Planning for active mapping of crop fields using UAVs

Nicolas Holvoet

École Nationale de l'Aviation Civile

Toulouse, France

MÉMOIRE DE FIN D'ÉTUDES

Planning for active mapping of crop fields using UAVs

Auteur :
Nicolas HOLVOET

Encadrants :
Christophe REYMANN
Simon LACROIX
Daniel DELAHAYE

Stage de fin d'études réalisé en vue de l'obtention du diplôme d'ingénieur ENAC et du Master 2 Recherche Opérationnelle

au

Laboratoire d'Analyse et d'Architecture des Systèmes
Équipe Robotique et Interactions

28 janvier 2019

Abstract

As more and more farmers are turning to drones to monitor the health of their crops and to assist them in decision making, there is an increasing interest in efficient mapping algorithms. Nowadays, the coverage path planning problem for crop fields using unmanned aerial vehicles (UAVs) is often solved offline with a pre-planned trajectory and the acquisitions are then compiled at the end of the flight. Such a non-adaptive approach may yield, for some reasons, sub-optimal maps which do not cover the whole target area or with quality defects.

In this research, we propose a novel method based on metaheuristics to produce an optimized flight time trajectory to survey non-convex polygon areas in wind with a fixed-wing UAV. While offering better results than classical greedy approaches, this algorithm also proves to greatly reduce execution time. Thus, it can be used for online replanning tasks where the plan goals are deduced from a vision-based simultaneous localization and mapping (SLAM), which exhibits areas where additional data must be gathered. This work is developed and validated within a realistic simulation framework.

Keywords: aerial survey, coverage path planning, fixed-wing unmanned aerial vehicle, flight time minimization, online planning, metaheuristics, simultaneous localization and mapping

Résumé

Alors que les agriculteurs sont de plus en plus nombreux à recourir aux drones pour analyser l'évolution de leurs cultures et les aider dans leur prise de décision, le besoin d'algorithmes efficaces pour la cartographie se fait ressentir. Aujourd'hui, le problème de couverture d'une parcelle agricole à l'aide d'un drone est souvent résolu de manière hors ligne, avec un plan de vol pré-défini, et les acquisitions sont traitées et assemblées à l'issue du vol. Cette démarche non réactive peut produire des cartes qui, pour plusieurs raisons, ne recouvrent pas l'intégralité de la zone ou qui présentent des problèmes de qualité.

Dans ce rapport, nous présentons une nouvelle approche basée sur des métaheuristiques pour générer une trajectoire de durée optimisée permettant de cartographier à l'aide d'un drone à voilure fixe des champs polygonaux, non convexes, en présence de vent. Tout en offrant de meilleurs résultats que les méthodes gloutonnes classiques, cet algorithme permet de réduire considérablement les temps de calcul. Ainsi, nous pouvons l'utiliser dans des tâches de replanification en ligne où les décisions sont issues d'un module de localisation et cartographie simultanées (SLAM) qui détecte les zones où des données additionnelles doivent être collectées. Ce travail est développé et testé dans un environnement de simulation réaliste.

Remerciements

Je tiens à remercier mes encadrants de stage Christophe Reymann et Simon Lacroix pour leur disponibilité, leur sympathie et pour tout ce qu'ils ont pu m'apporter durant ces six mois dans l'environnement passionnant du LAAS.

Je remercie également l'ensemble des doctorants et stagiaires de l'équipe RIS qui ont contribué dans la bonne humeur à faire de ce séjour une expérience riche scientifiquement et humainement.

Enfin, je suis reconnaissant envers les équipes de l'ENAC pour ma formation et l'équipe du Master Recherche Opérationnelle qui m'a permis de développer un vif intérêt pour l'optimisation et la recherche appliquée. Je remercie plus particulièrement Daniel Delahaye pour ses conseils et la confiance qu'il m'a témoignée tout au long de ces deux dernières années.

Table des matières

Table des figures

Liste des abréviations

CPP	Coverage Path Planning
GIS	Geographic Information System
GTSP	Generalized Traveling Salesman Problem
SLAM	Simultaneous Localization and Mapping
TSP	Traveling Salesman Problem
UAV	Unmanned Aerial Vehicle

1 Présentation

1. Introduction

La croissance de la population mondiale et la crainte d'épisodes de famine, aujourd'hui accentuée par le réchauffement climatique, n'ont cessé de révolutionner l'agriculture. Les grands bouleversements ont été apportés par la mécanisation agricole au début du XX^{e} siècle. Les engrais chimiques, découverts dans les années 1840, se sont très vite répandus alors que les herbicides et insecticides ont attendu la Seconde Guerre Mondiale pour connaître une véritable industrialisation. Les organismes génétiquement modifiés, nés dans les années 1980 avec la découverte de la transgénèse, marqueront un nouveau tournant dans le processus de sélection et d'hybridation de cultures à haut potentiel de rendement. Alors que les politiques d'agriculture intensive se montrent par endroits à bout de souffle, la prochaine révolution agricole est peut-être en marche. L'hydroponie, ou agriculture hors-sol, ainsi que l'agriculture de précision sont pressenties pour la mener.

Le concept d'agriculture de précision a émergé aux États-Unis dans les années 1980 et est souvent résumé par la formule « la bonne dose, au bon endroit et au bon moment ». En se basant sur les connaissances apportées par l'agronomie, l'agriculture de précision se propose d'améliorer le rendement des parcelles tout en réduisant les impacts économiques et écologiques. Elle repose essentiellement sur l'imagerie aérienne, délivrée par satellite et de plus en plus par les drones, pour établir des diagnostics et des prévisions en analysant les données issues de différents capteurs : estimation du stress hydrique, de la biomasse, de la fertilisation, des maladies ou des mauvaises herbes, etc. Son but est d'aider les agriculteurs à ajuster leurs apports en intrants (semences, eau, engrais, produits phytosanitaires) en leur fournissant des recommandations détaillées, parfois même à l'échelle du plant (figures 1.1 et

1.2). Les progrès réalisés dans la perception et le traitement d'image (visible, infrarouge ou autre) ainsi que l'apprentissage artificiel ouvrent de nouvelles voies pour mieux comprendre le rendement agricole, influencé par un grand nombre de facteurs qui peuvent varier au sein même d'une parcelle.

Jusque-là peu accessible et réservée aux grandes exploitations, l'agriculture de précision entend bien se démocratiser avec l'essor des drones. De plus en plus d'entreprises de la *smart farming* – et parfois les constructeurs de drones eux-mêmes – proposent aujourd'hui des solutions clé en main destinées aux professionnels de l'agriculture. De la station sol permettant de planifier et de suivre la mission aux logiciels d'analyse de données et d'aide à la décision, l'ensemble du processus est sur le point d'être entièrement automatisable.

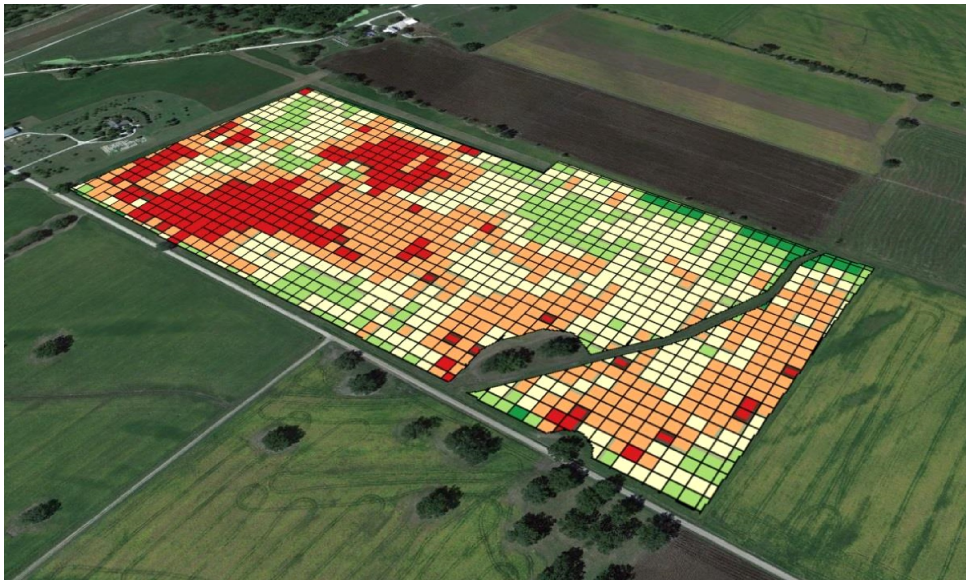


FIGURE 1.1 – Visualisation du stress hydrique d'une parcelle (Agribotix)

1.1 Contexte et motivation du stage

En 2015 est né le projet Precidrone, d'un partenariat entre le groupe coopératif agricole InVivo, trois de ses coopératives (Arterris, Terres du Sud et Ovalie Innovation), deux instituts techniques agricoles (Arvalis et Terres Inovia), deux instituts de recherche (INRA Avignon, LAAS-CNRS) et le constructeur de drones Delair. Sur une durée de 4 ans, son objectif est d'alimenter les modèles agronomiques pour les cultures de maïs, blé, tournesol et colza et de proposer aux agriculteurs des outils d'aide à la décision grâce aux drones et aux capteurs embarqués. Nous détaillons dans la suite la partie recherche menée au LAAS.

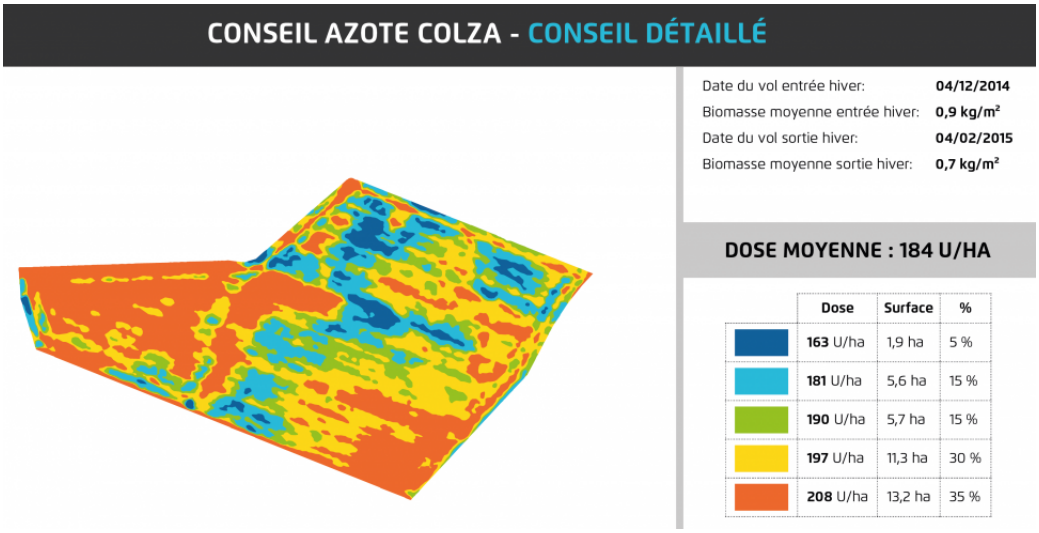


FIGURE 1.2 – Conseils d'apport en azote pour une culture de colza (Airinov)

Aujourd'hui, une mission de cartographie agricole se déroule classiquement en trois phases. La première consiste, pour l'opérateur, à préparer un plan de vol qui devra permettre de survoler et de photographier intégralement la zone. Ce plan de vol est ensuite envoyé au drone, qui pourra alors décoller et le suivre au mieux grâce à son autopilote, le vol constituant la deuxième phase. Enfin, dans un troisième temps, au retour du drone, les différentes prises de vue sont traitées et combinées pour construire une carte de l'environnement.

Malheureusement, il arrive que l'opérateur ne soit pas entièrement satisfait en constatant sur son écran que la carte ainsi produite comporte des défauts. Ceux-ci s'expliquent par divers aléas opérationnels et se traduisent par des zones non perçues ou des zones dont la qualité des clichés n'est pas suffisante pour permettre leur exploitation. En effet, il est possible qu'une zone n'ait pu être couverte lorsque le drone a dévié de sa trajectoire à la suite d'une bourrasque de vent, d'une erreur de capteurs et de positionnement, etc. Une mauvaise qualité d'acquisition peut, quant à elle, résulter d'un flou de bougé ou d'un changement soudain de conditions d'illumination, comme l'ombre d'un nuage. Si les ressources le lui permettent, l'opérateur doit alors planifier une nouvelle mission et faire redécoller le drone pour acquérir les données manquantes. Le projet de recherche mené au LAAS a pour but de palier ce problème à l'aide d'une boucle réactive de perception, décision et action adaptée au problème de cartographie d'une zone agricole par un drone.

Le premier objectif est de garantir une cartographie exhaustive et de qualité de la zone. En construisant la carte en temps réel par l'intégration continue des prises de vue, il devient possible d'analyser régulièrement au cours de la mission la carte obtenue jusqu'à présent dans

le but de corriger les défauts qui pourraient déjà être détectés et de prévenir les prochains. Ainsi, le drone devra être capable de prendre seul des décisions telles que la replanification d'une série d'acquisitions et le cas échéant de les intégrer dans son plan de vol, qui pourra alors être mis à jour en temps réel.

Le deuxième objectif est d'optimiser la durée de la mission, l'autonomie des batteries étant toujours, pour l'heure, un paramètre contraignant lors de la cartographie de grandes parcelles. Cette optimisation sera présente lors de la préparation du plan de vol initial mais aussi dans les airs lorsque le drone décidera de mettre à jour son plan de vol.

Au-delà de l'autonomie d'exécution offerte par les capteurs et l'autopilote, ce projet s'intéresse donc à apporter au drone une autonomie décisionnelle pour la cartographie d'environnements agricoles.

1.2 Objectifs

La sujet du stage porte plus précisément sur les algorithmes de planification, c'est-à-dire :

- la génération d'un plan de vol initial permettant de cartographier la zone en un minimum de temps (optimisation *hors ligne* qui se déroule avant le début de la mission) ;
- la mise à jour (ou réparation) en temps réel du plan de vol pour réagir aux aléas opérationnels et modifier si besoin certains paramètres de la mission, avec toujours pour objectif la minimisation de la durée (optimisation *en ligne* qui tire profit des connaissances acquises depuis le début de la mission).

2. Environnement

Ce projet de fin d'études se déroule au Laboratoire d'Analyse et d'Architecture des Systèmes (LAAS-CNRS) de Toulouse, spécialisé dans l'informatique, la robotique, l'automatique et les micro et nano-systèmes. Le stage a lieu dans l'équipe Robotique et InteractionS (RIS) qui a pour cœur de recherche les machines autonomes (bras robotisés, robots d'assistance à la personne, rovers planétaires, drones…) et leurs interactions (avec l'environnement physique, l'homme ou d'autres robots). À ces fins, l'équipe développe une certaine expertise couvrant toute la chaîne de perception, apprentissage et raisonnement, action et réaction.

Je travaille en collaboration avec Christophe Reymann, doctorant, et sous la supervision de Simon Lacroix, directeur de recherche.

2 Données et architecture

Nous présentons dans ce chapitre les prérequis et les données d'entrée ainsi que l'architecture de simulation dans laquelle s'inscrit ce travail.

1. Modèle dynamique du drone

1.1 Ailes volantes

Les drones à voilure fixe sont couramment utilisés pour les missions de cartographie, offrant un rayon d'action et une autonomie supérieure aux multirotors ainsi qu'une meilleure stabilité en présence de vent. Depuis le début des années 2010, de plus en plus d'industriels comme senseFly ou Delair (figure 2.1) se disputent le marché florissant des ailes volantes pour la cartographie professionnelle. Ces drones partagent des caractéristiques similaires : une envergure d'environ 1 m, une masse oscillant entre 1 et 1.5 kg, une autonomie de 60 min permettant de cartographier 200 ha à 120 m d'altitude, une vitesse de croisière de l'ordre de 15 m/s et une résistance au vent jusqu'à 12 m/s.

