWaypoints = 10;   % minimum distance between generated
                                    % waypoints for the autopilot (m)
maxBankAngle = 35;                  % maximum bank angle (degrees)
maxClimbAngle = 10;                 % maximum climbing angle (degrees)
maxDescentAngle = 10;               % maximum descent angle (degrees)

% UAV propulsion characteristics
maxPower = 75;                      % available power at sea level (Watt),
                                    % (power*motorEfficiency*propEfficiency)
propEfficiency = 0.53;              % propeller efficiency (0 to 1)
specFuelConsumption = 1e-6;         % thrust-specific fuel con. (N/(sec*Watt))
betaFuelConsumption = 0.299e-8;     % parameters used to calculate the fuel
                                    % consumption (s^-1*N^-0.5)
airFuelRatio = 14.7;                % air/fuel ratio of the engine
```

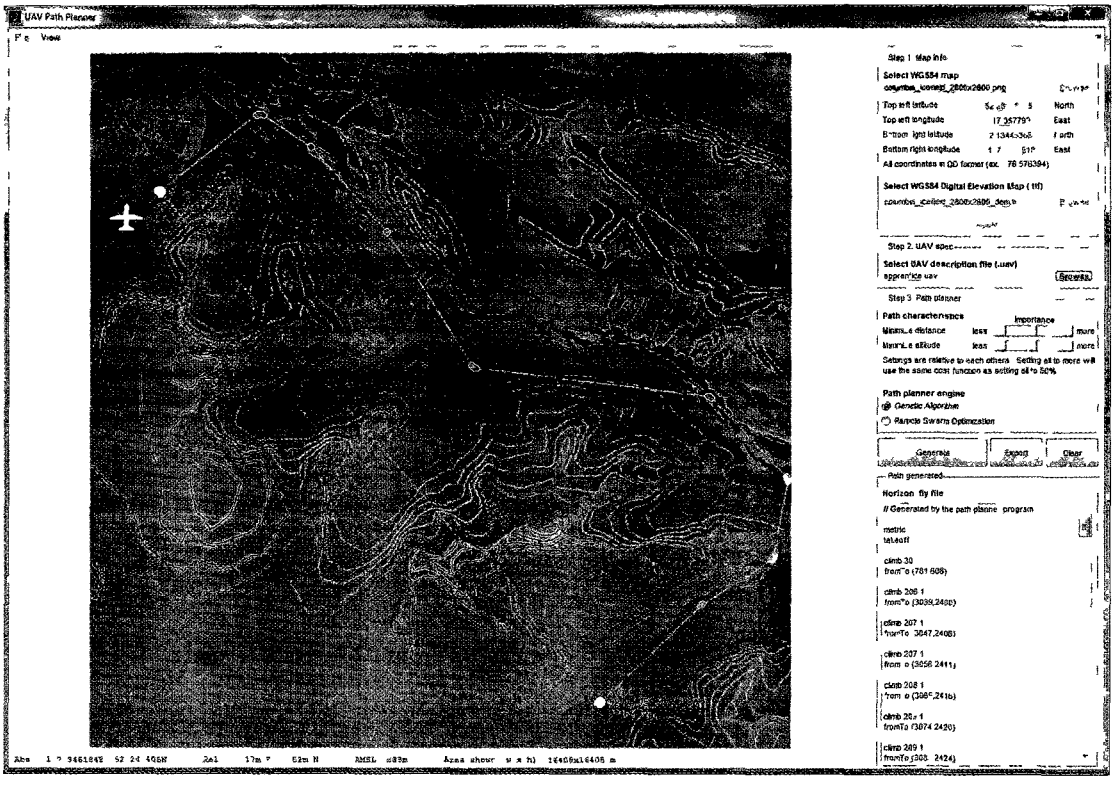


Figure 95 Capture d'écran de notre logiciel après que la trajectoire ait été calculée et le fichier de vol, générée

L'utilisateur peut ensuite cliquer sur le bouton « Export » pour sauvegarder le fichier de vol dans le dossier de Horizon. Le fichier devient alors visible à partir de la station de contrôle Horizon et peut être transféré au pilote automatique en utilisant les commandes sur le panneau de gauche à la figure 96 Tel qu'on le voit sur cette figure, le transfert s'effectue sans problème et la capture d'écran montre le drone en train de suivre la trajectoire en simulation. On peut ensuite analyser les données télémétriques du vol et remarquer que le drone suit parfaitement la trajectoire tout au long de la simulation. Nous pouvons donc confirmer le bon fonctionnement de notre logiciel et celui de l'interface entre notre logiciel et Horizon.

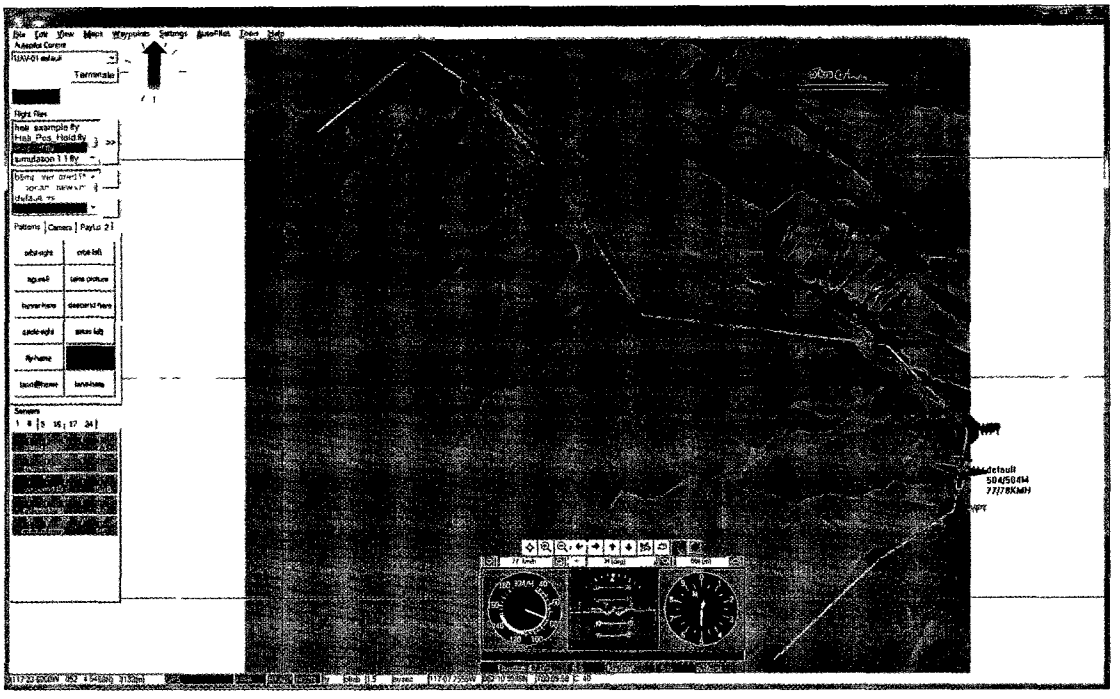


Figure 96. Capture d'écran de la simulation de vol dans Horizon

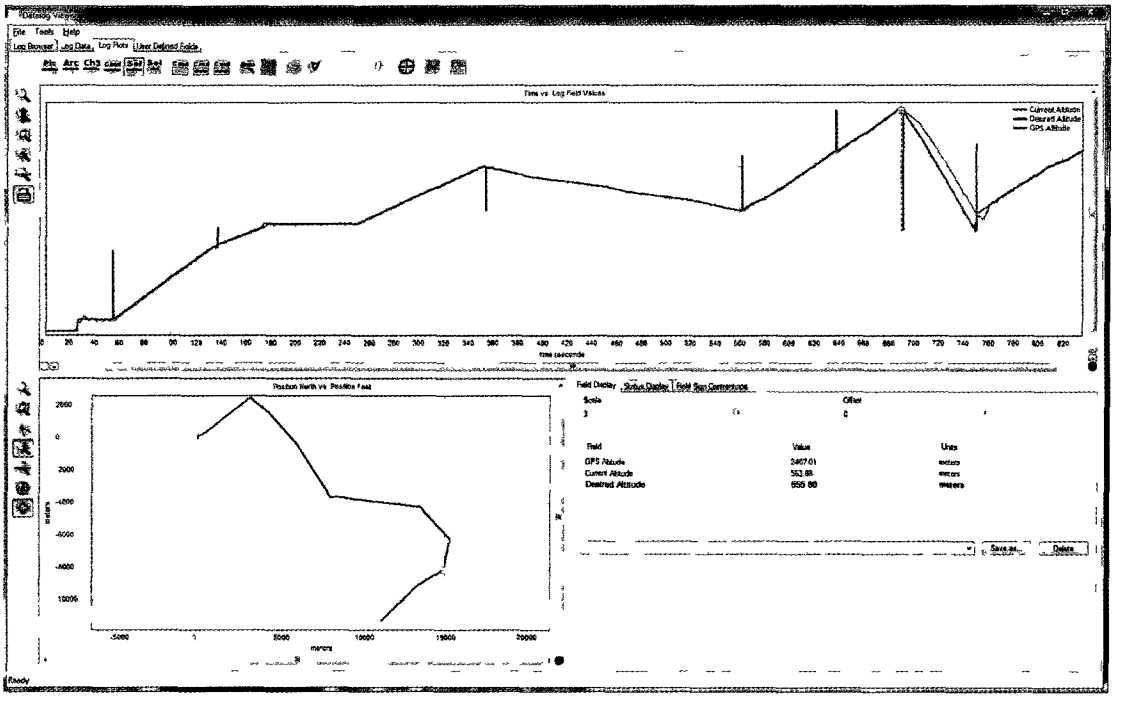


Figure 97. Visualisation des données télémetriques après le vol. Le graphique du haut montre l'altitude désirée (rouge) et l'altitude actuelle (bleu). Le graphique du bas montre la position latitude, longitude du drone pendant la simulation.