Bien que tous nos tests seront effectués en simulation, nous modéliserons le drone en nous inspirant des caractéristiques des ailes volantes actuellement commercialisées. La vitesse air v_a, considérée constante, sera fixée à 15 m/s. Le champ de vision de la caméra noté *FOV* (pour *field of view*) déterminera, avec l'altitude de vol h, la largeur w de la zone perçue au sol. L'altitude devra donc être choisie en fonction de la résolution souhaitée et nous la supposerons constante tout au long de la mission, ce qui serait en pratique un problème pour la cartographie de terrains en pente ou au relief irrégulier.

(A) eBee (senseFly) (B) UX11 (Delair)

FIGURE 2.1 – Exemples de drones à voilure fixe

Afin de pouvoir assembler les photographies lors de la construction de la carte, il est nécessaire que les clichés se recouvrent partiellement. Il existe un recouvrement frontal dit *overlap* qui détermine la fréquence de déclenchement du capteur lorsque le drone vole en ligne droite. Le recouvrement latéral dit *sidelap* correspond quant à lui au recouvrement nécessaire entre deux photographies voisines acquises sur des segments de trajectoire parallèles. Le *sidelap*, exprimé en pourcentage de la largeur de l'image, a une influence directe sur la génération du plan de vol car il détermine la distance *spacing* entre les segments d'observation (figure 2.2).

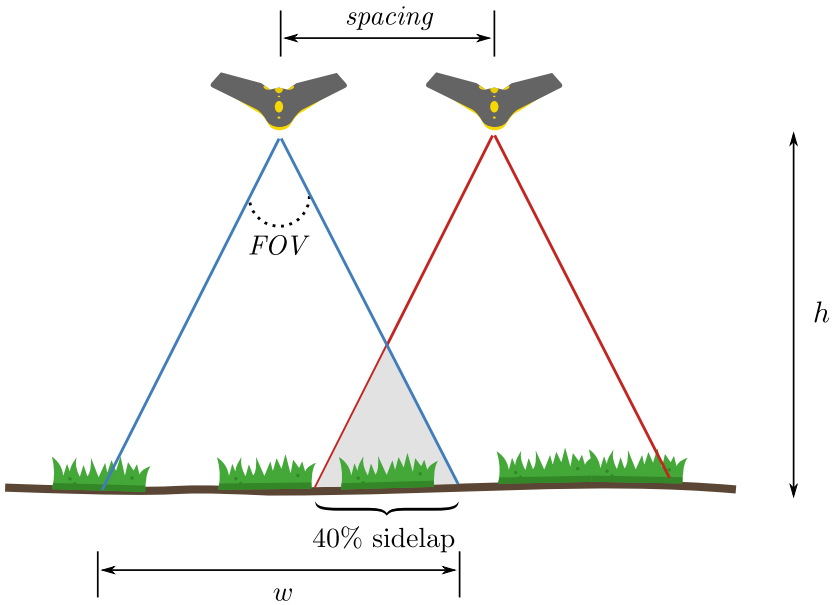


FIGURE 2.2 – Illustration du recouvrement latéral

Afin de pouvoir exploiter au mieux les photographies, les observations se feront uniquement au nadir et lorsque le drone sera à plat, en trajectoire rectiligne.

1.2 Trajectoires de Dubins

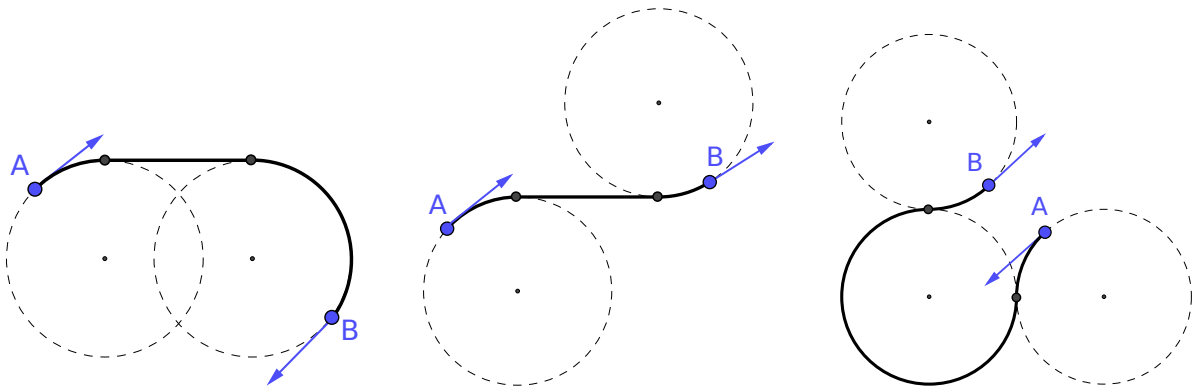
Contrairement aux multirotors qui peuvent soudainement changer d'attitude et qui sont donc des véhicules quasi holonomes, une aile volante est par nature non holonome. Elle dispose notamment d'un rayon de virage minimal qui contraint ses trajectoires.

Un véhicule qui évolue à vitesse constante et qui possède un rayon de virage minimal est appelé véhicule de Dubins, du fait d'un résultat très intéressant. Pour un tel véhicule, DUBINS, 1957 détermine la trajectoire de durée minimale permettant, à partir d'une position A et d'un cap donnés, de rejoindre un autre point orienté B. Il démontre que la trajectoire est constituée d'un enchaînement d'au plus trois mouvements primitifs qui sont :

- S, un déplacement en ligne droite sur une durée $t \geq 0$
- L, un virage par la gauche à rayon de virage minimal sur une durée $t \geq 0$
- R, un virage par la droite à rayon de virage minimal sur une durée $t \geq 0$

Plus précisément, la trajectoire optimale sera toujours de l'un de ces six types : LSL, RSR, RSL, LSR, LRL ou RLR. Cette notation symbolise l'enchaînement des actions. Par exemple, une trajectoire RSL correspond à un virage par la droite, à rayon de virage minimal sur une durée t_1, suivi d'un segment en ligne droite sur une durée t_2 et enfin complété par un virage par la gauche, à rayon de virage minimal sur une durée t_3. Grâce aux résultats de Dubins, il est très simple, étant donnés deux points orientés A et B, de déterminer le type de la trajectoire optimale et les durées associées.

La figure 2.3 illustre les trajectoires de durée minimale pour un véhicule de Dubins dans trois scénarios différents. Les cercles tangents aux virages sont représentés en pointillés.



(A) RSR (B) RSL (C) LRL

FIGURE 2.3 – Trois exemples de trajectoires de Dubins optimales (Wikimedia)

Dans la littérature, les drones à voilure fixe sont très souvent modélisés par un véhicule de Dubins. Les trajectoires optimales sont très faciles à calculer et, en première approximation, suffisamment réalistes bien qu'il ne soit pas possible en pratique de reproduire les changements soudains de rayon de virage, sans continuité.

2. Modélisation du champ

Dans ce travail, l'environnement est supposé suffisamment plat pour être représenté en 2D. Un champ est défini par un polygone, convexe ou concave, avec la seule contrainte que ses arêtes ne peuvent s'intersecter entre elles. À l'altitude où le drone volera, nous supposons qu'il n'y aura pas d'obstacles. Néanmoins, le champ peut contenir des « zones sans intérêt » qu'il n'est pas nécessaire de cartographier mais qui restent survolables ; il peut par exemple s'agir d'habitations, de bosquets ou d'un étang présents au milieu du champ. Ces zones, dont le nombre n'est pas limité, correspondent à des polygones, convexes ou concaves, qui seront vus comme des trous dans le polygone du champ (figure 2.4). Il nous arrivera de parler d'arêtes extérieures et d'arêtes intérieures pour différencier les arêtes du champ des arêtes dues aux zones sans intérêt.

FIGURE 2.4 – Un champ et sa modélisation par un polygone

Nous avons décidé d'encoder un tel polygone avec le format de géométrie utilisé habituellement pour représenter les informations géographiques dans les applications GIS (*Geographic Information System*). Il s'agit du format de fichier Shapefile, qui permet entre autres de représenter ce type de polygone, accompagné d'un fichier qui contient des informations sur

le système de coordonnées utilisé, renseignant le système géodésique (e.g. WGS 84) et la projection (e.g. latitude/longitude brutes, Lambert, UTM, etc.).

Cela nous permet d'utiliser des applications spécialisées dans la création, le traitement et la visualisation de données GIS, comme le logiciel QGIS. Nous l'utilisons pour dessiner les polygones des champs et pour visualiser sans peine le plan de vol généré, en superposant si besoin plusieurs couches d'information quelles que soient leur provenance et leur projection (cartes satellites issues de Google, photographies aériennes de l'Institut national de l'information géographique et forestière (IGN), fichiers Shapefile, etc.).

3. Localisation et cartographie simultanées (SLAM)

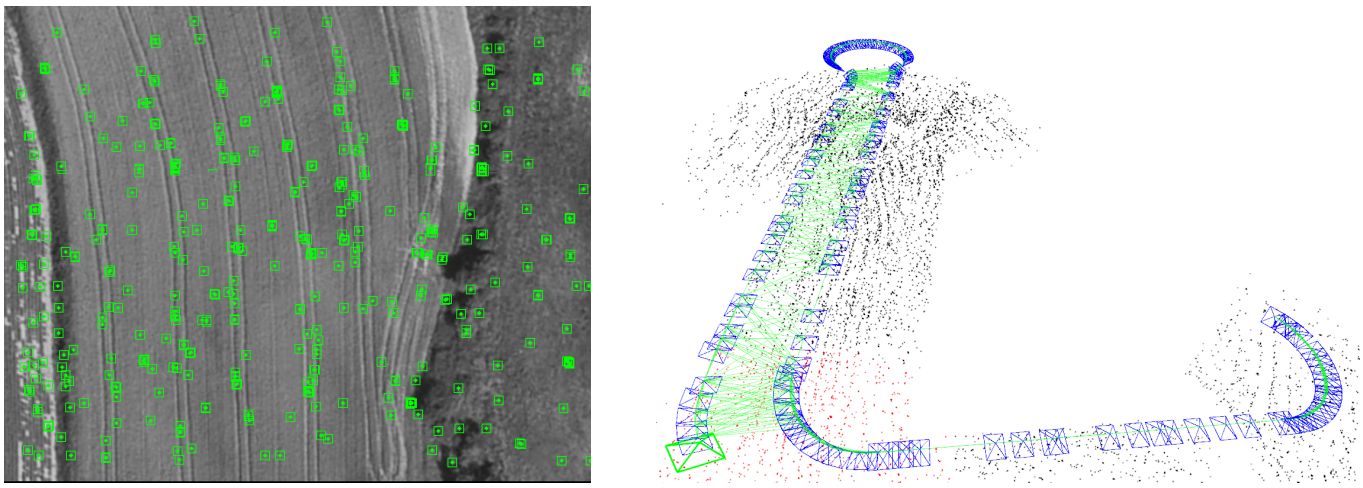
3.1 Présentation

Au cours d'une mission, un robot a sans cesse besoin de se localiser pour suivre une trajectoire prédéfinie, rejoindre un lieu précis, éviter certaines zones, géolocaliser ses acquisitions, etc. Pour estimer sa position, le robot peut utiliser les données de différents capteurs tels qu'un odomètre ou une centrale inertielle (localisation à l'estime). Ces données étant soumises à des erreurs cumulatives, il peut être nécessaire de les coupler à d'autres sources, cette fois externes, comme des balises ou un GPS. Néanmoins, selon l'environnement, ces sources ne sont pas toujours disponibles. Il reste alors la possibilité d'utiliser l'environnement pour estimer sa position en percevant et en identifiant des points d'intérêt ou amers (*landmarks* ou *features*). Pour un robot équipé d'un capteur optique dans le domaine du visible, un point d'intérêt peut par exemple correspondre à un changement de texture, de forme etc. que l'on peut facilement détecter. Connaissant l'orientation du capteur et donc la position relative du robot par rapport au point d'intérêt, il devient possible, si l'on dispose d'une carte de l'environnement où l'on peut identifier le point, de connaître la position absolue du robot dans l'environnement. De manière analogue, un humain perdu en zone urbaine, sans GPS mais avec un plan de la ville, peut estimer sa position en repérant autour de lui des points d'intérêt qu'il localisera sur le plan, l'incertitude sur sa position étant réduite par la quantité et la qualité des points d'intérêt qu'il parvient à distinguer.

Que faire lorsque l'on ne dispose pas a priori d'une telle carte de l'environnement ? Il est tout à fait possible de demander à un robot de construire incrémentalement une carte « mentale » de son environnement en le laissant explorer la zone… à condition qu'il sache se localiser précisément afin de pouvoir assembler correctement ses différentes perceptions du monde, ce qui nous ramène au problème initial de la localisation. Face à ce paradoxe (la localisation

nécessite une carte et construire une carte nécessite la localisation), on comprend que les problèmes de localisation et de cartographie ne peuvent être traités indépendamment et le SLAM (*simultaneous localization and mapping*) a pour objectif de les résoudre simultanément. Il s'agit d'un champ de recherche très actif en robotique depuis la fin des années 1980 car il conditionne en partie le développement de systèmes réellement autonomes. De multiples approches différentes ont été développées pour le résoudre.

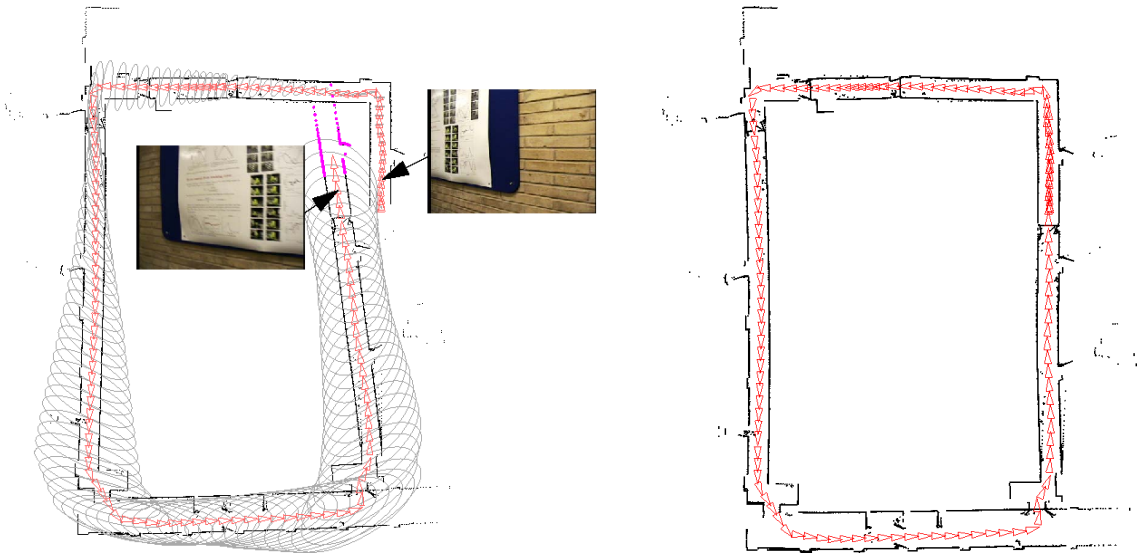
Dans le cas du SLAM visuel où le capteur embarqué est une caméra, le robot est mis en mouvement et l'idée générale consiste à identifier et à suivre un ensemble de points d'intérêt fixes (figure 2.5a) dans des observations successives. Seule la position de départ étant connue, la position des points d'intérêt ne peut qu'être estimée par triangulation en même temps que l'on cherche les poses (position et orientation) de la caméra qui expliquent au mieux les différentes observations. Il s'agit d'un problème d'optimisation qui repose essentiellement sur les statistiques bayésiennes et dont l'objectif est de minimiser l'erreur de reprojection lors de l'estimation simultanée des poses de la caméra et des positions des points d'intérêt. On peut modéliser le problème sous la forme d'un graphe (*pose graph*) où les nœuds correspondent aux poses successives estimées de la caméra ; deux poses seront connectées et dites contraintes si les observations partagent au moins un point d'intérêt commun. Un tel graphe est illustré par la figure 2.5b, où les points d'intérêt correspondent aux points noirs et rouges et où les poses de la caméra sont représentées en bleu et connectées en vert lorsqu'elles sont contraintes. Intuitivement, plus le graphe comporte de connexions, plus le problème est contraint et plus l'incertitude sur la trajectoire et la carte sera réduite.



(A) Points d'intérêt (B) Graphe des poses

FIGURE 2.5 – Utilisation du SLAM pour la cartographie agricole

Chaque nouvelle observation vient alimenter ce problème d'optimisation et peut potentiellement permettre de corriger la carte en réestimant la position des points d'intérêt et les poses de la caméra, tout en introduisant de nouveaux points d'intérêt. Or le SLAM doit offrir des performances compatibles avec la vitesse du robot et la fréquence des observations. Le problème d'optimisation sous-jacent devenant de plus en plus grand au cours de la mission, on le limite généralement à un sous-ensemble d'observations relativement proches de la dernière observation. Ces optimisations locales et incrémentales peuvent entraîner une dérive du SLAM et une erreur sur l'estimation de la position. On peut alors de manière ponctuelle relancer l'optimisation globale sur l'ensemble des observations. Une dérive peut également s'expliquer par un manque de données (faible densité de points d'intérêt, peu d'observations…) ou la forme de la trajectoire qui ne permet pas de contraindre suffisamment le graphe des poses. Après un long parcours, il devient alors nécessaire de détecter des éventuelles *fermetures de boucle* lorsque le robot perçoit de nouveau des points d'intérêt qu'il avait perçus dans le passé et forcer une nouvelle optimisation de la trajectoire et de la carte. La figure 2.6a illustre les poses successives, en rouge, et l'incertitude associée, en gris, ainsi que la carte à un moment donné de la mission. On détecte alors que le robot perçoit un point d'intérêt déjà perçu au début de la mission, ce qui nous permet de fermer la boucle et d'améliorer la robustesse du SLAM (figure 2.6b). La recherche actuelle sur le SLAM s'intéresse notamment à mieux identifier voire à provoquer ce type de scénarios en guidant de temps à autre le robot vers une zone susceptible de mieux contraindre le problème. Reconnaître que deux points d'intérêt sont identiques est l'une des principales difficultés.



(A) Avant (B) Après

FIGURE 2.6 – Exemple d'une fermeture de boucle (NEWMAN et HO, 2005)

3.2 Apport pour la planification en ligne

L'utilisation du SLAM pour la cartographie agricole par drone, dans le cadre du projet Precidrone, fait l'objet de la thèse de Christophe Reymann. Le SLAM permettra notamment de créer des requêtes ou « buts » pour la planification en ligne.

En mettant à jour les estimations de trajectoire et la carte en temps réel, il est possible d'estimer qu'une zone n'a pas été perçue, si le drone a dévié de sa trajectoire, ou a été mal perçue, pour diverses raisons. Le premier but consiste donc à demander au module de planification de (re)visiter une zone.

Entraîner un modèle pour prédire la densité de points d'intérêt dans le champ devrait à terme permettre d'apprendre une valeur de recouvrement *sidelap* optimale pour garantir un bon assemblage de la carte. De nos jours, les opérateurs de drones ont tendance à surestimer le recouvrement nécessaire, ce qui allonge la durée de la mission (segments de trajectoire plus rapprochés). Modifier l'espacement entre les segments d'observation forme un deuxième but.

Enfin, l'analyse du graphe des poses devrait permettre de trouver un chemin capable de provoquer des fermetures de boucle et donc d'améliorer au besoin la robustesse de la carte. Planifier un tel chemin constitue un troisième but.

4. Architecture

Ce travail s'insère dans une plate-forme logicielle de simulation robotique construite autour de ROS (*Robot Operating System*) permettant de simuler une mission de cartographie. ROS est un *middleware* destiné à agréger différents systèmes, capteurs et applications hétérogènes (nœuds) avec une couche d'abstraction, simplifiant notamment la communication entre les nœuds.

Dans notre cas, nous pouvons identifier trois nœuds :

- Un module de SLAM et de planification. Le SLAM maintient une carte de l'environnement et les buts de la planification sont issus de l'analyse de cette carte comme expliqué dans le paragraphe précédent.
- Un module simulant le drone et son autopilote, éventuellement perturbé par du vent, des turbulences, etc. À terme, il pourrait par exemple s'agir de Paparazzi.
- Un module simulant l'environnement et la caméra.

Ce dernier nœud utilise MORSE, un projet initié au LAAS basé sur le moteur de jeu intégré de Blender. L'environnement est représenté par une scène 3D et MORSE permet de simuler à l'intérieur de celle-ci une grande variété de capteurs comme une caméra. Ainsi, en précisant la pose (position et orientation) de la caméra, MORSE est capable de « prendre » une photographie de la scène.

L'architecture et les échanges de données sont illustrés sur la figure 2.7.

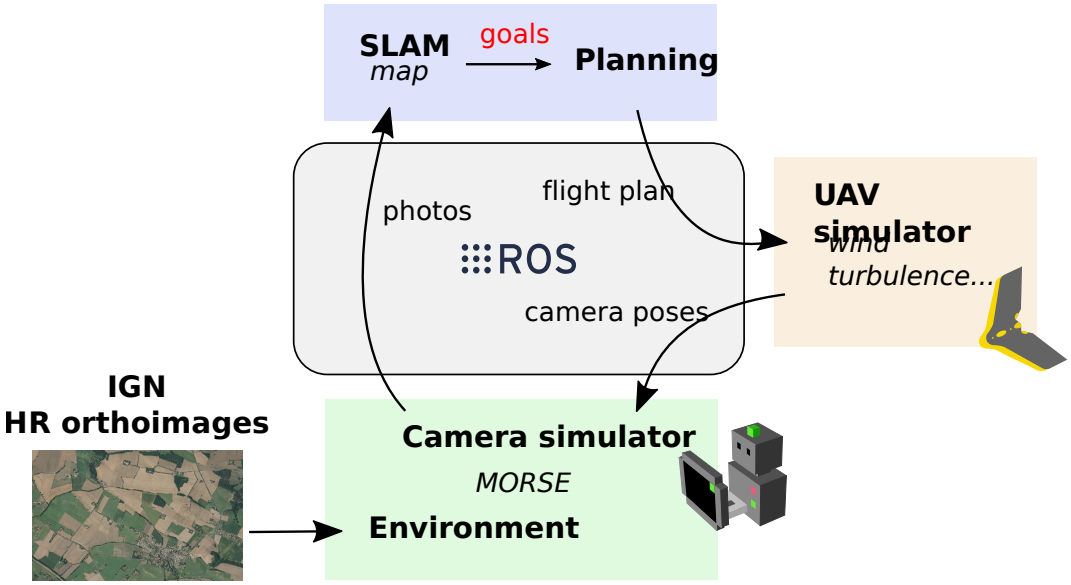


FIGURE 2.7 – Architecture de la plate-forme de simulation robotique

L'environnement correspond à une scène 2D tapissée d'orthophotographies aériennes haute résolution obtenues auprès de l'IGN. Le module de planification envoie un plan de vol au simulateur de drone qui, simulant son déplacement, enverra à intervalles réguliers la pose de la caméra à MORSE. Celui-ci pourra alors extraire des images IGN (environnement) une photo de ce que le drone est censé voir avec cette pose. Le module du SLAM utilisera ces photos pour alimenter sa carte et ses modèles et sera en mesure d'envoyer de nouveaux buts à la planification et ainsi de suite.

3 État de l'art

1. Problème du *Coverage Path Planning*

Le *Coverage Path Planning* (CPP) est un problème qui consiste à déterminer un chemin permettant de couvrir intégralement une zone de l'environnement. La notion de couverture s'entend au regard de la portée et de la cinématique des outils ou capteurs embarqués : une tondeuse à gazon devra effectivement « passer partout », alors que la cartographie aérienne prendra en compte l'altitude et le champ de vue de la caméra pour définir la zone perçue au sol.

Omniprésent en robotique, le CPP est notamment utilisé par les robots terrestres pour l'aspiration de sols, l'exploration de bâtiments ou la détection de mines ; par les robots sous-marins (AUV, *Autonomous Underwater Vehicle*) pour la cartographie du fond marin, la recherche d'épaves ou l'exploration de réacteurs nucléaires sinistrés ; par les robots aériens ou drones (UAV, *Unmanned Aerial Vehicle*) pour les relevés topographiques, la météorologie ou la cartographie. L'industrie a aussi largement participé à l'essor de la robotique et du CPP avec l'usinage de pièces, les robots de peinture ou aujourd'hui les imprimantes 3D. Enfin, les robots chirurgicaux s'invitent de plus en plus dans les salles d'opération, le CPP pouvant par exemple être utilisé pour l'exploration et l'ablation de tissus tumoraux.

Le problème du CPP étant exprimé en termes très génériques, il regroupe en réalité plusieurs dizaines de variantes propres au champ d'application. Ainsi, si les publications sur le CPP sont extrêmement nombreuses, il convient de les distinguer selon plusieurs critères plus ou moins contraignants, parmi lesquels (en italique, les spécificités de notre problème) :

- environnement connu ou inconnu : *connu*

- présence d'obstacles infranchissables : *non*
- présence de zones sans intérêt mais franchissables : *oui*
- formes de l'environnement (convexité et concavité) et relief : *toute forme acceptée, 2D*
- nombre de robots collaborant à la tâche : *un seul*
- dynamique du ou des robot(s) et contraintes associées : *véhicule de Dubins*
- prise en compte ou non du vent, des courants… : *prise en compte du vent*
- avec ou sans optimisation (en temps, énergie, matériau…) : *optimisation de la durée*
- avec ou sans replanification en temps réel au cours de la mission : *avec replanification*
- contraintes de performances de l'algorithme : *performances adaptées à la replanification en ligne*

2. Planification hors ligne

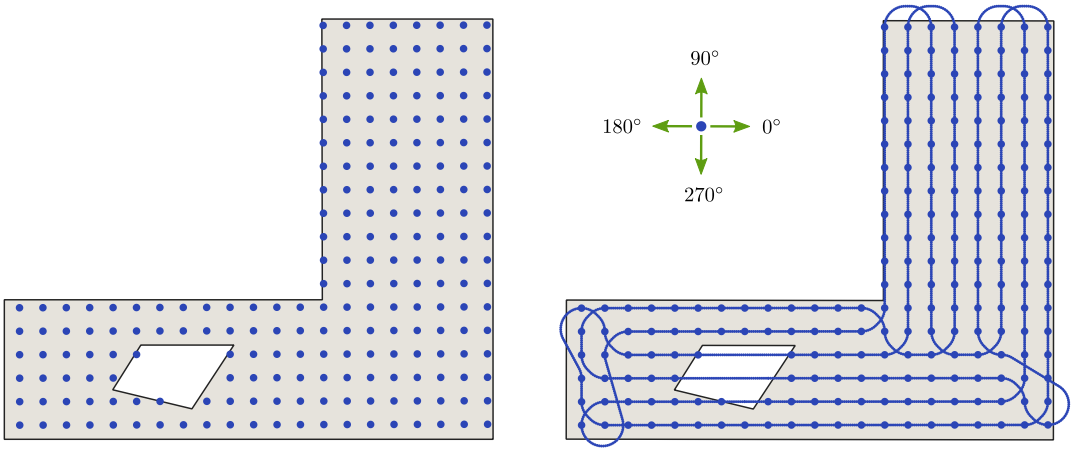
Nous nous intéressons ici au problème de génération hors ligne de plan de vol, en tant que CPP. La taxonomie du CPP qui est aujourd'hui largement adoptée distingue les méthodes de résolution par *décomposition approximative* et par *décomposition exacte*. Nous reprenons ici cette classification comme l'ont fait par exmple GALCERAN et CARRERAS, 2013 dans un état de l'art sur le CPP dédié à la robotique.

2.1 Décomposition approximative

Les méthodes de décomposition approximative travaillent sur une représentation simplifiée de l'environnement où la zone à parcourir est approximée par un ensemble de sous-zones à la structure géométrique simple (points, carrés, hexagones, treemap…).

Par exemple, la figure 3.1a illustre l'approximation d'un champ par une grille de points régulièrement espacés, l'espacement étant ici imposé par le recouvrement souhaité entre deux acquisitions voisines. Le problème consiste alors à déterminer la trajectoire optimale permettant de visiter tous les points et peut être résolu par des algorithmes à front d'onde (NAM et al., 2016), des algorithmes génétiques (YAKOUBI et LASKRI, 2016), ou encore des variantes du problème du voyageur de commerce (TSP) adaptées au modèle dynamique du véhicule de Dubins, dites « Dubins-TSP ». Pour prendre en compte le degré de liberté apporté par le cap du drone, NY et al., 2012 résolvent un problème du voyageur de commerce généralisé (GTSP). Sur la figure 3.1b est représentée la trajectoire de durée minimale lorsque le cap

du drone est limité à quatre valeurs (discrétisation avec un pas de 90°). Cette approche peut sembler intéressante à première vue car avec une connaissance très limitée du problème, elle parvient à générer un plan de vol qui présente une certaine structure reflétant l'organisation du champ, avec de surcroît un gage d'optimalité.



(A) Discrétisation (B) Trajectoire optimale

FIGURE 3.1 – Application du Dubins-TSP sur un champ

Cependant, cet exemple est très particulier : la grille ainsi que les valeurs de cap autorisées pour le drone sont parfaitement alignées avec les directions principales du champ. La figure 3.2 illustre dans les mêmes conditions la trajectoire obtenue pour un autre champ. Cette fois, ni la disposition des points, ni les valeurs de cap autorisées ne permettent de structurer la trajectoire comme l'intuition le voudrait. Notons d'ailleurs que la trajectoire contient de nombreux virages à l'intérieur du champ, ce que l'on souhaiterait éviter.

Ainsi, pour notre utilisation, ces approches ne sont pas suffisamment flexibles pour s'adapter facilement à toute forme de champ. De plus, le passage à l'échelle est délicat car l'ajout d'un point ou la diminution du pas de discrétisation pour le cap entraîne une explosion de la combinatoire. Sur des champs plus étendus, la résolution d'un tel problème par un Dubins-TSP peut prendre plusieurs heures.

2.2 *Décomposition exacte*

Les méthodes de recouvrement par décomposition exacte découpent l'environnement en plusieurs sous-zones ou cellules qui forment une partition de la zone à parcourir. Chaque cellule est ensuite parcourue avec un motif particulier.

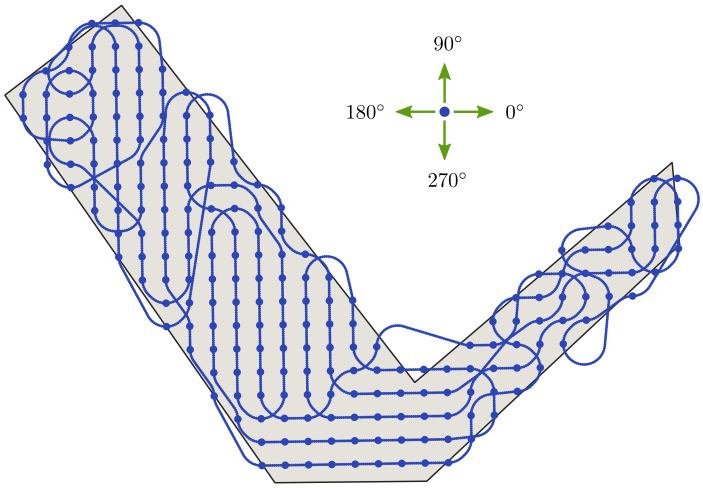


FIGURE 3.2 – Application du Dubins-TSP sur un autre champ

2.2.1 *Décomposition de boustrophedon*

Il existe deux motifs de prédilection lorsque l'on souhaite couvrir une zone. Le premier consiste à suivre les contours de la zone pour la parcourir avec un chemin en forme de spirale (figure 3.3). Ce schéma n'est pas adapté pour un drone à voilure fixe, les virages étant nombreux et de plus en plus resserrés.

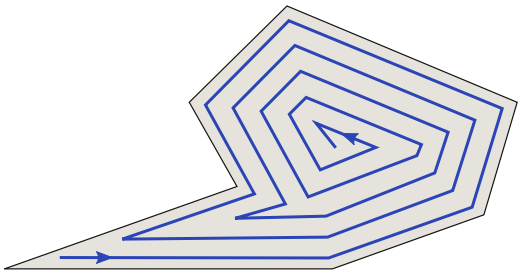


FIGURE 3.3 – Couverture par motif de spirale

Le deuxième motif consiste à effectuer des allers-retours sur des segments parallèles, dans une direction donnée, comme l'illustre la figure 3.4. Ce motif est appelé « boustrophedon », étymologiquement en référence à la trajectoire du bœuf qui parcourt un champ lors de travaux agricoles. Comme le montre la figure, il existe une orientation préférentielle qui minimise le nombre de segments et donc de demi-tours nécessaires pour parcourir l'intégralité de la zone.