9.3.2 Vol d'essai

Le second test consiste à refaire les étapes décrites ci-dessus pour générer la trajectoire pour ensuite effectuer le suivi en vol d'essai. Il serait très intéressant d'utiliser le même terrain qu'au premier test, mais cela est impossible pour plusieurs raisons. Premièrement, il est peu pratique pour nous de se rendre dans cette région. Deuxièmement, pour des mesures de sécurité, nous devons toujours garder un contact visuel et radio avec le drone pendant le vol autonome ce qui ne serait pas possible dans une région montagneuse. Finalement, nous avons déterminé par expérimentation que le modem que nous utilisons (dans la configuration présentée à la figure 90) a une portée maximale d'environ 500 m seulement. Nous effectuons donc le vol d'essai sur un terrain plat de 1 000 m par 800 m. Pour rendre le scénario plus intéressant, nous superposons une carte d'élévation fictive sur ce terrain plat. Tel qu'illustré à la figure 98, la trajectoire générée par nos algorithmes évite les zones dangereuses, contourne les montagnes fictives tout en minimisant son altitude moyenne et sa longueur.

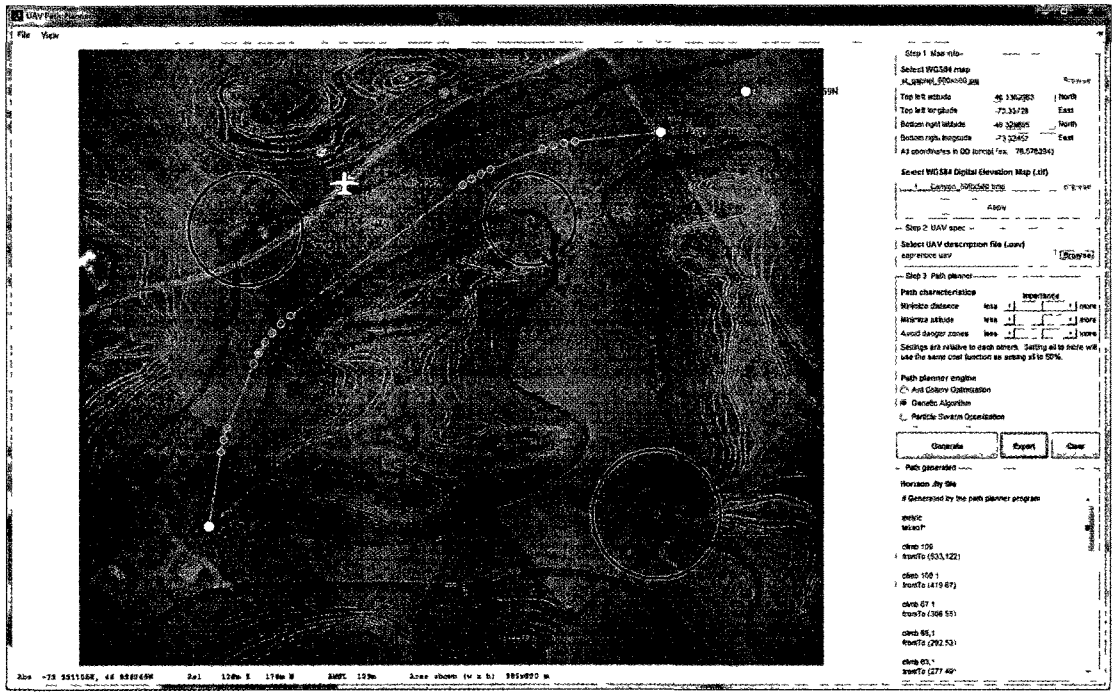


Figure 98. Capture d'écran de notre logiciel après que la trajectoire ait été calculée et le fichier de vol, généré

Le transfert du fichier de vol vers la station de contrôle Horizon se fait encore sans problème. Afin de démontrer la capacité MP2128g à changer sa trajectoire en vol, nous programmons d'abord le pilote automatique pour effectuer une boucle au-dessus de la station de

contrôle puis téléchargeons ensuite le fichier de commandes en utilisant le modem sans fil. Le transfert de données prend quelques secondes puis le drone change immédiatement sa trajectoire pour suivre celle calculée par nos algorithmes. Cette complexité peut sembler superflue, mais confirme qu'il est possible de modifier le trajet du drone en vol. Cette fonctionnalité est essentielle pour qu'un drone autonome puisse mettre à jour sa trajectoire à mesure qu'il avance.

À la figure 99, la différence entre la trajectoire calculée et celle suivie peut sembler considérable. Cependant, il est important de se rappeler que notre objectif est de valider notre logiciel, le fonctionnement du pilote automatique et l'interface entre les deux, et non pas d'évaluer le module de suivi de trajectoire. Lors de la calibration du MP2128g, par manque de temps et d'expertise, nous n'avons pas calibré les régulateurs contrôlant la navigation du drone, mais uniquement ceux contrôlant sa stabilité. De plus, la longueur totale du trajet est trop courte pour être représentative d'une opération normale d'un drone. Les scénarios présentés au chapitre précédent varient de 5 à 60 km de longueur et le drone aurait alors le temps de se stabiliser entre les virages. Aussi, nous avons conduit ce vol d'essai dans des conditions météorologiques défavorables. À partir de la figure 100, on peut comparer la vitesse GPS à la vitesse anémométrique et déduire que les bourrasques de vent provenant du Nord-Ouest étaient de 30 $km \cdot h^{-1}$, ce qui déstabilise facilement notre drone dû à son poids léger et sa vitesse de croisière lente. On peut voir à la figure 101 un second graphique montrant l'altitude désirée et actuelle du drone pendant le vol d'essai.

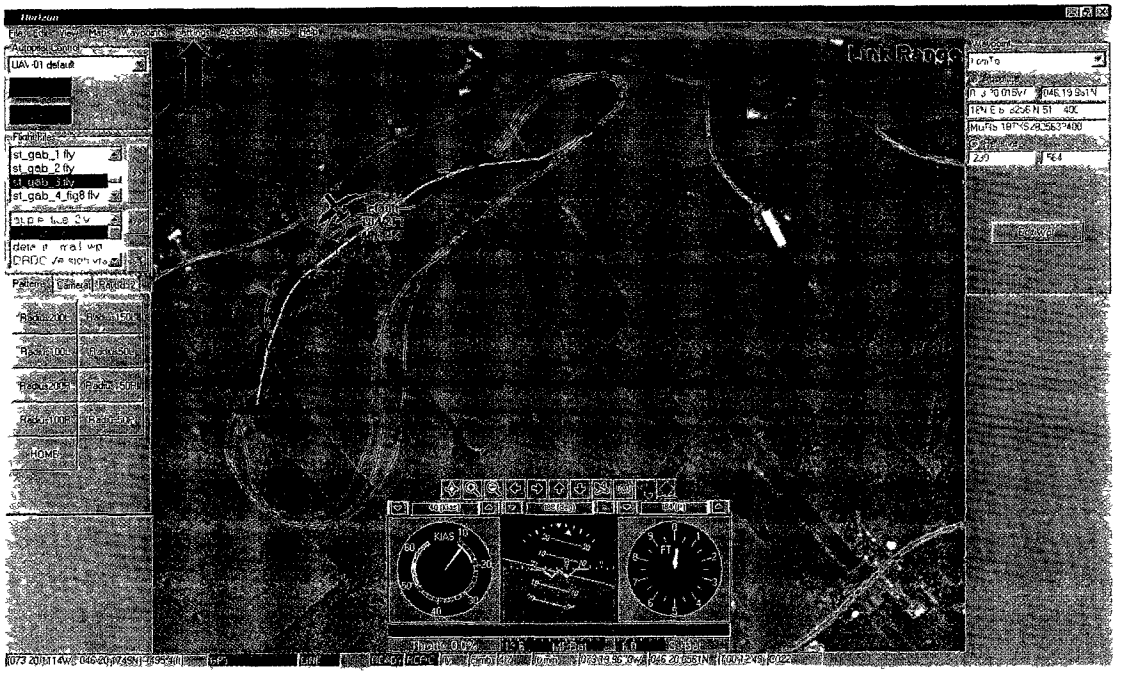


Figure 99. Capture d'écran de Horizon montrant la trajectoire calculée (en jaune) et suivie (en noir) par l'avion pendant le vol d'essai.