Pour parcourir une zone complexe (concavités, obstacles, …) avec un tel motif et « sans lever le crayon », il peut être nécessaire de décomposer la zone. CHOSET et PIGNON, 1998 ont

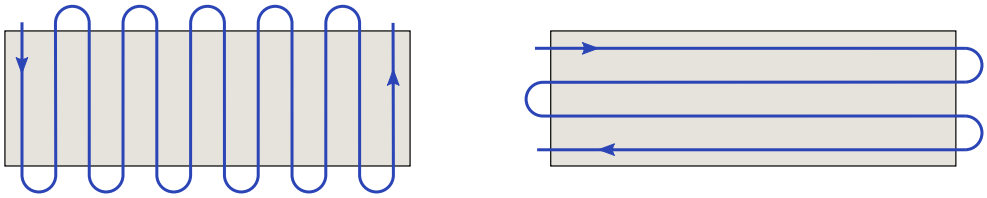
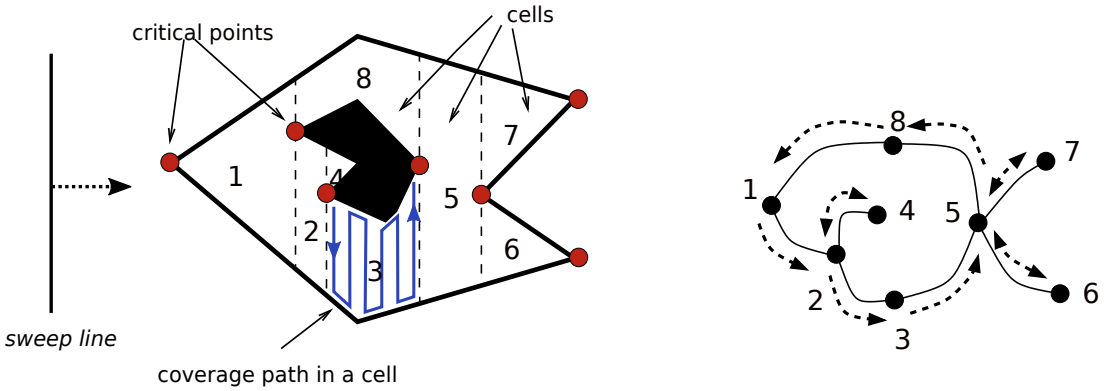


FIGURE 3.4 – Couverture par motif de boustrophedon

introduit un algorithme qui balaie la zone à l'aide d'une *sweep line* (orientée dans la direction de parcours) détectant les points de l'environnement qui introduisent un changement de convexité dans la direction de balayage (points critiques). Ces points critiques permettent alors de délimiter des cellules qui seront « convexes dans la direction » de parcours et qui pourront donc être individuellement couvertes par des motifs de boustrophedon (figure 3.5a). Pour couvrir l'ensemble de la zone, il est nécessaire de couvrir individuellement chaque cellule. On peut alors s'aider du graphe d'adjacence des cellules pour déterminer une séquence de visite optimale des cellules (figure 3.5b).



(A) Création des cellules

(B) Graphe d'adjacence

FIGURE 3.5 – Décomposition de boustrophedon d'une zone contenant un obstacle (inspiré de ACAR et al., 2002)

Bien que très courante en robotique et parfois appliquée aux drones pour l'agriculture (XU et al., 2014), cette approche n'est pas adaptée à notre problème car couvrir la zone dans une unique direction n'est pas optimal lorsque les virages sont coûteux en temps et en énergie.

Le motif de boustrophedon reste néanmoins tout à fait pertinent car il s'agit de la manière la plus naturelle de parcourir une zone. C'est d'ailleurs le motif utilisé par la majorité des agriculteurs pour entretenir leur champ à l'aide d'un tracteur. Cependant, un tracteur possède un rayon de virage minimal et les demi-tours sont toujours coûteux en temps. Pour cette raison, on constate que les agriculteurs adaptent l'orientation de la trajectoire en fonction de la

géométrie de leur champ comme le montre la figure 3.6. On rencontre la même problématique avec les drones à voilure fixe et les travaux les plus récents cherchent donc à analyser la forme du champ pour en déterminer une décomposition « optimale », permettant de couvrir chaque cellule dans une direction spécifique qui reflète localement la forme du champ.

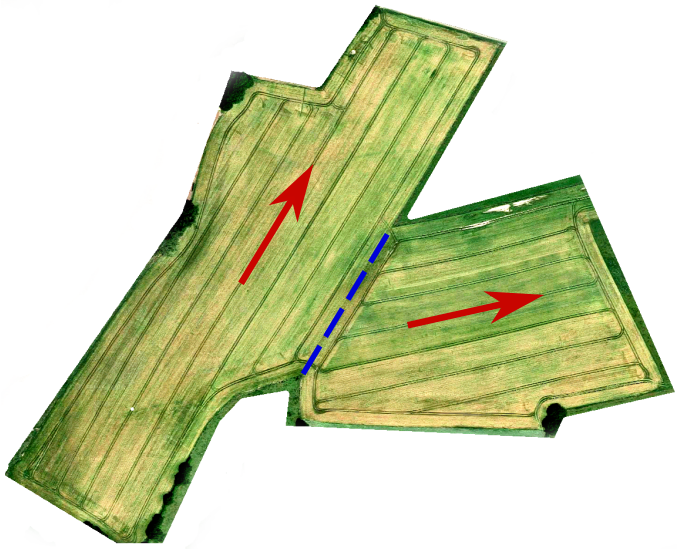


FIGURE 3.6 – Motifs de boustrophedon réalisés par un tracteur

LI et al., 2011 déterminent de manière gloutonne une décomposition convexe qui réduise globalement le nombre de segments (et donc de demi-tours) nécessaires pour couvrir l'ensemble des cellules (figure 3.7).

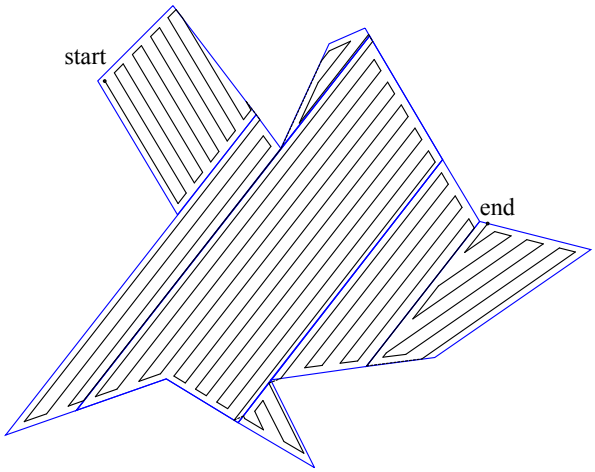


FIGURE 3.7 – Décomposition convexe et gloutonne (LI et al., 2011)

HUANG, 2001 cherche par programmation dynamique une décomposition optimale du

champ qui minimise là aussi le nombre de segments. À partir d'une décomposition initiale, la méthode consiste à explorer toutes les recombinaisons possibles par fusion de cellules voisines (figure 3.8). Cette recherche exhaustive à la complexité exponentielle présente des performances qui se dégradent très rapidement, en fonction du nombre d'arêtes du champ (une dizaine de minutes à plusieurs heures de calcul).

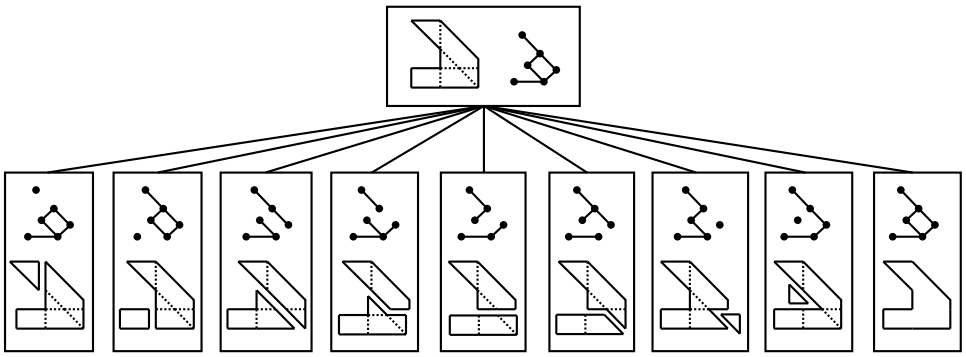


FIGURE 3.8 – Premier niveau d'une exploration exhaustive par programmation dynamique (HUANG, 2001)

JIN et TANG, 2010 proposent une approche intéressante qui consiste non pas à minimiser le nombre de segments mais la durée des demi-tours. En effet, ils constatent que la durée d'un demi-tour n'est pas la même sur chaque arête pour un véhicule de Dubins (un tracteur, dans leur cas), puisque la forme du demi-tour dépend de la pente de l'arête et de l'orientation des segments. La décomposition optimale, qui tient compte d'éventuels obstacles, est déterminée de manière récursive et exhaustive en utilisant un jeu d'arêtes générées préalablement (figure 3.9). La méthode de résolution souffre de problèmes de performance au-delà d'un certain nombre d'arêtes.

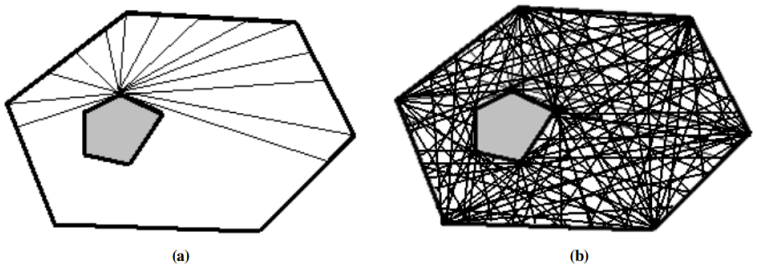


FIGURE 3.9 – Construction du graphe de recherche (b) en traçant de nouvelles arêtes à partir de chaque sommet (a) (JIN et TANG, 2010)

COOMBES et al., 2018 s'intéressent au CPP pour un drone à voilure fixe en présence de vent. La durée totale de la mission est estimée et constitue le critère d'optimisation. Les décompositions sont une à une construites et évaluées de manière analogue à HUANG, 2001,

à l'exception que les cellules obtenues sont toutes convexes. Les auteurs notent à leur tour que les performances de la résolution par programmation dynamique ne permettent pas d'utiliser leur algorithme sur le terrain. La figure 3.10 donne un exemple de résultat.

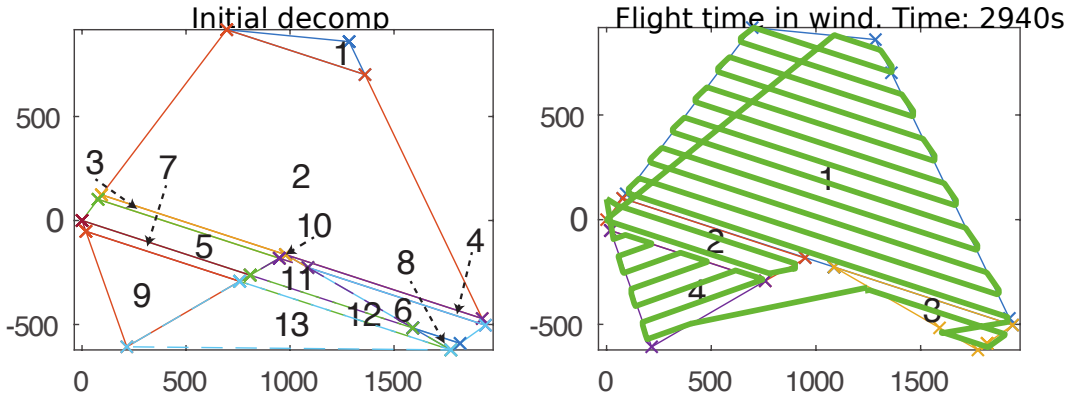


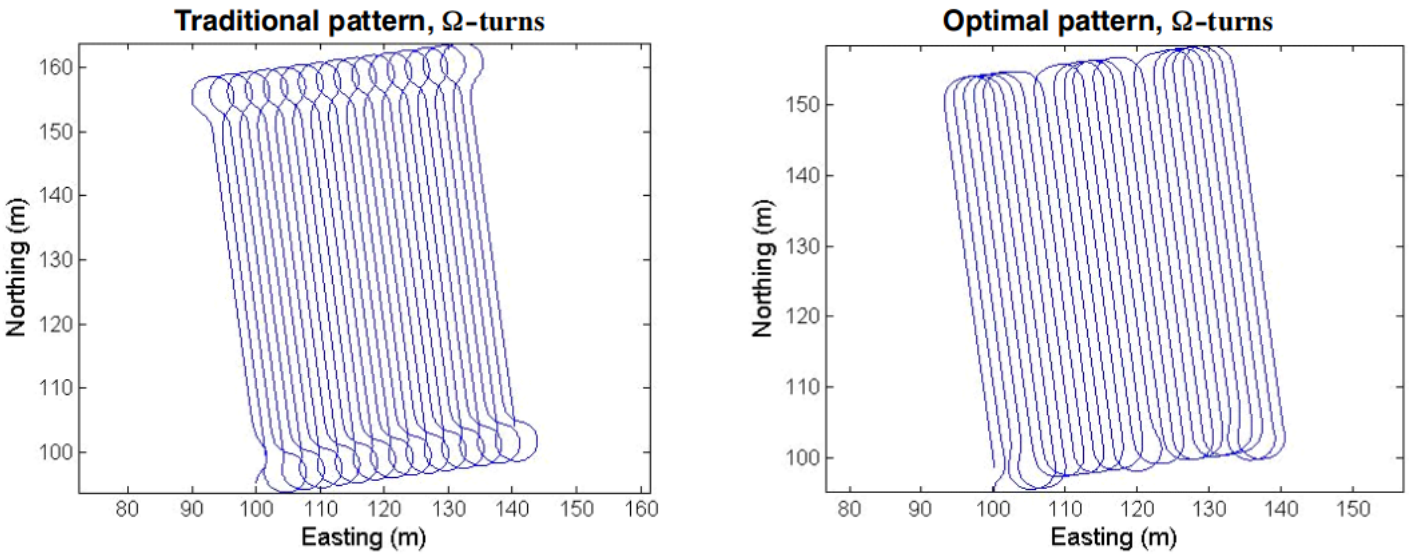
FIGURE 3.10 – Exemple de résultat issu de COOMBES et al., 2018

Toutes ces approches restent fidèles à l'idée de base du CPP par motifs de boustrophedon tel qu'introduit par CHOSET et PIGNON, 1998. En effet, une fois la décomposition du champ obtenue (quels que soient la méthode et le critère d'optimisation), le principe reste identique : chaque cellule est parcourue individuellement par des motifs de boustrophedon avant d'être « connectée » à la cellule qui doit être couverte ensuite.

2.2.2 *Optimisation des transitions*

Lorsque l'on diminue l'espacement entre les segments, il se peut, selon le rayon de virage minimal du véhicule, qu'il devienne très long de rejoindre le segment voisin après avoir parcouru un segment, comme c'est traditionnellement le cas dans les motifs de boustrophedon. BOCHTIS et al., 2009 montrent, pour un tracteur modélisé par un véhicule de Dubins, qu'il peut être plus judicieux de connecter un segment non pas à son voisin mais à un autre segment, plus facilement accessible en considérant le rayon de virage, en « enjambant » un ou plusieurs segments. À condition, bien sûr, que tous les segments aient été parcourus à la fin de la mission. La figure 3.11a illustre la trajectoire obtenue dans le cas d'une séquence de boustrophedon classique où chaque segment est connecté à son segment voisin. L'espacement des segments est tel que le demi-tour, en forme de bulbe, est très long. La figure 3.11b donne la trajectoire obtenue lorsque l'ordre de parcours des segments a été optimisé de manière à réduire le durée de la mission. Dans cet article, la combinaison optimale qui permet de parcourir tous les segments est déterminée en résolvant un problème de tournée de véhicule.

YU et al., 2015 modéliseront de manière équivalente le problème sous la forme d'un problème du voyageur de commerce généralisé.



(A) Séquence de boustrophedon

(B) Séquence optimale, en temps

FIGURE 3.11 – Influence de l'ordre de parcours des segments sur la durée de la mission (BOCHTIS et al., 2009)

Cette idée ne se limite pas uniquement à trouver la séquence optimale en temps qui permettrait de connecter des segments parallèles. Elle peut en effet être utilisée pour déterminer la trajectoire optimale en temps qui connecte un ensemble de segments issus de différentes cellules et avec des orientations différentes. En laissant l'algorithme déterminer la séquence de parcours des segments, il n'y a alors plus aucune raison de parcourir tous les segments d'une cellule avant de passer à la cellule suivante et il n'est plus nécessaire de définir des points d'entrée et de sortie dans les cellules, qui pouvaient engendrer des transitions coûteuses (comme on peut en voir sur l'exemple de la figure 3.10). À notre connaissance, les publications sur le CPP qui intégrent cette idée se font encore rares. Citons BOCHKAREV et SMITH, 2016 qui proposent une méthode en deux étapes pour parcourir une zone potentiellement concave et avec des obstacles. Ils commencent par déterminer une décomposition exacte du champ en tentant de minimiser le nombre de segments, par une approche gloutonne. Ensuite, les segments des différentes cellules sont connectés de manière à optimiser la trajectoire finale avec l'idée que nous venons de présenter, ce qui permet de maîtriser la longueur des transitions (figure 3.12).

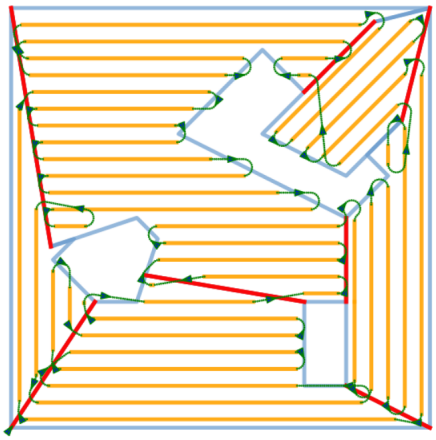


FIGURE 3.12 – Ordre de parcours des segments optimisé (BOCHKAREV et SMITH, 2016)

3. Planification en ligne

Les publications qui traitent de la planification en ligne pour le CPP à l'aide d'un drone sont peu nombreuses.

PAULL et al., 2014 développent un modèle basé sur la théorie de l'information. Tout au long du vol, une carte indiquant les zones perçues est mise à jour et à chaque instant, le drone choisit l'orientation qui maximise le gain d'information à l'aide de la carte et d'un modèle prédictif de caméra. La figure 3.13 illustre la trajectoire obtenue lors d'une expérimentation. Bien que la caméra soit montée sur un cardan à deux axes dans leur article, la trajectoire est trop irrégulière pour nous permettre d'acquérir des photographies de bonne qualité. Par ailleurs, les auteurs ne cherchent pas à minimiser la durée de la mission mais uniquement à garantir la cartographie exhaustive du champ.

L'intégration du SLAM dans le problème du CPP d'un environnement connu (et donc avec une trajectoire initiale) reste encore peu exploitée dans la littérature. Certains articles s'intéressent au problème de la fermeture de boucle présenté précédemment, qui consiste, une fois la trajectoire nominale parcourue, à déterminer un chemin qui permette d'augmenter la robustesse de la carte en contraignant un maximum d'observations dans le graphe des poses. Dans le cadre de l'exploration d'une coque de bateau à l'aide de véhicules sous-marins autonomes, CHAVES et al., 2016 développent un modèle permettant de prédire la densité de points d'intérêt dans une zone donnée (saillance), utilisé ensuite pour trouver un chemin intéressant pour fermer la boucle (figure 3.14).

FIGURE 3.13 – Exemple de trajectoire calculée en ligne par PAULL et al., 2014

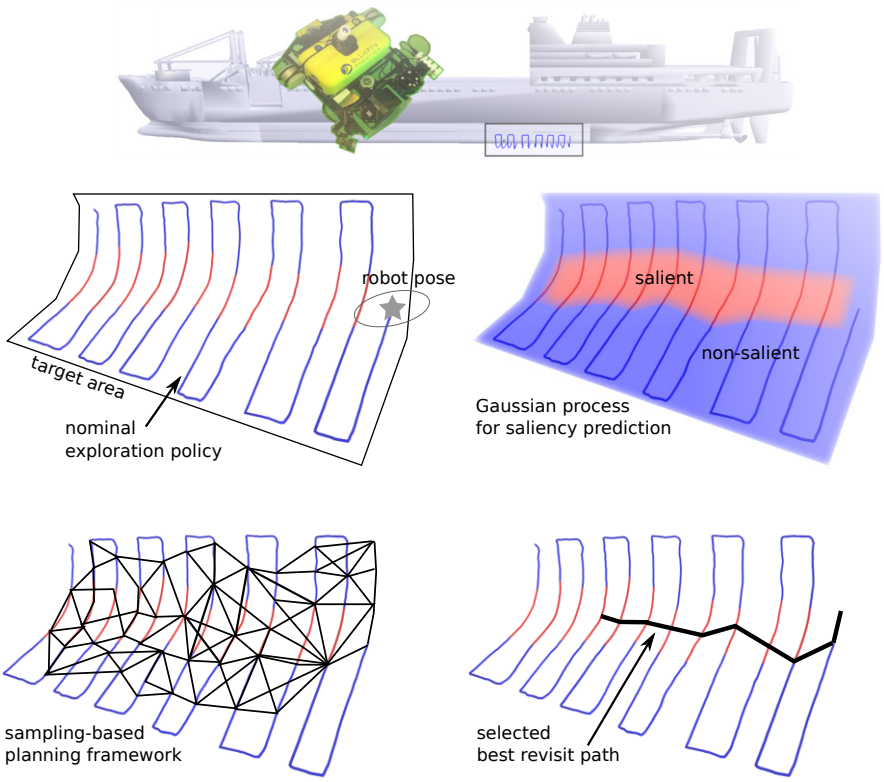


FIGURE 3.14 – Détermination d'un chemin permettant de fermer la boucle du SLAM pour réduire l'incertitude de la carte (CHAVES et al., 2016)

4. Cas du vent

Les chemins de Dubins nous permettent de déterminer la trajectoire de durée minimale permettant, à partir d'une position et d'un cap donnés, de rejoindre un autre point orienté. Le véhicule est supposé posséder une vitesse air constante et un rayon de virage minimal. La trajectoire obtenue est alors composée de tronçons de virage L ou R, effectués à rayon de virage minimal, et de tronçons S, segments en ligne droite. Que se passe-t-il en présence d'un vent uniforme et constant ?

G MCGEE et al., 2005 traitent la question comme un problème de rendez-vous entre le drone et une cible virtuelle, correspondant au point d'arrivée mis en mouvement avec la même vitesse que le vent mais dans la direction opposée. Le but est alors de trouver dans le référentiel air un chemin de Dubins qui intercepte, en lieu et en temps, la trajectoire de cette cible virtuelle, par une méthode itérative.

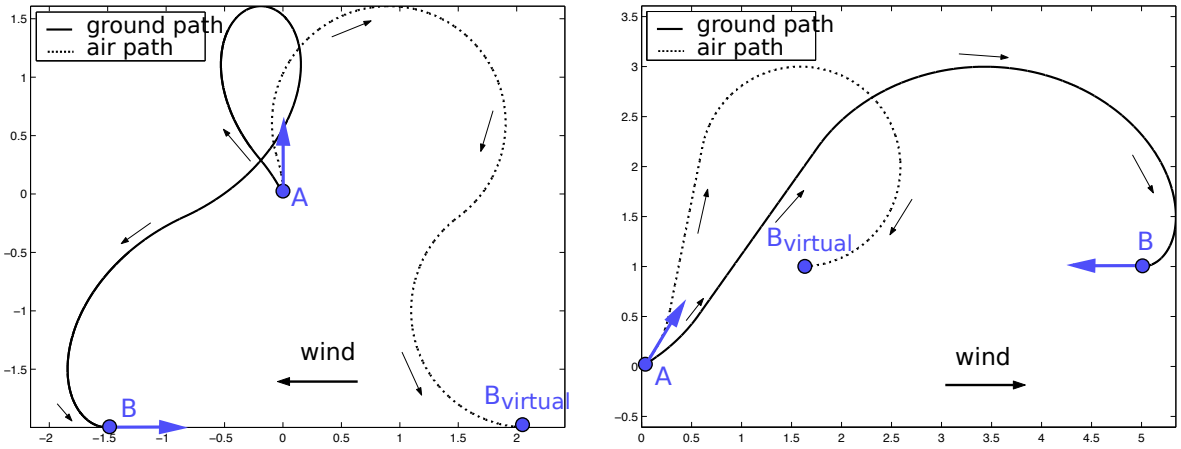


FIGURE 3.15 – Illustration du problème de rendez-vous (G MCGEE et al., 2005)

TECHY et WOOLSEY, 2009 proposent quant à eux les équations mathématiques du mouvement, en observant qu'en présence de vent, un virage (en tant que portion de cercle) produit au sol une forme trochoïdale. Les auteurs parviennent à formuler des solutions analytiques dans certains cas uniquement (semblables à leurs cousins LSL ou RSR de Dubins), les autres cas (similaires aux cas RSL, LSR, RLR, LRL) étant résolus par des méthodes numériques de recherche de racines qui peuvent être compliquées à paramétrer. En faisant l'hypothèse que l'espacement entre les segments est suffisamment important vis-à-vis du rayon de virage, COOMBES et al., 2018 utilisent dans leur critère d'optimisation de la décomposition les solutions analytiques correspondant aux cas « faciles » LSL ou RSR.

5. Synthèse

Il existe une infinité de trajectoires permettant de couvrir un champ. Comme nous l'avons vu, certaines méthodes de décomposition approximative permettent de générer une trajectoire « optimisée » sans beaucoup d'hypothèses quant à la forme du plan de vol. Ce manque de structure ne contraint pas suffisamment le problème, ce qui entraîne des temps de calcul conséquents.

Étant donné que les images doivent être acquises en ligne droite et non en virage, nous pouvons dès lors contraindre le problème en cherchant une trajectoire constituée, à l'intérieur du champ, de segments. Les méthodes de décomposition exacte permettent alors de diviser le champ en plusieurs cellules qui posséderont toutes une direction de parcours privilégiée au regard de leur forme, fixant ainsi l'orientation des segments. La grande majorité des publications de l'état de l'art résolvent le problème de la décomposition soit :

- de manière gloutonne, avec des recettes géométriques parfois compliquées et qui peuvent produire des résultats sous-optimaux voire surprenants sur certaines instances ;
- de manière exhaustive, avec un gage d'optimalité mais au détriment des performances, qui deviennent rapidement une vraie limitation.

Si l'idée de décomposer le champ nous semble incontournable, il nous faudra néanmoins trouver un bon compromis entre optimalité et performance. Par ailleurs, le critère d'optimisation adopté par les différents auteurs varie. Certains minimisent le nombre de segments et donc de demi-tours, ce qui n'est pas un bon indicateur pour les véhicules non holonomes, d'autant plus en présence de vent. D'autres auteurs cherchent directement à estimer et minimiser la durée totale de la trajectoire, ce que nous essayerons de faire également.

Nous retiendrons l'idée présentée dans le paragraphe 2.2.2 car elle permet de résoudre le problème de l'ordre de parcours des segments, qui a une influence considérable sur la durée de la mission mais qui reste pour l'heure peu exploitée dans les méthodes de CPP par décomposition exacte.

Rares sont aussi les publications qui prennent en compte la présence de zones sans intérêt mais survolables à l'intérieur du champ. Certaines publications sont tentées de considérer ces zones comme des obstacles pour pouvoir appliquer des algorithmes inspirés du monde de la robotique terrestre. Néanmoins, dans la mesure où ces zones ne nécessitent pas nécessairement d'être contournées, la trajectoire obtenue peut être sous-optimale. Nous tâcherons donc de les intégrer correctement dans notre modèle.

Enfin, compte tenu des difficultés présentées par l'ajout de vent, nous avons pour le moment choisi de l'ignorer lors de la détermination des trajectoires sol à l'aide des chemins de Dubins, en comptant sur le contrôleur du pilote automatique pour corriger les effets du vent. Une fois la trajectoire déterminée, sa durée sera néanmoins estimée en prenant en compte le vent, pour rester réaliste.

4 Algorithme de planification

Nous présentons ici le travail qui a été réalisé pour répondre au problème de génération de plan de vol. Son extension à la planification en ligne sera discutée à la fin du chapitre.

1. Stratégie de résolution

Après analyse de l'état de l'art, l'approche que nous avons retenue pour résoudre le problème de génération de plan de vol est divisée en deux temps :

1. génération de segments de trajectoire recouvrant le champ ;
2. connexion des segments pour créer le plan de vol qui sera envoyé au drone.

Comme nous l'avons vu, la deuxième étape a déjà été résolue de manière efficace par certains auteurs. La section 2 décrira la méthode utilisée, que nous étendrons pour prendre en compte l'effet du vent. Concernant la première étape, l'étude de l'existant ne nous a pas permis de trouver une approche convaincante, qui puisse à la fois répondre à nos contraintes de performance et d'optimalité sur la durée de la mission. Nous détaillerons dans la section 3 la solution que nous avons développée.

Notons que la deuxième étape est indépendante de la première. Ainsi, il est possible de développer une toute autre méthode de génération de segments et de la substituer à celle que nous présentons, si celle-ci venait à montrer ses limites. Bien qu'elles soient architecturalement indépendantes, la combinaison des deux étapes impacte néanmoins l'optimalité.

Pour plus de clarté, l'algorithme de planification hors ligne est d'ores et déjà résumé sur la figure 4.1 et nous ferons référence aux différentes étapes à l'aide des pastilles ● numérotées.

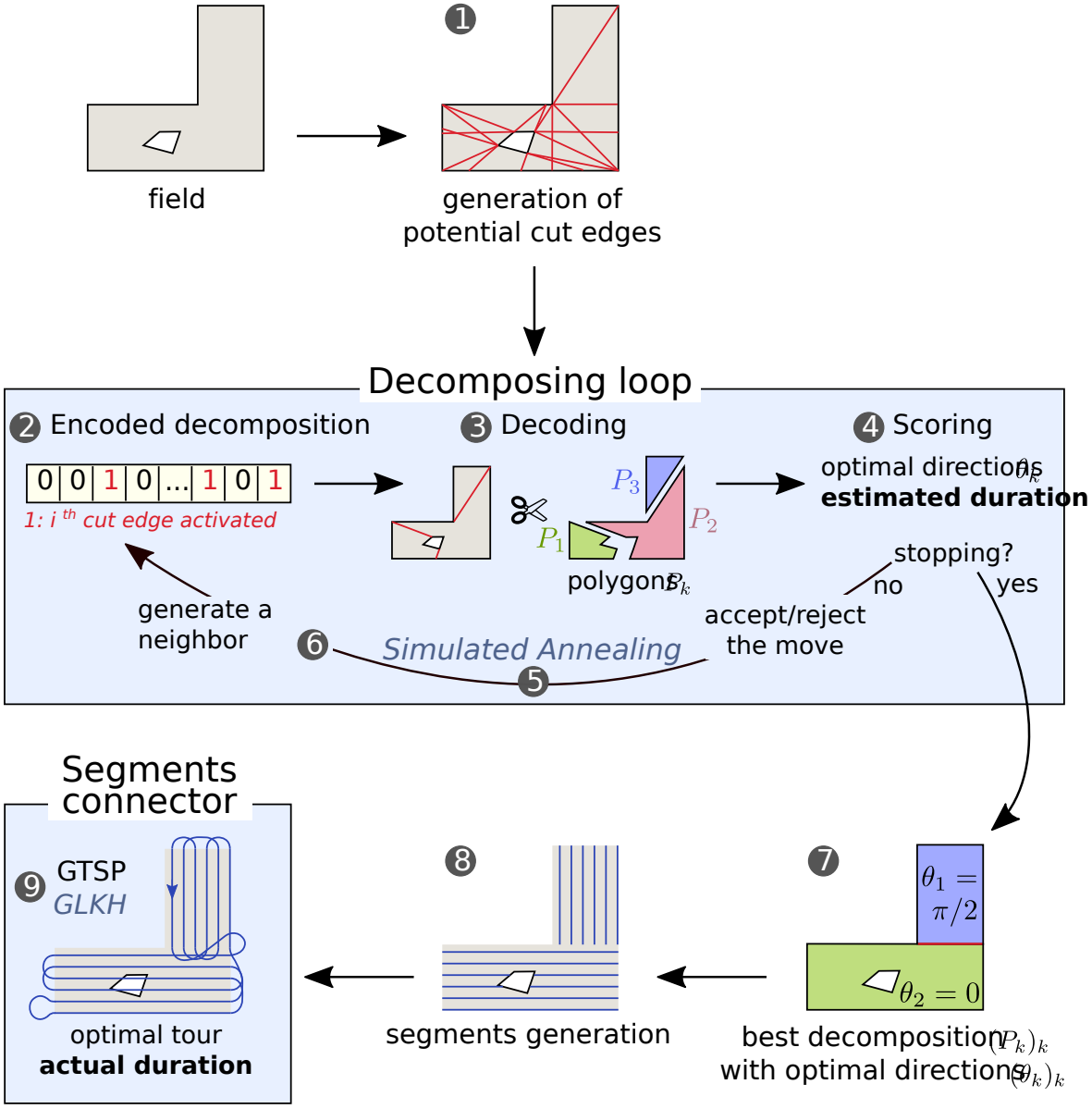


FIGURE 4.1 – Vue d'ensemble de l'algorithme de planification hors ligne

2. Création de trajectoire par connexion de segments 9

Nous disposons d'une liste de segments couvrant le champ. Notre but est de les connecter de manière à produire un chemin de Dubins optimal, en temps, qui sera envoyé au drone.