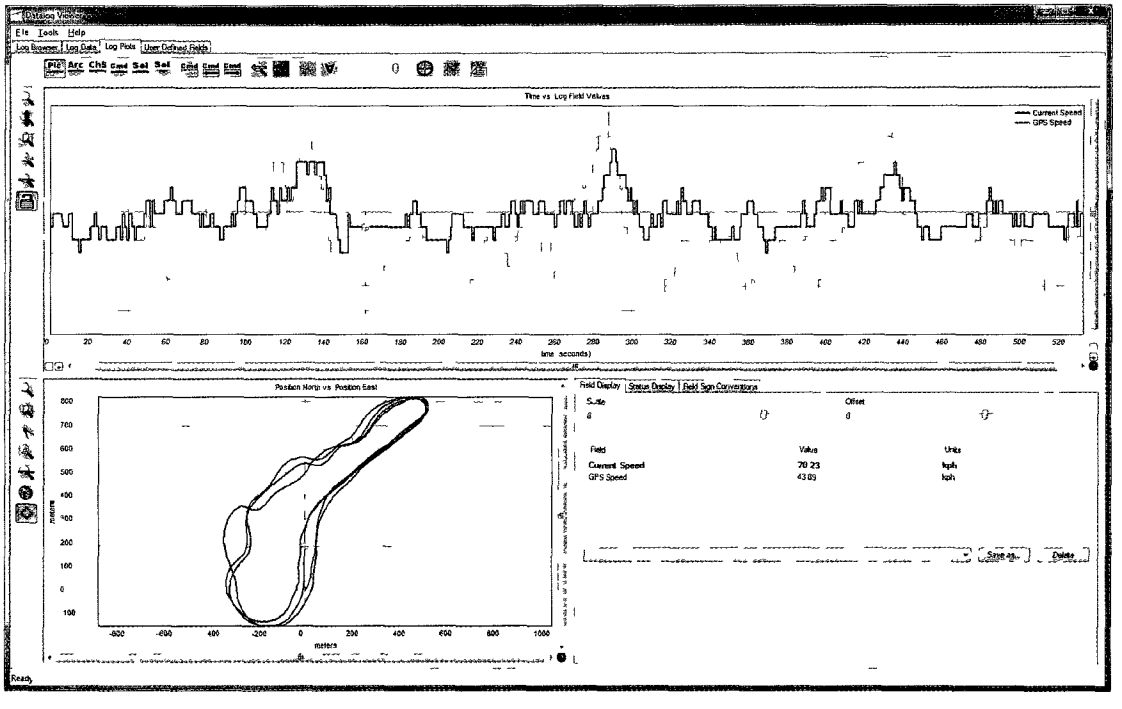


Figure 100 Visualisation des données télémétriques après le vol. Le graphique du haut montre la vitesse GPS du drone (bleu foncé) et la vitesse anémométrique (turquoise). Le graphique du bas montre la position latitude, longitude du drone pendant le vol.

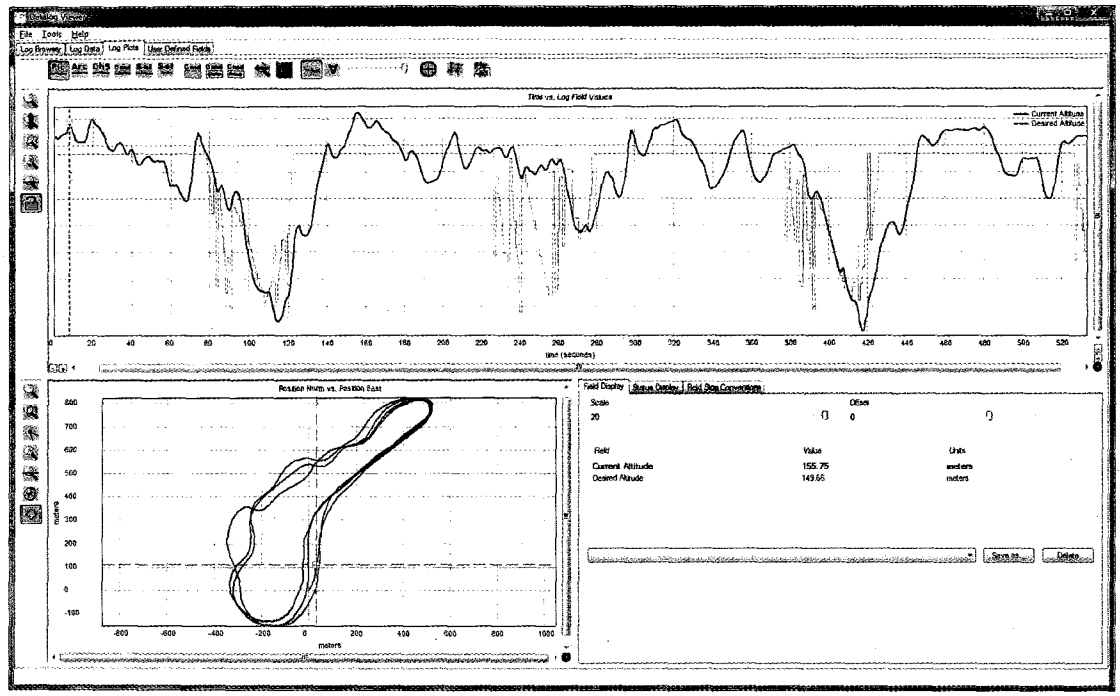


Figure 101. Visualisation des données télémétriques après le vol. Le graphique du haut montre l'altitude désirée (turquoise) et l'altitude actuelle (bleu foncé). Le graphique du bas montre la position latitude, longitude du drone pendant le vol.

En conclusion, les résultats des deux tests démontrent le bon fonctionnement de notre logiciel, du pilote automatique et de l'interface entre les deux systèmes. Au cours de ces tests, nous avons utilisé des terrains réels, nous avons généré des trajectoires faisables que nous avons ensuite suivies avec succès en simulation et en vol d'essai à l'aide d'un avion E-Flite Apprentice équipé d'un pilote automatique MP2128g.

Chapitre 10

Conclusion

En conclusion, nous récapitulons brièvement le contenu de chacun des chapitres, nous soulignons les principales contributions apportées par notre recherche et nous proposons quelques sujets de travaux futurs.

10.1 Récapitulation

Au chapitre 1, nous expliquons que la planification de trajectoires pour les drones consiste à calculer une route optimale et dynamiquement faisable entre un point de départ et un point d'arrivée. Il s'agit en fait d'un problème d'optimisation multidimensionnelle très complexe qui nécessite une approche non déterministe. Élément essentiel à l'autonomie d'un drone, la planification automatique de trajectoire est actuellement d'un grand intérêt pour plusieurs organisations gouvernementales et militaires. Les objectifs de ce mémoire répondent aux lacunes des solutions existantes et participent à l'avancement de ce domaine.

Nous présentons au chapitre 2 les différentes solutions proposées antérieurement pour résoudre le problème de la planification de trajectoires. Celles-ci suivent souvent une approche par étapes qui se résume par : 1) la représentation de l'environnement; 2) la définition de la fonction de coût; 3) le calcul d'une route quasi optimale à l'aide d'un algorithme d'optimisation; et 4) le lissage de la trajectoire finale. Nous suivons cette même approche pour le développement de notre module de planification de trajectoires. Nous identifions dans ce chapitre quelques limitations des solutions actuelles : l'utilisation fréquente de modèles trop simplistes pour être correcte, le temps de calcul souvent trop long pour permettre une application en temps réel, le manque de comparaisons rigoureuses entre les différents algorithmes proposés et la rareté des validations par vol d'essai. Nous expliquons ensuite comment notre recherche répond à ces problèmes.

Nous présentons au chapitre 3 la représentation de l'environnement que nous utilisons et la fonction de coût que nous avons développée. Cette fonction inclut les différentes caractéristiques d'une trajectoire optimale et est utilisée par les algorithmes d'optimisation pour générer la route.

Les chapitres 4 et 5 traitent de nos implémentations de l'AG et de l'OEP. Contrairement à plusieurs auteurs qui incluent des modifications pour forcer la qualité des solutions, nous reproduisons avec soin les algorithmes originaux afin de conserver leur efficacité prouvée au cours des années. En fait, les améliorations apportées par des modifications sont souvent restreintes aux exemples utilisés par l'auteur et rendent l'algorithme d'optimisation moins robuste. Après avoir implémenté l'AG et l'OEP, nous apportons quelques optimisations à notre code au niveau de la gestion de variables et de l'évaluation du coût et diminuions ainsi le temps de calcul de 321 % dans le cas de l'AG et de 183 % dans le cas de l'OEP.

La route générée par les moteurs d'optimisation est composée de segments de droite et inclut des discontinuités dans la vitesse. Nous traitons au chapitre 6 du lissage de la trajectoire par les arcs de cercle et les cercles sur plateaux horizontaux. Cette approche provient de [37] et a l'avantage d'être suffisamment simple pour permettre une vérification complète des contraintes de faisabilité en tout endroit sur trajectoire finale.

Afin de réduire le temps de calcul de nos algorithmes de planification de trajectoires, nous développons des versions parallèles suivant le paradigme de programmation « instruction unique, données multiples ». L'accélération atteinte est quasi linéaire et permet un temps d'exécution plus de 7 fois plus court pour les deux algorithmes sur un ordinateur de bureau à 8 cœurs. Les mesures présentées laissent croire que cette accélération s'améliorera avec un plus grand nombre de cœurs. Les temps d'exécution des programmes parallèles est alors réduit à 10 s et permet une planification en temps réel.

Nous comparons la qualité des trajectoires produites par l'AG et l'OEP au chapitre 8. Cette comparaison se fait en utilisant une fonction de coût unique et 40 scénarios distincts sur 8 terrains différents (dont 6 sont réels). Nous évaluons chaque trajectoire 60 fois afin d'obtenir suffisamment de données pour permettre une comparaison valide et statistiquement significative. Nos résultats démontrent que l'AG produit de meilleures solutions.

Finalement, nous traitons au chapitre 9 de l'installation du pilote automatique MP2128g de MicroPilot [3] dans un avion E-Flite Apprentice [4]; du développement de l'outil logiciel permettant d'utiliser des cartes d'élévation digitales GeoBase [5] et de générer un fichier de vol; et de la validation de l'interface entre l'outil logiciel et le pilote automatique par vol d'essai.

10.2 Contributions

En réponse aux limitations des solutions présentées au chapitre 2, notre recherche apporte trois contributions distinctes.

Premièrement, nous proposons deux implémentations basées sur des algorithmes d'optimisation non déterministes qui utilisent un modèle suffisamment complet pour générer des solutions faisables et quasi optimales dans un temps relativement court. Nous réduisons davantage ce temps de calcul par une approche parallèle et permettons ainsi la planification en temps réel.

Deuxièmement, nous avons défini une fonction de coût inédite où chacun des termes est normalisé et représente une qualité recherchée pour une trajectoire optimale. Nous considérons cette fonction plus complète que celles couramment utilisées dans la littérature puisqu'elle permet la génération de trajectoires par des algorithmes d'optimisation génériques sans nécessiter l'ajout d'opérations manuelles comme il est souvent fait pour forcer la qualité de la solution finale. Notre fonction de coût est donc universelle et utilisable par différents algorithmes d'optimisation et peut être utilisée pour les comparer.

Troisièmement, nous présentons les résultats d'une comparaison rigoureuse entre l'AG et l'OEP appliqués à la planification de trajectoires pour les drones. Malgré que l'AG soit la méthode la plus couramment utilisée pour la planification de routes, l'OEP a récemment gagnée beaucoup de popularité. Nous avons cependant démontré que l'AG produit de meilleurs résultats et gère mieux la complexité des trajectoires à plusieurs points de passages.

Finalement, dans le but de permettre une validation future de nos algorithmes par vol d'essai, nous avons développé une plateforme test. Celle-ci inclut le logiciel que nous avons conçu, l'avion E-FLite Apprentice [4] et le pilote automatique MP2128g de MicroPilot [3]. Nous avons validé l'interface entre nos algorithmes et le pilote automatique par vol d'essai.

10.3 Travaux futurs

10.3.1 Validation par vols d'essai

La faisabilité des trajectoires générées par nos algorithmes est vérifiée par la fonction de coût d'après les spécifications du drone et les équations d'aérodynamique présentées au

chapitre 3. Il serait toutefois important de valider nos algorithmes par essai de vol sur notre plateforme test. Les routines de contrôle du pilote automatique peuvent être utilisées sans modification, mais nécessitent d'abord un ajustement plus rigoureux.

10.3.2 Développement d'un module de suivi de trajectoires

Le MP2128g est entièrement programmable et permet le développement d'un système de suivi de trajectoire sur mesure. Un sujet de recherche possible serait alors de développer et d'analyser un tel module. Dans le cas du suivi des trajectoires que nous avons générées, le travail pourrait être très laborieux. Puisque nous avons utilisé les caractéristiques dynamiques du drone, les routes calculées sont toutes faisables, mais nécessitent peut-être des manœuvres difficiles. Une première solution serait de sous-estimer les valeurs entrées dans le fichier de spécification du drone (voir tableau 15). Les trajectoires générées seraient donc plus facilement suivies. Une seconde solution consisterait à modéliser le comportement dynamique du système avion-pilote automatique et à générer des trajectoires qui le respectent. Cette approche ne demande aucune modification au MP2128g, mais nécessite un grand nombre de vols d'essai pour bien modéliser le véhicule équipé de son système de contrôle.

10.3.3 Implémentation parallèle sur processeur graphique

Notre recherche démontre qu'il est possible de réduire significativement le temps d'exécution de la planification de trajectoires par implémentation parallèle sur microprocesseurs multicœurs. Il serait facile d'étendre ce parallélisme à une grappe d'ordinateurs et l'accélération obtenue serait probablement très bonne puisque nous avons limité la communication entre les processus dans nos implémentations parallèles. Ce port nécessiterait l'utilisation de « MATLAB Distributed Computing Server » pour notre code MATLAB et de MPICH2 pour une implémentation en C.