2.1 Durée d'une trajectoire en présence de vent

Comme expliqué précédemment, nous considérons que le drone vole à une vitesse air constante et qu'en présence de vent, le contrôleur du pilote automatique corrige uniquement le cap afin de reproduire le plus fidèlement possible le plan de vol, défini en terme de trajectoire sol (route). Le vent est considéré uniforme et constant lors de la génération du plan de vol et s'il venait à évoluer, la décision adaptée serait prise par la planification en ligne.

Les vecteurs vitesse air, vitesse sol et vent seront notés respectivement $\overrightarrow{V_a}$, $\overrightarrow{V_g}$ et $\overrightarrow{W}$ et leurs normes v_a (constante), v_g et v_w (constante). La direction relative du vent par rapport à la route correspond à l'angle γ, avec $\gamma = 0$ pour un vent de dos et $\gamma = \pi$ pour un vent de face.

La figure 4.2 illustre comment, en posant l'extrémité du vecteur vent sur le cercle de rayon v_a de telle manière que l'autre extrémité rencontre la trajectoire sol, on obtient le vecteur $\overrightarrow{V_a}$ permettant de conserver le drone sur sa trajectoire sol ainsi que la valeur de la vitesse sol v_g.

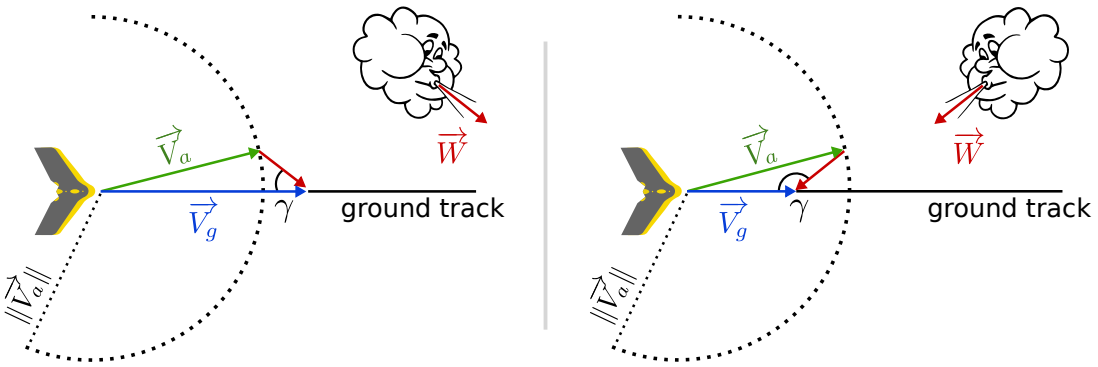


FIGURE 4.2 – Triangle des vitesses avec vitesse air constante

La loi des cosinus s'écrit :

$$v_a^2 = v_g^2 + v_w^2 - 2\cos(\gamma)v_g v_w$$

La vitesse sol v_g correspond alors à la racine positive de ce polynôme du second degré :

$$v_g(\gamma, v_w) = \sqrt{v_a^2 - \sin^2(\gamma)v_w^2} + \cos(\gamma)v_w$$

Ou encore, en notant θ_g la route de l'UAV et θ_w la direction du vent, dans les conventions trigonométriques :

$$v_g(\theta_g, \theta_w, v_w) = \sqrt{v_a^2 - \sin^2(\theta_g - \theta_w)v_w^2} + \cos(\theta_g - \theta_w)v_w \tag{4.1}$$

Lorsque le vent est aligné avec la trajectoire sol, la vitesse sol est bien égale à $v_a + v_w$ dans le cas d'un vent de dos et à $v_a - v_w$ dans le cas d'un vent de face. Notons que cette construction impose que la vitesse air soit supérieure à la vitesse du vent, condition a priori nécessaire à tout vol.

Une trajectoire est composée de segments (tronçons de Dubins S) connectés par des courbes de Dubins (combinaison de tronçons S, L et R). La trajectoire correspond donc à une succession de tronçons S, L et R et on définit la durée d'une trajectoire par la somme des durées individuelles des tronçons.

Durée d'un tronçon S Pour un segment en ligne droite de longueur l, orienté dans la direction θ_g, la durée de parcours est simplement égale à :

$$\frac{l}{v_g(\theta_g, \theta_w, v_w)} \tag{4.2}$$

Durée d'un tronçon L ou R Un tronçon de virage L ou R peut se définir par une direction de départ $\theta_{g,start}$ et une direction d'arrivée $\theta_{g,end}$ ainsi qu'un sens de parcours (gauche ou droite), le déplacement se faisant sur le cercle de rayon de virage minimum r.

On a de manière immédiate l'équation différentielle suivante, pour un tronçon L (la petite variation $d(r\theta_g)$ étant positive) :

$$\begin{aligned} v_g(\theta_g, \theta_w, v_w) &= \frac{d(r\theta_g)}{dt} \\ \iff \frac{d\theta_g}{dt} &= \frac{1}{r} v_g(\theta_g, \theta_w, v_w) \end{aligned} \tag{4.3}$$

Une intégration numérique, avec une direction initiale θ_0 quelconque, permet alors d'obtenir des couples (t, θ_g) qui représentent la durée t nécessaire pour atteindre une orientation θ_g en partant de l'orientation θ_0, en virage et par la gauche et en supposant que le contrôleur corrige au cap à chaque instant l'effet du vent. En réglant le pas d'intégration correctement, on peut alors obtenir une valeur approchée de la durée du tronçon en calculant une simple différence de temps, en faisant tout de même attention à la positon de θ_0 par rapport à $\theta_{g,start}$

et $\theta_{g,end}$. Dans l'exemple illustré par la figure 4.3, la durée de parcours est égale à $t_2 - t_1$. Le graphique est obtenu par intégration numérique dans les conditions suivantes : direction du vent $\theta_w = \pi$, vitesse du vent $v_w = 10\,\mathrm{m/s}$, vitesse air de l'UAV $v_a = 15\,\mathrm{m/s}$, rayon de virage r de 40 m. Comme attendu, on constate qu'en présence du vent venant de l'est, la vitesse sol est maximale aux alentours de la route $\theta_g = \pi$ et minimale aux alentours de la route $\theta_g = 0$.

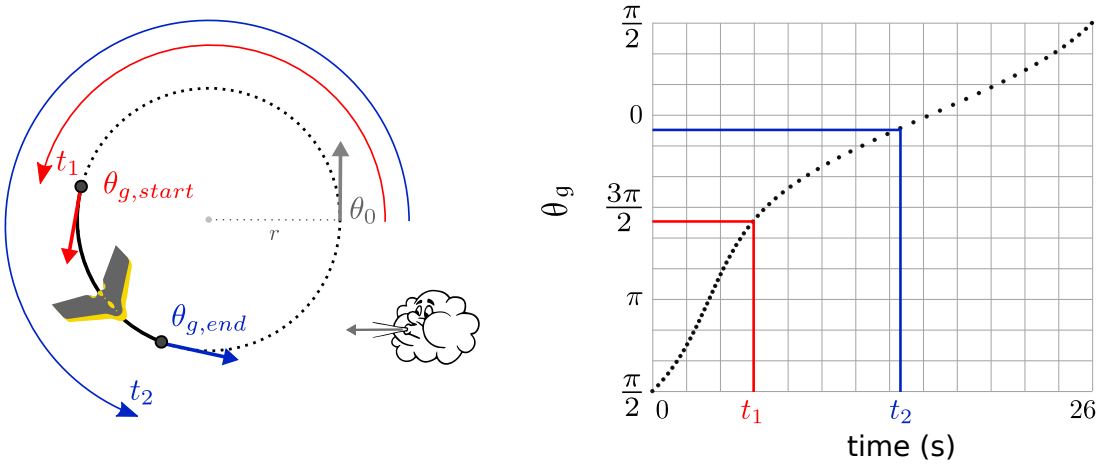


FIGURE 4.3 – Calcul de la durée de parcours d'un tronçon L par intégration numérique

Le cas du tronçon R peut se résoudre de manière analogue en modifiant le sens de parcours dans l'équation différentielle 4.3. Pour simplifier l'implémentation, il peut être plus intéressant de ramener le problème à un tronçon L par n'importe quelle symétrie axiale comme l'illustre la figure 4.4.

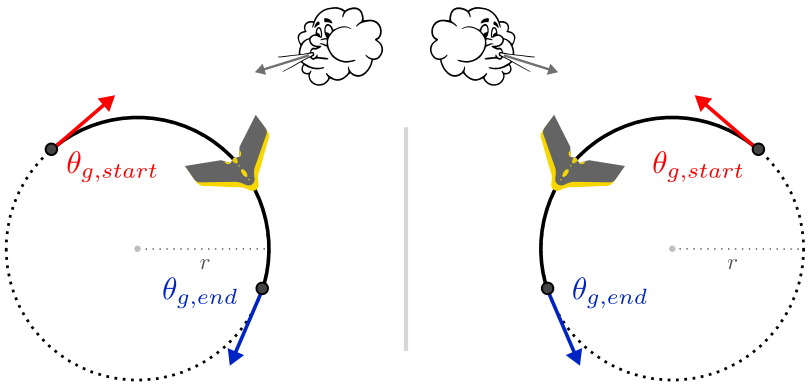


FIGURE 4.4 – La réflexion verticale transforme un tronçon R en un tronçon L de même durée, où les angles de virage et de vent ont été remplacés par leurs angles supplémentaires

2.2 Problème du voyageur de commerce généralisé

Définition Le problème du voyageur de commerce généralisé (*Generalized Traveling Salesman Problem*, GTSP) est une extension du problème du voyageur de commerce (TSP). Étant donnés un ensemble de villes à visiter et un coût associé à chaque paire de ville (la distance qui les sépare, le prix d'un billet de train…), l'objectif du TSP est de trouver le tour de coût minimal qui permette de visiter chacune des villes une et seule fois en revenant au même point de départ. Dans le GTSP, l'ensemble de nœuds (villes) est divisé en sous-ensembles mutuellement disjoints ou *clusters* et l'on cherche le tour de coût optimal visitant exactement un nœud de chaque cluster[1]. En référence aux campagnes électorales américaines, le GTSP est parfois appelé *Traveling Politician Problem*, le candidat souhaitant habituellement visiter une ville de chaque état, ce qui peut soulever un vrai problème d'optimisation au regard du calendrier et de l'étendue du pays.

La figure 4.5 illustre, pour un GTSP composé de 23 nœuds répartis dans 6 clusters, le tour optimal, pour la distance euclidienne, visitant chacun des sous-ensembles.

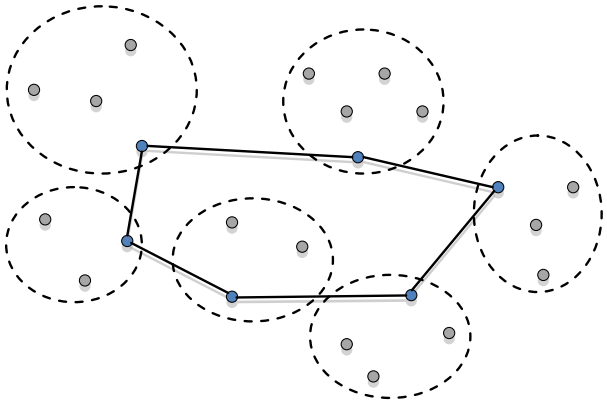


FIGURE 4.5 – Problème du voyageur de commerce généralisé pour 23 villes réparties en 6 clusters (HELSGAUN, 2015)

Soit V l'ensemble des nœuds du problème. Le coût pour connecter le nœud $v_i \in V$ au nœud $v_j \in V$ est noté c_{ij}, la matrice $C = (c_{ij})_{i,j}$ étant appelée matrice de coût ou matrice de distance. Les clusters sont définis par une partition $\mathcal{P}$ de V. Le problème est dit symétrique s'il existe un chemin bidirectionnel entre toute paire de nœuds v_i et v_j appartenant à deux clusters différents et si $c_{ij} = c_{ji}$, auquel cas le graphe est non orienté et le tour correspondra à un cycle. Le problème est dit asymétrique s'il existe, dans deux clusters différents, i et j tels qu'il ne

[1]Certains auteurs préfèrent l'appellation E-GTSP (*Equality Generalized Traveling Salesman Problem*) pour distinguer ce problème d'une autre variante de GTSP où la contrainte devient « au moins un nœud de chaque cluster », parfois préférable en cas de coûts non métriques qui ne vérifient pas l'inégalité triangulaire.

soit pas possible de rejoindre v_j à partir de v_i (e.g. route à sens unique), le cas échéant et par convention $c_{ij} = \infty$, ou s'il existe i et j tels que $c_{ij} \neq c_{ji}$ (coûts non symétriques, e.g. tarifs de billets d'avion ou temps de parcours en présence de vent). Dans ce cas, le graphe est orienté et le tour correspondra à un circuit.

Résolution Notons que le TSP est un cas particulier du GTSP où les sous-ensembles sont réduits à des singletons. Le TSP étant bien connu pour être NP-complet, le GTSP l'est par réduction immédiate tout autant. Ainsi, aucun algorithme de complexité temporelle polynomiale n'est connu, à l'heure actuelle et probablement de manière définitive[2], pour résoudre le GTSP. Les méthodes de résolution exacte connues ayant une complexité exponentielle, la résolution du GTSP pour de grandes instances, notamment asymétriques et non métriques, est problématique. S'il existe dans l'état de l'art quelques métaheuristiques conçues spécialement pour le GTSP, l'implémentation la plus simple et la plus courante consiste à transformer le GTSP en TSP et d'utiliser l'un des nombreux solveurs de TSP disponibles. Il est en effet possible, pour une instance de GTSP asymétrique donnée, de créer une instance de TSP asymétrique qui, une fois résolue, permettra d'extraire la solution du GTSP. La transformation la plus utilisée est celle de NOON et BEAN, 1993, qui a l'avantage d'être exacte tout en gardant la taille du TSP comparable (même nombre de nœuds, quelques arcs supplémentaires). Néanmoins, l'astuce de la transformation repose essentiellement sur la définition des nouveaux coûts du TSP, qui peuvent être nuls mais aussi devenir très grands ; les auteurs notent alors que certaines métaheuristiques conçues pour le TSP pourraient être déstabilisées par cette structure inhabituelle de TSP.

Ce n'est pas le cas de l'heuristique de Lin-Kernighan qui est fréquemment utilisée pour résoudre le TSP. Elle est notamment implémentée dans le solveur LKH dont le code est mis à la disposition des chercheurs[3]. Ce solveur a été testé sur des instances de TSP générées par la transformation de Noon-Bean de diverses instances de GTSP et a produit des résultats très satisfaisants. Cela a encouragé son auteur à intégrer directement la transformation de Noon-Bean comme surcouche à LKH dans un nouveau solveur nommé GLKH (HELSGAUN, 2015), lui aussi disponible au téléchargement[4].

[2] Un algorithme permettant de résoudre en un temps polynomial, sur une machine de Turing déterministe, n'importe quelle instance d'un problème NP-complet quelconque invaliderait l'hypothèse $P \neq NP$.

[3] http://www.akira.ruc.dk/~keld/research/LKH/

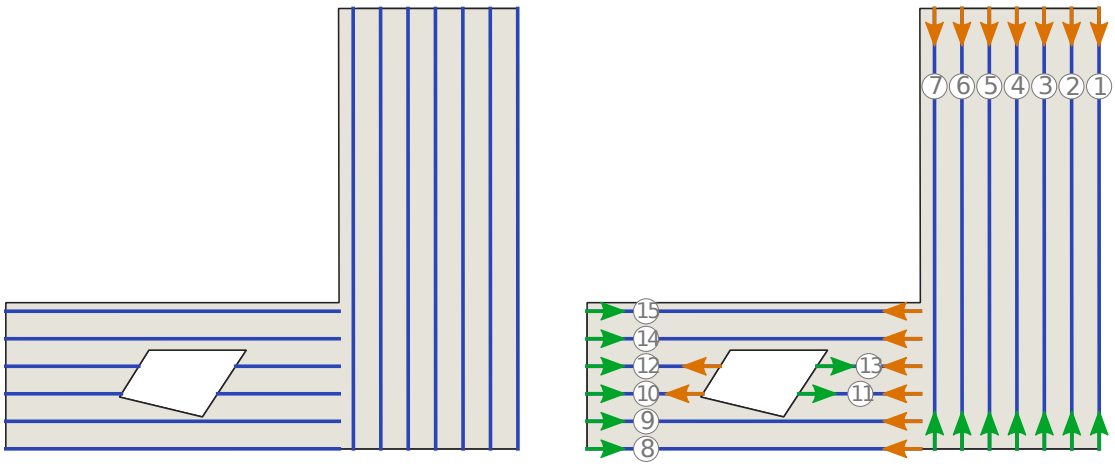
[4] http://www.akira.ruc.dk/~keld/research/GLKH/

2.3 Application du GTSP et premières observations

Nous allons utiliser le GTSP pour connecter nos segments d'acquisition dans le but de déterminer la trajectoire de Dubins optimale en temps, en l'illustrant sur un exemple. Le champ, en forme de botte et contenant une zone sans intérêt (mais survolable), est représenté sur la figure 4.6a ainsi qu'un ensemble S de 15 segments d'acquisition, en bleu.

2.3.1 Construction du GTSP

Nœuds et clusters Le GTSP est construit ainsi : chaque segment représente un cluster de deux nœuds qui correspondent aux deux points d'entrée possibles du segment et donc à son sens de parcours. On souhaite en effet parcourir tous les segments (clusters) mais qu'une seule fois et dans un sens donné (nœud). Sur la figure 4.6b, les segments sont numérotés et les nœuds sont représentés par les flèches vertes et oranges qui symbolisent aussi l'attitude (position et cap) que le drone doit posséder à l'entrée du segment. Pour un segment $s \in S$, on notera $(s, \rightarrow)$ et $(s, \leftarrow)$ ses deux nœuds.



(A) Segments (en bleu)

(B) Nœuds du GTSP (flèches)

FIGURE 4.6 – Création du GTSP à partir d'un ensemble de segments

Matrice de coûts Pour $n = |S|$ segments, la matrice de coûts est de dimension $2n \times 2n$ et est illustrée par l'équation 4.4.

$$
\begin{array}{c|ccccccc}
 & s_1 & s_1 & s_2 & s_2 & \dots & s_n & s_n \\
 & \rightarrow & \leftarrow & \rightarrow & \leftarrow & \dots & \rightarrow & \leftarrow \\
\hline
s_1\rightarrow & 0 & 0 & \bullet & \bullet & \dots & \bullet & \bullet \\
s_1\leftarrow & 0 & 0 & \bullet & \bullet & \dots & \bullet & \bullet \\
s_2\rightarrow & \bullet & \bullet & 0 & 0 & \dots & \bullet & \bullet \\
s_2\leftarrow & \bullet & \bullet & 0 & 0 & \dots & \bullet & \bullet \\
\dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\
s_n\rightarrow & \bullet & \bullet & \bullet & \bullet & \dots & 0 & 0 \\
s_n\leftarrow & \bullet & \bullet & \bullet & \bullet & \dots & 0 & 0
\end{array}
\tag{4.4}
$$

Le coût de transition entre deux nœuds d'un même segment s_i est fixé à 0, ce qui importe peu puisqu'ils ne pourront jamais être connectés, appartenant au même cluster.

Par rapport au GTSP « classique », les coûts présentent ici une subtilité car il ne s'agit pas de connecter un nœud à un autre nœud comme l'on connecterait deux villes. En effet, et pour exemple, le coût de la transition de $(s_i, \leftarrow)$ à $(s_j, \rightarrow)$, avec $i \neq j$, correspond à la somme des durées nécessaires pour :

- parcourir le segment s_i avec pour point d'entrée et donc comme direction le nœud $\leftarrow$;
- rejoindre ensuite, à partir de l'extrémité du segment s_i atteinte, le point d'entrée $\rightarrow$ du segment s_j.

La première durée se calcule facilement avec l'équation 4.2. La deuxième durée nécessite tout d'abord de déterminer le chemin optimal entre les deux points orientés. Comme expliqué dans la synthèse de l'état de l'art, nous simplifions le problème en ignorant le vent et en calculant le chemin de Dubins optimal. Sa durée est ensuite calculée, cette fois-ci en considérant le vent, par la méthode d'intégration numérique présentée au paragraphe 2.1.

Ainsi construit, le coût d'un tour, qui sera la variable d'optimisation lors de la résolution du GTSP, correspond donc à la durée totale pour parcourir la trajectoire (segments et transitions). Notons que le GTSP obtenu est asymétrique.

Le temps de calcul nécessaire pour remplir cette matrice, qui croît avec le nombre de segments, est négligeable dans notre cas.

2.3.2 *Résolution et observations*

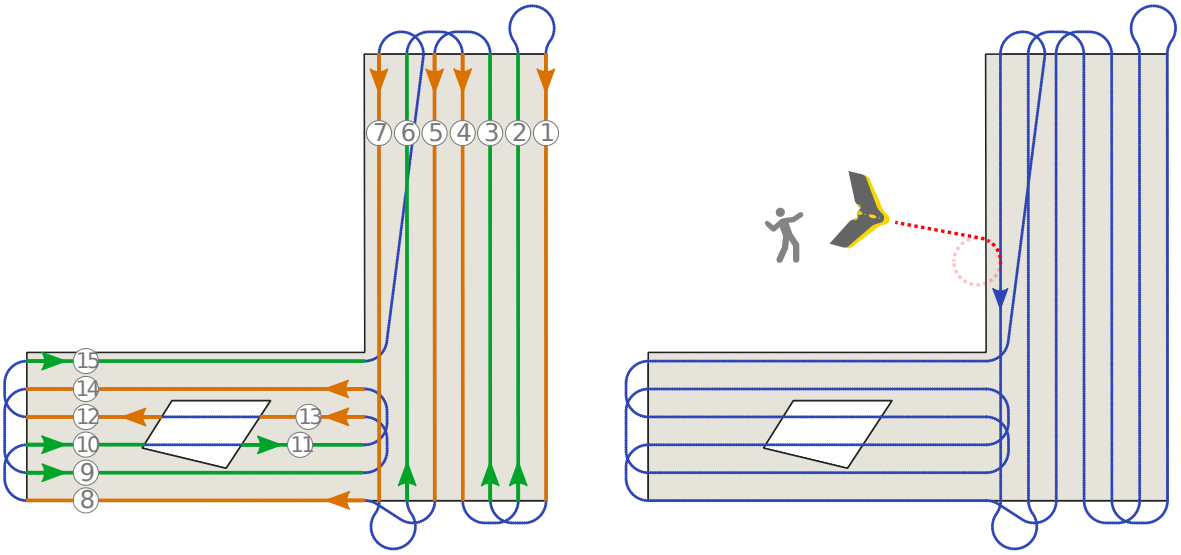
Le solveur GLKH présenté au paragraphe 2.2 est utilisé pour résoudre le problème.

Dans notre exemple, la solution (très probablement optimale compte tenu du faible nombre de nœuds) correspond au tour orienté ou circuit suivant :

$$(s_1, \leftarrow) \Rightarrow (s_3, \rightarrow) \Rightarrow (s_5, \leftarrow) \Rightarrow (s_8, \leftarrow) \Rightarrow (s_{10}, \rightarrow) \Rightarrow (s_{11}, \rightarrow) \Rightarrow (s_{14}, \leftarrow) \Rightarrow (s_9, \rightarrow)$$
$$\Rightarrow (s_{13}, \leftarrow) \Rightarrow (s_{12}, \leftarrow) \Rightarrow (s_{15}, \rightarrow) \Rightarrow (s_7, \leftarrow) \Rightarrow (s_6, \rightarrow) \Rightarrow (s_4, \leftarrow) \Rightarrow (s_2, \rightarrow) \Rightarrow (s_1, \leftarrow)$$

Ce circuit est illustré sur la figure 4.7a où sont représentées en bleu les transitions « $\Rightarrow$ » connectant les segments, qui correspondent à quelques-uns des chemins de Dubins qui avaient été préalablement calculés au moment de remplir la matrice de coûts.

Si le chemin est orienté du fait de l'asymétrie du problème, il n'existe cependant pas de point de départ ou d'arrivée préférentiel, la solution étant cyclique. En effet, en partant d'une position quelconque du chemin, on y reviendra nécessairement après avoir parcouru l'intégralité de la trajectoire. Ainsi, l'opérateur pourra faire décoller le drone et lui demander de s'insérer dans le circuit au plus vite, avec une trajectoire de Dubins, pour débuter la mission (figure 4.7b).



(A) Détail du circuit

(B) Initialisation du plan de vol

FIGURE 4.7 – Trajectoire optimale obtenue par la résolution du GTSP

Contrairement à d'autres méthodes plus systématiques présentées dans l'état de l'art, cette approche de connexion de segments par résolution d'un GTSP présente plusieurs avantages que l'on peut d'ores et déjà observer sur notre exemple.

À première vue, les transitions semblent bien optimisées car hormis la transition du segment (15) au segment (7), le surplus de durée engendré par les transitions est moindre. En cherchant la trajectoire de durée minimale, le GTSP a en quelque sorte mené une optimisation double qui porte à la fois sur l'ordre des segments et leur sens de parcours. Les conséquences sont intéressantes :

- Avec ce rayon de virage et l'espacement imposé entre les segments d'acquisition, on constate que le demi-tour nécessaire pour passer d'un segment au segment voisin (chemin RLR ou LRL en forme de bulbe, e.g. la transition (2) ⇒ (1)) est plus coûteux que la transition qui, « enjambant » un segment, connecterait un segment sur deux (chemin LSL ou RSR, e.g. la transition (1) ⇒ (3)). Cette dernière est donc préférée dès que possible, ce qui est bien sûr contraint par la parité du nombre de segments ou ce qu'il est possible de faire pour clore le circuit, par exemple.
- La zone sans intérêt a été couverte car elle est suffisamment petite pour qu'il soit moins coûteux de la traverser plutôt que de la contourner, en gardant en esprit et pour but l'optimisation globale de la trajectoire. Les transitions (10) ⇒ (11) et (13) ⇒ (12) ont d'ailleurs été facilitées, pour ne pas dire encouragées, par l'alignement des segments. Cela ne sera pas toujours le cas : en agrandissant un petit peu la zone sans intérêt, on constate que la trajectoire optimale tire profit de cette zone pour économiser quelques secondes par rapport à la trajectoire qui l'aurait couverte (figure 4.8).

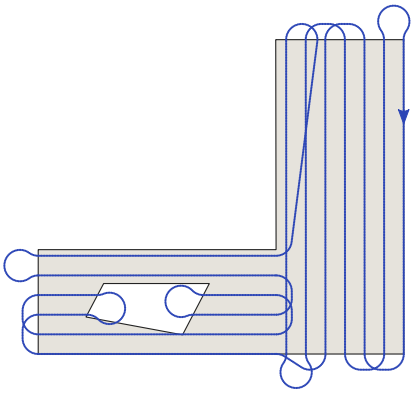


FIGURE 4.8 – Trajectoire optimale lorsque l'on agrandit la zone sans intérêt

- La trajectoire obtenue passe allègrement d'une partie du champ à une autre, sans avoir nécessairement terminé de la couvrir : après avoir couvert les segments (1), (3) et (5), le drone rejoint le segment (8) et couvre la partie gauche du champ avant de revenir compléter la partie droite avec les segments (7), (6), (4) et (2).

Cette approche est donc bien plus probante, du moins dans le cas d'un drone à voilure fixe, que la méthode qui consisterait par exemple à visiter les segments de voisin en voisin, en contournant systématiquement les zones sans intérêt. Sa flexibilité lui permet en effet de s'abstraire d'un ordre de parcours préétabli et d'une vision trop rigide de l'environnement, et elle résout le problème du coût des transitions non maîtrisé entre les différentes cellules que l'on rencontre dans les méthodes classiques « boustrophedon ».

3. Génération de segments par décomposition du champ

La deuxième étape ayant été présentée, il nous est maintenant possible de générer une trajectoire optimale en temps à partir d'un ensemble de segments d'acquisition. Il nous reste à déterminer comment générer ces segments, première étape de la planification.

Décomposition et directions Soit P le polygone représentant le champ. Nous cherchons une décomposition exacte de P en un ensemble $(P_k)_k$ de polygones ou « sous-champs » qui ne s'intersectent pas, c'est-à-dire une partition de P. Nous considérons que chacun des polygones sera couvert par des segments parallèles et espacés d'une certaine distance connue. À chaque sous-champ P_k est donc associée une direction θ_k qui correspondra à l'orientation des segments de la trajectoire à l'intérieur de ce polygone. La figure 4.9 illustre la décomposition et les directions utilisées sur l'exemple de la section précédente.

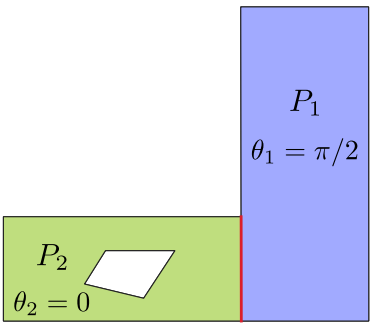


FIGURE 4.9 – Décomposition du champ $(P_k)_k$ avec les directions $(\theta_k)_k$

Formalisation du problème Notons $d_{GTSP}((P_0, \theta_0), \ldots, (P_n, \theta_n))$ la durée de la trajectoire optimale obtenue après la résolution du GTSP, détaillée dans la section précédente, sur la décomposition $(P_k)_k$ et avec les directions $(\theta_k)_k$.

Idéalement, il faudrait trouver la décomposition et les directions optimales qui minimisent la durée de la trajectoire en résolvant le problème d'optimisation suivant :

$$\begin{aligned} & \min_{(P_k,\theta_k)_k} d_{GTSP}((P_k,\theta_k)_k) \\ \text{s.t.} \quad & P = \bigcup_k P_k \quad \text{et} \quad P_i \cap P_j = \emptyset,\ i \neq j \\ & \theta_k \in [0,\pi[\end{aligned} \tag{4.5}$$

Exprimé ainsi, ce problème comporte deux difficultés majeures. La première concerne la complexité et les performances car la fonction d_{GTSP} étant elle-même un problème d'optimisation NP-complet, il serait relativement coûteux de l'utiliser ici. La deuxième difficulté est liée à l'espace de recherche des décompositions et directions, a priori continu et infini.