La planification de trajectoires sur une grappe d'ordinateurs serait très intéressante, mais peu utile pour une application embarquée à bord du drone. Il serait toutefois possible d'utiliser un processeur graphique (GPU) pour accélérer davantage l'exécution du programme. Tel qu'expliqué dans [78], CUDA (une extension au langage C et C++) peut être utilisé pour effectuer des calculs scientifiques sur GPU. Cette technologie date de 2007 et permet d'accéder à une capacité de calcul environ 100 fois plus grande que celle d'un microprocesseur multicœur (voir figure 102). Tel que visible à la figure 103, cette puissance est produite par l'architecture

massivement parallèle du GPU (par exemple, la carte GeForce GTX590 de NVIDIA possède 1024 cœurs [79]) et requiert une bonne extensibilité de l'implémentation parallèle. Dans le cas de nos implémentations, nous avons considérablement réduit la communication entre les processus et une telle extensibilité semble probablement possible. De plus, puisque les GPU sont relativement petits, une application embarquée pourrait être envisageable.

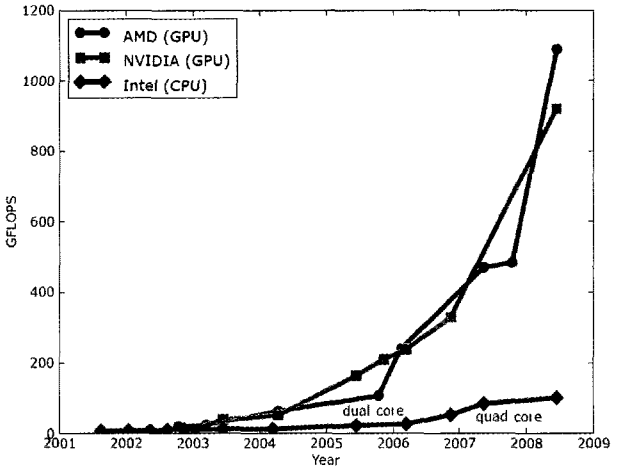


Figure 102. Évolution de la puissance de calcul des microprocesseurs et des processeurs graphiques (reproduit de [78])

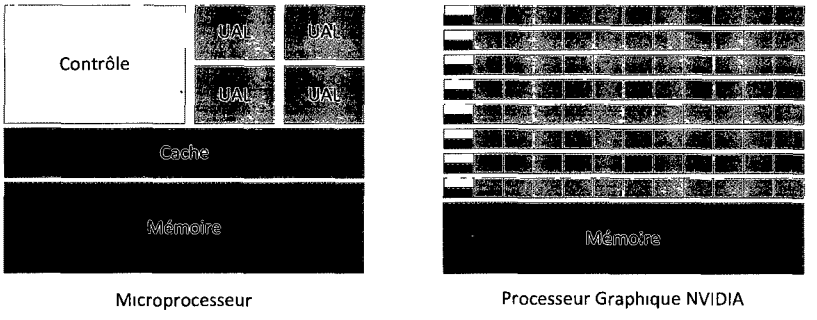


Figure 103. Comparaison de l'architecture parallèle d'un microprocesseur multicœur et d'un processeur graphique (reproduit de [78])

Références

[1] C. Goerzen, Z. Kong, and B. Mettler, "A Survey of Motion Planning Algorithms from the Perspective of Autonomous UAV Guidance," *J. Intell. Robotics Syst.*, vol. 57, 2010, pp. 65-100.

[2] H. Chen, X.-min Wang, and Y. Li, "A Survey of Autonomous Control for UAV," *Proceedings of the 2009 International Conference on Artificial Intelligence and Computational Intelligence - Volume 02*, IEEE Computer Society, 2009, pp. 267-271.

[3] "MicroPilot - Products - MP2128g" Disponible: http://www.micropilot.com/products-mp2128g.htm.

[4] "Apprentice 15e RTF" Disponible: http://www.e-fliterc.com/Products/Default.aspx?ProdID=EFL2725.

[5] "GeoBase - Canadian Digital Elevation Data" Disponible: http://www.geobase.ca/geobase/en/data/cded/index.html;jsessionid=E999310F1B8F4087A4913460B9E5EE47.

[6] D. Pugliese, "Canadian Forces will continue operating Heron UAVs in Afghanistan; project extended," *Ottawa Citizen*, May. 2010.

[7] "ScanEagle Surpasses 17,000 Combat Flight Hours with Canadian Forces," *ASD News*, Jun. 2010.

[8] D. Pugliese, "Maveric UAV purchased by the Canadian Forces," *Ottawa Citizen*, Aug. 2010.

[9] D. Pugliese, "Forces surveillance drone program short of remote pilots, say experts," *Calgary Herald*, May. 2010.

[10] D. Pugliese, "Lack of operators clips wings of drone plan," *Ottawa Citizen*, May. 2010.

[11] N. Sariff and N. Buniyamin, "An overview of autonomous mobile robot path planning algorithms," *2006 4th Student Conference on Research and Development, 27-28 June 2006*, Piscataway, NJ, USA: IEEE, 2006, pp. 183-8.

[12] D. Henrich, "Fast Motion Planning by Parallel Processing - A Review," *J. Intell. Robotics Syst.*, vol. 20, 1997, pp. 45-69.

[13] C. Cocaud, "Autonomous Tasks Allocation and Path Generation of UAV's," M.A.Sc. thesis, University of Ottawa, 2006.

[14] F.C.J. Allaire, "FPGA Implementation of an Unmanned Aerospace Vehicle Path Planning Genetic Algorithm," M.A.Sc. thesis, Royal Military College of Canada, 2007.

[15] D.G.H. Bélanger, "Trajectory Planning with Ant Colony Optimization," M.S. thesis, Royal Military College of Canada, 2008.

[16] N. Eslam Pour, "Path Planning of Unmanned Aerial Vehicles Using Swarm Optimization and B-Spline Curves," M.S. thesis, Royal Military College of Canada, 2009.

[17] X. Yu and M. Gen, *Introduction to Evolutionary Algorithms*, London: Springer, 2010.

[18] D. Ferguson, M. Likhachev, and A. Stentz, "A Guide to Heuristic-based Path Planning," *American Association for Artificial Intelligence*, 2005.

[19] E. Masehian and D. Sedighizadeh, "Classic and Heuristic Approaches in Robot Motion Planning - A Chronological Review," 2007.

[20] J.H. Holland, *Adaptation in Natural and Artificial Systems*, 1975.

[21] Y.V. Pehlivanoglu and A. Hacioglu, "Vibrational genetic algorithm based path planner for autonomous UAV in spatial data based environments," *3rd International Conference on Recent Advances in Space Technologies, RAST 2007, June 14, 2007 - June 16, 2007*, Istanbul, Turkey: Inst. of Elec. and Elec. Eng. Computer Society, 2007, pp. 573-578.

[22] H. Meng and G. Xin, "UAV route planning based on the genetic simulated annealing algorithm," China: IEEE, 2010.

[23] M. Dorigo, "Optimization, Learning and Natural Algorithms," PhD thesis, Politecnico di Milano, 1992.

[24] J.-L. Deneubourg, P.-L. Clip, and S.S. Camazine, "Ants, buses and robots-self-organization of transportation systems," *Proceedings of PerAc '94. From Perception to Action, 7-9 Sept. 1994*, Los Alamitos, CA, USA: IEEE Comput. Soc. Press, 1994, pp. 12-23.

[25] J. Kennedy and R. Eberhart, "Particle swarm optimization," *Proceedings of the 1995 IEEE International Conference on Neural Networks. Part 1 (of 6), November 27, 1995 - December 1, 1995*, Perth, Aust: IEEE, 1995, pp. 1942-1948.

[26] Y. Shi and R. Eberhart, "A modified particle swarm optimizer," *1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence, 4-9 May 1998*, New York, NY, USA: IEEE, 1998, pp. 69-73.

[27] Fei Wang and Baiquan Lu, "The robot path planning based on PSO algorithm integrated with PID," *2009 International Conference on Information Engineering and Computer Science. ICIECS 2009, 19-20 Dec. 2009*, Piscataway, NJ, USA: IEEE, 2009, p. 4 pp.

[28] Z. Hongguo, Z. Changwen, H. Xiaohui, and L. Xiang, "Path planner for unmanned aerial vehicles based on modified PSO algorithm," *2008 IEEE International Conference on Information and Automation, ICIA 2008, June 20, 2008 - June 23, 2008*, Zhangjiajie, Hunan, China: Inst. of Elec. and Elec. Eng. Computer Society, 2008, pp. 541-544.

[29] M. Qianzhi, L. Xiujuan, and Z. Qun, "Mobile robot path planning with complex constraints based on the second-order oscillating particle swarm optimization algorithm," *2009 WRI World Congress on Computer Science and Information Engineering, CSIE 2009, March 31, 2009 - April 2, 2009*, Los Angeles, CA, United states: IEEE Computer Society, 2009, pp. 244-248.

[30] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi, "Optimization by Simulated Annealing," May. 1983, pp. 671-680.

[31] H. Miao and Y.-C. Tian, "Robot path planning in dynamic environments using a simulated annealing based approach," *2008 10th International Conference on Control, Automation, Robotics and Vision, ICARCV 2008, December 17, 2008 - December 20, 2008*, Hanoi, Viet nam: Inst. of Elec. and Elec. Eng. Computer Society, 2008, pp. 1253-1258.

[32] R. Durbin and D. Willshaw, "Analogue approach to the travelling salesman problem using an elastic net method," *Nature*, vol. 326, 1987, pp. 689-691.

[33] J.A. Moreno and M. Castro, "Heuristic algorithm for robot path planning based on a growing elastic net," *12th Portuguese Conference on Artificial Intelligence, EPIA 2005 - Progress in Artificial Intelligence, December 5, 2005 - December 8, 2005*, Covilha, Portugal: Springer Verlag, 2005, pp. 447-454.

[34] I. Hasircioglu, H.R. Topcuoglu, and M. Ermis, "3-D path planning for the navigation of unmanned aerial vehicles by using evolutionary algorithms," *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, Atlanta, GA, USA: ACM, 2008, pp. 1499-1506.

[35] E.P. Anderson, R.W. Beard, and T.W. McLain, "Real-time dynamic trajectory smoothing for unmanned air vehicles," *IEEE Transactions on Control Systems Technology*, vol. 13, May. 2005, pp. 471-7.

[36] C.L. Bottasso, D. Leonello, and B. Savini, "Path planning for autonomous vehicles by trajectory smoothing using motion primitives," *IEEE Transactions on Control Systems Technology*, vol. 16, 2008, pp. 1152-1168.

[37] G. Labonté, "Sur la Construction de trajectoires dynamiquement réalisables pour les avions à partir de suites de segments de droites, 2e version," Nov. 2009.

[38] S.B. Hamida and A. Petrowski, "The need for improving the exploration operators for constrained optimization problems," *Proceedings of the 2000 Congress on Evolutionary Computation*, La Jolla, CA, USA: IEEE, , pp. 1176-1183.

[39] G. Labonté, "Approximations de la consommation de carburant des avions sur des trajectoires ascendantes à vitesse constante," Nov. 2010.

[40] "Bresenham's line algorithm - Wikipedia, the free encyclopedia" Disponible: http://en.wikipedia.org/wiki/Bresenham%27s_line_algorithm.

[41] F. Rothlauf, *Representation for Genetic and Evolutionary Algorithms*, Springer Berlin Heidelberg, 2006.

[42] S.N. Sivanandam and S.N. Deepa, *Introduction to Genetic Algorithms*, Berlin Heidelberg New York: Springer, 2008.

[43] R. Hassan, B. Cohanim, O. de Weck, and G. Venter, "A Comparison of Particle Swarm Optimization and the Genetic Algorithm," 2004.

[44] K.E. Parsopoulos and M.N. Vrahatis, *Particle Swarm Optimization and Intelligence, Advances and Application*, New York: Hershey, 2010.

[45] G.E. Moore, "Cramming more components onto integrated circuits," *Electronics*, vol. 38, Apr. 1965.

[46] G. Moore, "Excerpts from A Conversation with Gordon Moore: Moore's Law," 2005.

[47] T. Rauber and G. Rünger, *Parallel Programming for Multicore and Cluster Systems*, New York: 2010.

[48] R. Serrano, J. Tapia, O. Montiel, R. Sepúlveda, and P. Melin, "High Performance Parallel Programming of a GA Using Multi-core Technology," *Soft Computing for Hybrid Intelligent Systems*, O. Castillo, P. Melin, J. Kacprzyk, and W. Pedrycz, eds., Springer Berlin / Heidelberg, 2008, pp. 307-314.

[49] "MATLAB - The Language Of Technical Computing" Disponible: http://www.mathworks.com/products/matlab/.

[50] L. Nan, G. Pengdong, L. Yongquan, and Y. Wenhua, "The Implementation and Comparison of Two Kinds of Parallel Genetic Algorithm Using Matlab," *Proceedings of the 2010 Ninth International Symposium on Distributed Computing and Applications to Business, Engineering and Science*, IEEE Computer Society, 2010, pp. 13-17.

[51] "Function Reference (Parallel Computing Toolbox™)" Disponible: http://www.mathworks.com/help/toolbox/distcomp/f1-6010.html.

[52] C. Guifen, W. Baocheng, and Y. Helong, "The implementation of parallel genetic algorithm based on MATLAB," *Proceedings of the 7th international conference on Advanced parallel processing technologies*, Guangzhou, China: Springer-Verlag, 2007, pp. 676-683.

[53] M. Deep and K. Deep, "A State-of-the-art Review of Population-based Parallel Meta-heuristics," 2009.

[54] C. Paz, "A Survey of Parallel Genetic Algorithms," 1997.

[55] A. Munawar, M. Wahib, M. Munetomo, and K. Akama, "A Sruvey: Genetic Algorithms and the Fast Evolving World," IEEE, 2008.

[56] A. Afsahi, "ELEC-873 Course Note on Scalability," 2009.

[57] "Box plot - MATLAB" Disponible: http://www.mathworks.com/help/toolbox/stats/boxplot.html.

[58] "Two-sample t-test - MATLAB" Disponible: http://www.mathworks.com/help/toolbox/stats/ttest2.html.

[59] B.T. Skinner, H.T. Nguyen, and D.K. Liu, "Performance Study of a Multi-Deme Parallel Genetic Algorithm with Adaptive Mutation," Palmerston North, New Zealand: 2004.

[60] J.W.S. Liu, *Real-Time Systems*, New Jersey: Prentice Hall, 2000.

[61] J.D.J. Anderson, *Introduction to Flight*, New York: McGraw-Hill, 2008.

[62] "Prioria Robotics Maveric - Wikipedia, the free encyclopedia" Disponible: http://en.wikipedia.org/wiki/Prioria_Robotics_Maveric.

[63] "Boeing ScanEagle - Wikipedia, the free encyclopedia" Disponible: http://en.wikipedia.org/wiki/Boeing_ScanEagle.

[64] "IAI Heron - Wikipedia, the free encyclopedia" Disponible: http://en.wikipedia.org/wiki/IAI_Heron.

[65] "BAE Systems Silver Fox - Wikipedia, the free encyclopedia" Disponible: http://en.wikipedia.org/wiki/Silver_Fox_UAV.

[66] M. Clerc, *Particle Swarm Optimization*, France: Lavoisier, 2005.

[67] "Google Maps" Disponible: http://maps.google.ca/.

[68] "GIS Mapping Software - GPS Mapping Software - Satellite Maps - Aerial Photos [Global Mapper]" Disponible: http://www.globalmapper.com/.

[69] G. Sanders and T. Ray, "Optimal offline path planning of a fixed wing unmanned aerial vehicle (UAV) using an evolutionary algorithm," Singapore: IEEE, 2007, pp. 4410 - 4416.

[70] F. Allaire, M. Tarbouchi, G. Labonté, and G. Fusina, "FPGA Implementation of Genetic Algorithm for UAV Real-Time Path Planning," *Journal of Intelligent & Robotic Systems*, vol. 54, 2009, pp. 495-510.

[71] J. Yu, Q. Zhang, V. Kroumov, S. Cheng, and Z. Zhang, "Path Planning Algorithm for Robot in 3D Environment Based on Neural Network," *Intelligent Robotics and Applications*, C. Xiong, H. Liu, Y. Huang, and Y. Xiong, eds., Springer Berlin / Heidelberg, 2008, pp. 1081-1088.

[72] M. Shashi and K. Deb, "Three-dimensional offline path planning for UAVs using multiobjective evolutionary algorithms," Singapore: IEEE, 2007.

[73] M. Bo-bo and G. Xiaoguang, "UAV Path Planning based on Bidirectional Sparse A* Search Algorithm," China: IEEE, 2010.

[74] I. Nikolos, E. Zografos, and A. Brintaki, "UAV Path Planning Using Evolutionary Algorithms," *Innovations in Intelligent Machines - 1*, Springer Berlin / Heidelberg, 2007, pp. 77-111.

[75] S. Li, S. Xiuxia, and X. Yuejian, "Particle Swarm Optimization for Route Planning of Unmanned Aerial Vehicles," Shandong: IEEE, 2006.

[76] C. Zhang, Z. Zhen, D. Wang, and M. Li, "UAV Path Planning Method Based on Ant Colony Optimization," 2010.

[77] "MicroPilot - Products - HORIZONmp" Disponible: http://www.micropilot.com/products-horizonmp.htm.

[78] D.B. Kirk and W.-mei W. Hwu, *Programming Massively Parallel Processors, A Hands-on Approach*, Burlington, MA: Elsevier, Morgan Kaufmann, 2010.

[79] "NVIDIA GeForce GTX 590" Disponible: http://www.geforce.com/#/Hardware/GPUs/geforce-gtx-590/specifications.

Annexe

Annexe

Code source

Le code source MATLAB du module de planification de trajectoires que nous avons développé pour cette recherche peut être téléchargé à l'adresse internet suivante :

http://cirl.dyndns.org/en/downloads/publications/52-vincent-roberge

Curriculum Vitae

Curriculum vitae

Nom : Vincent Rémi Roberge

Lieu de naissance : Joliette, Québec, Canada

Date de naissance : 7 août 1983

Cheminement académique :

Petit Séminaire de Québec,
Québec, QC, 1995-2000,
Diplôme d'étude secondaire.

Collège Militaire Royal du Canada (campus de St-Jean),
St-Jean-sur-Richelieu, QC,
2000-2001.

Collège Militaire Royal du Canada,
Kingston, ON, 2001-2005,
Diplôme B.Ing. *en génie informatique logiciel.*

Collège Militaire Royal du Canada,
Kingston, ON, 2009-2011,
Programme d'étude actuel.

Cheminement professionnel :

Officier de génie aéronautique dans les Forces canadiennes depuis 2000.

Cours élémentaire d'officier,
St-Jean-sur-Richelieu, QC, 2000.

Cours de base des officier de génie aéronautique,
Borden, ON, 2005-2006.

Officier de maintenance de 1re et 2e ligne pour les CF-188,
Cold Lake, AB, 2006-2009.