Pour simplifier le problème, nous utilisons une heuristique pour calculer le coût d'une trajectoire. Nous faisons l'hypothèse que les transitions entre les différents polygones seront d'une durée négligeable dans la trajectoire finale obtenue par le GTSP. Autrement dit, il devrait être possible d'estimer plus ou moins précisément la durée de la trajectoire finale en s'intéressant séparément à chacun des sous-champs et en sommant les durées nécessaires pour les couvrir individuellement. Nous définissions alors la fonction d_{est} qui, pour un polygone P_k et une direction θ_k, calcule une estimation de la durée de la trajectoire nécessaire pour couvrir le polygone dans la direction donnée. L'optimisation s'écrit maintenant :

$$\begin{aligned} & \min_{(P_k,\theta_k)_k} \sum_k d_{est}(P_k,\theta_k) \\ \text{s.t.} \quad & P = \bigcup_k P_k \quad \text{et} \quad P_i \cap P_j = \emptyset,\ i \neq j \\ & \theta_k \in [0,\pi[\end{aligned} \tag{4.6}$$

Les directions θ_k étant indépendantes des polygones P_k, la séparabilité introduite dans le problème nous permet de le récrire ainsi :

$$\begin{aligned} & \min_{(P_k)_k} \sum_k \min_{\theta_k \in [0,\pi[} d_{est}(P_k,\theta_k) \\ \text{s.t.} \quad & P = \bigcup_k P_k \quad \text{et} \quad P_i \cap P_j = \emptyset,\ i \neq j \end{aligned} \tag{4.7}$$

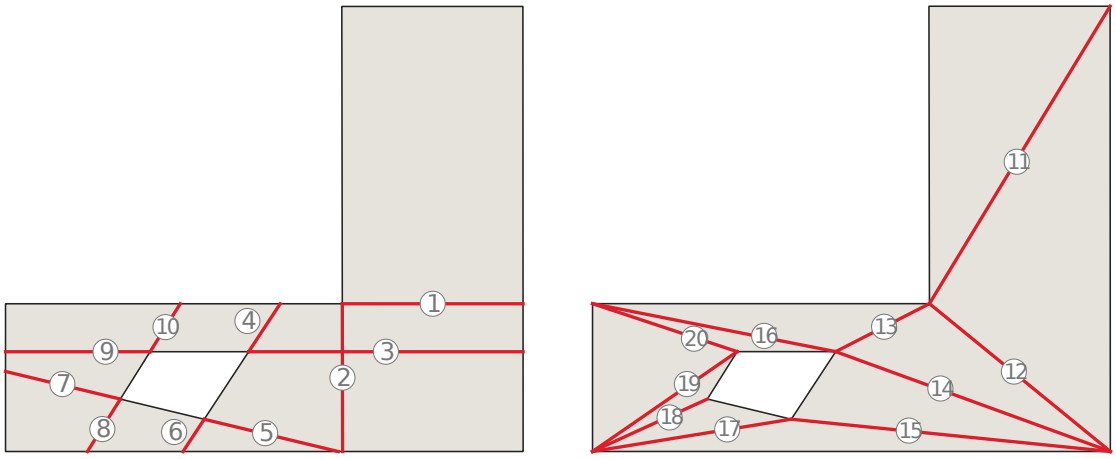
Le paragraphe 3.1 s'intéressera à la description et à la modélisation de l'espace de recherche de la décomposition $(P_k)_k$. Le paragraphe 3.2 traitera de la fonction d_{est} et des directions θ_k. Enfin, le paragraphe 3.3 présentera la métaheuristique utilisée pour résoudre le problème.

3.1 Modélisation d'une décomposition

3.1.1 Génération de coupes potentielles 1

Il existe une infinité de décompositions $(P_k)_k$ du champ. Pour réduire la taille de l'espace de recherche, continu et infini, nous restreignons drastiquement la manière dont est construite la partition $(P_k)_k$ en générant un ensemble de « coupes potentielles ». Un certain nombre d'entre elles pourront alors être utilisées pour couper le champ et former des polygones qui auront pour contour à la fois des arêtes issues du champ originel et des arêtes correspondant à ces coupes.

La figure 4.10 illustre deux manières arbitraires de générer des coupes potentielles. La première (figure 4.10a) consiste à étendre les arêtes aux sommets introduisant de la concavité. La seconde (figure 4.10b) utilise un algorithme classique de décomposition d'un polygone par triangles (triangulation par méthode des oreilles ou *ear clipping*).



(A) Extension aux concavités

(B) Triangulation

FIGURE 4.10 – Génération de coupes potentielles

Sur la figure 4.11 est représentée la superposition des deux jeux de coupes. Le compromis consiste alors à espérer que ces coupes puissent permettre d'approcher la décomposition optimale.

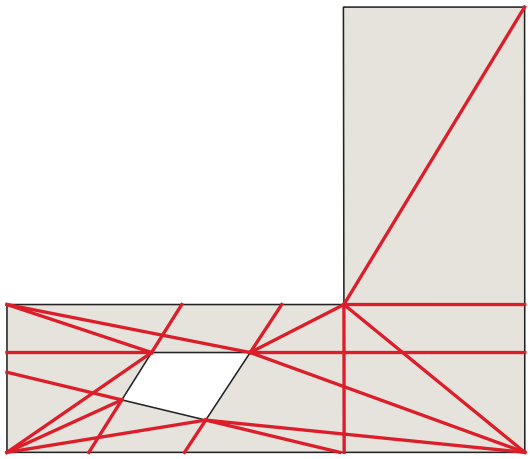


FIGURE 4.11 – Coupes potentielles

3.1.2 *Encodage d'une décomposition* 2

Pour résoudre le problème à l'aide d'une métaheuristique, nous avons besoin de définir l'*encodage* d'une décomposition, c'est-à-dire une représentation informatique simple de la décomposition et si possible bijective avec l'espace de recherche des décompositions.

Comme expliqué dans le paragraphe précédent, la décomposition $(P_k)_k$ résultera d'une série de coupes réalisées dans le champ parmi un ensemble de coupes potentielles autorisées. La décomposition peut donc être représentée par un vecteur binaire dont la taille n est égale au nombre de coupes potentielles générées (dans notre exemple, $n = 20$) et qui, pour chaque coupe, indiquera si celle-ci est active, c'est-à-dire sélectionnée pour découper le champ. Ce vecteur sera appelé *vecteur de décision*, noté x, et est illustré par la figure 4.12.

	1	2	3	4	$\cdots$	$n-2$	$n-1$	n
$x =$	0	0	1	0	$\cdots$	0	0	1

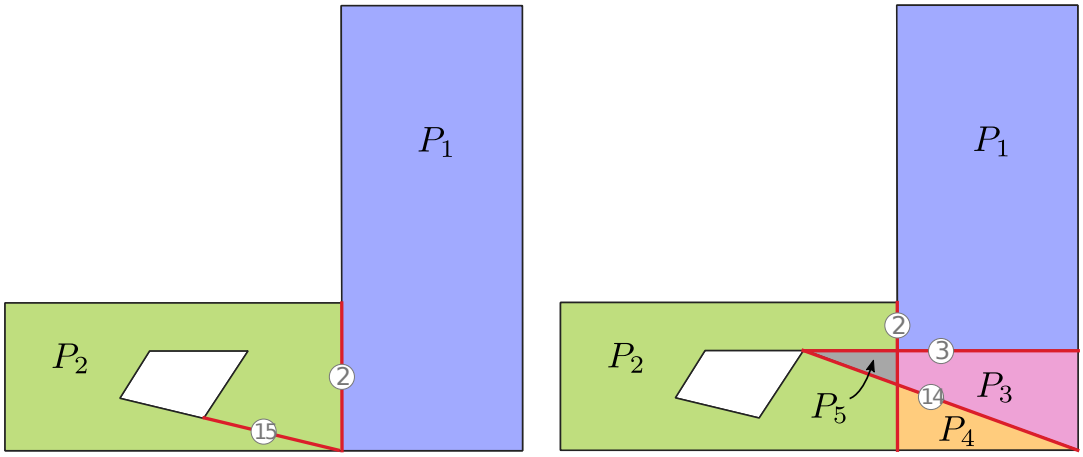
FIGURE 4.12 – Vecteur de décision encodant la décomposition obtenue par les coupes (3) et (n)

À titre d'exemple, la décomposition de la figure 4.9 est encodée par le vecteur binaire nul partout sauf pour la coupe (2), seule sélectionnée pour couper le champ.

Cet encodage peut générer des décompositions problématiques lorsqu'une coupe est sélectionnée sans qu'elle ne puisse délimiter de nouveau polygone. C'est par exemple le cas de la décomposition obtenue après sélection des coupes (2) et (15), cette dernière étant superflue comme l'illustre la figure 4.13a. La décomposition est alors équivalente à la décomposition faisant intervenir uniquement la coupe (2). Par la suite, nous ne nous intéresserons

pas à détecter ces cas a priori : une telle décomposition sera considérée comme valide et algorithmiquement, la présence d'une coupe superflue ne nous gênera pas.

Enfin, pour restreindre davantage l'espace de recherche et gagner en performance, nous interdirons la sélection de coupes qui s'intersectent : en pratique, aucune décomposition obtenue au cours du stage ne faisait intervenir de coupes qui s'entrecoupaient. Cela s'explique simplement en observant que l'ajout d'une coupe qui intersecte une ou plusieurs autres coupes peut engendrer un grand nombre de polygones comme le montre la figure 4.13b, rendant cette décomposition très probablement sous-optimale.



(A) Coupe superflue

(B) Coupes qui s'intersectent

FIGURE 4.13 – Exemples de décompositions problématiques

Le *décodage* 3 du vecteur de décision x consistera à sectionner le champ suivant les coupes actives afin d'extraire les polygones $(P_k)_k$.

En ignorant les quelques cas problématiques mentionnés ci-dessus, cet encodage produit un espace de recherche exponentiel en nombre de coupes potentielles.

3.1.3 *Voisinage d'une décomposition* 6

Finalement, nous avons besoin de définir le *voisinage* d'une décomposition pour pouvoir utiliser certaines métaheuristiques. En optimisation, le voisinage d'une solution, d'un état, d'un vecteur de décision… consiste en une petite modification facilement réversible de la solution qui permet de produire de nouveaux états « voisins ».

Notre encodage nous permet de définir simplement une structure de voisinage. Lorsque l'on souhaitera générer une décomposition voisine à la décomposition encodée par le vecteur de décision x, on tirera aléatoirement un indice i entre 1 et n. Deux cas se présenteront alors :

- si $x[i] = 1$, i.e. la coupe (i) fait partie de la décomposition : on désactive la coupe en effectuant $x[i] = 0$;
- si $x[i] = 0$, i.e. la coupe (i) ne fait pas partie de la décomposition : on active la coupe en effectuant $x[i] = 1$ et on désactive toutes les coupes (j) qui intersectent la coupe (i) en effectuant $x[j] = 0$.

3.2 Fonction objectif 4

Rappelons le problème d'optimisation que nous souhaitons résoudre :

$$\begin{aligned} &\min_{(P_k)_k} \sum_k \min_{\theta_k \in [0,\pi[} d_{est}(P_k, \theta_k) \\ \text{s.t.} \quad &P = \bigcup_k P_k \quad \text{et} \quad P_i \cap P_j = \emptyset,\ i \neq j \end{aligned} \tag{4.7}$$

Dans le paragraphe précédent, nous avons introduit tout ce qui nous permettra de générer des décompositions $(P_k)_k$. Il nous reste donc maintenant à nous intéresser à la fonction objectif

$$f((P_k)_k) = \sum_k \min_{\theta_k \in [0,\pi[} d_{est}(P_k, \theta_k) \tag{4.8}$$

qui somme les durées estimées de couverture de chaque sous-champ P_k dans sa direction optimale θ_k. Comme expliqué au début de cette section, nous formulons en effet l'hypothèse que les transitions entre les différents polygones seront négligeables dans la trajectoire finale, ce qui nous incite à considérer chaque sous-champ séparément.

3.2.1 Estimation de la durée de couverture dans une direction

Intéressons-nous d'abord à la fonction d_{est}. Étant donné un sous-champ P_k à couvrir avec des segments de trajectoire orientés dans la direction θ_k, $d_{est}(P_k, \theta_k)$ devra approcher au mieux $d_{GTSP}(P_k, \theta_k)$, c'est-à-dire la durée de la trajectoire optimale obtenue par résolution du GTSP après génération de segments régulièrement espacés dans la direction θ_k et recouvrant le polygone P_k.

En plus de l'impossibilité de résoudre un GTSP dans la fonction d_{est} pour des raisons de performance, nous nous sommes également interdit de construire ici des segments, la géométrie algorithmique étant coûteuse et notamment les fonctions d'intersection dont nous aurions besoin.

Pour écrire la fonction d_{est}, nous allons devoir anticiper les choix réalisés par le GTSP. La durée de la trajectoire fait intervenir deux contributions : la durée $d_{segments}$ nécessaire pour parcourir à proprement parler les segments et la durée $d_{transitions}$ induite par les transitions entre les segments :

$$d_{est}(P_k, \theta_k) = d_{segments}(P_k, \theta_k) + d_{transitions}(P_k, \theta_k) \tag{4.9}$$

Segments Intuitivement, nous pouvons supposer que la trajectoire sera majoritairement constituée d'allers-retours puisqu'une fois l'extrémité d'un segment atteinte, la transition optimale semble consister à rejoindre et parcourir un segment voisin dans la direction opposée et ainsi de suite, ce que l'on peut constater sur l'exemple de la figure 4.7.

Puisque nous nous sommes interdit de générer des segments, nous devons estimer la durée nécessaire pour parcourir ces segments, encore imaginaires, d'une manière détournée. En mettant bout à bout les segments, espacés de $spacing$ m, on peut estimer la longueur totale à parcourir par :

$$l = \frac{\mathcal{A}(P_k)}{spacing} \tag{4.10}$$

où $\mathcal{A}(P_k)$ désigne l'aire utile du polygone P_k, en excluant les zones sans intérêt – qui s'apparentent à des trous et qui seront traitées plus tard.

On peut donc prédire statistiquement que la moitié de cette distance sera parcourue dans la direction θ_k tandis que l'autre moitié sera parcourue dans la direction $\theta_k + \pi$. L'équation 4.2 permet alors d'exprimer la durée $d_{segments}$ nécessaire pour parcourir les segments :

$$\begin{aligned} d_{segments}(P_k, \theta_k) &= \frac{l}{2v_g(\theta_k, \theta_w, v_w)} + \frac{l}{2v_g(\theta_k + \pi, \theta_w, v_w)} \\ &= \frac{\mathcal{A}(P_k)}{spacing} \times \frac{\sqrt{v_a^2 - \sin^2(\theta_k - \theta_w)v_w^2}}{v_a^2 - v_w^2} \end{aligned} \tag{4.11}$$

Transitions sur les arêtes extérieures Nous nous intéressons maintenant aux transitions induites par les arêtes extérieures du polygone, en ignorant pour le moment le cas des zones sans intérêt.

Les formes et les durées des différentes transitions dépendent fortement de la géométrie du polygone et plus précisément des pentes des différentes arêtes, comme le montre la figure 4.14. En effet, lorsque l'arête est presque perpendiculaire aux segments (arête (1)), les transitions ont un coût minimal et plus l'angle entre l'arête et les segments diminue, plus le coût des transitions augmente (arête (2)), avec une singularité lorsque l'angle est nul, signifiant alors une arête parallèle aux segments et donc sans demi-tours (arête (5)).

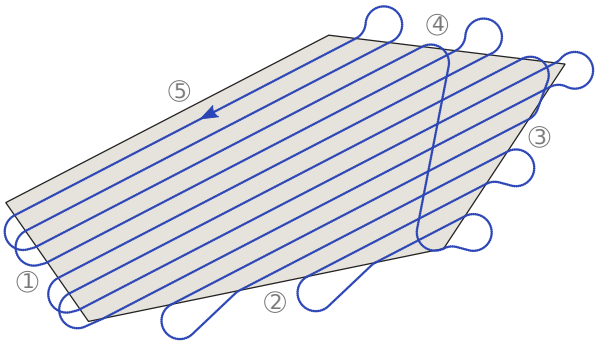


FIGURE 4.14 – Influence de la géométrie du polygone sur les transitions

Pour cette raison, nous allons estimer la durée $d_{transitions}$ nécessaire pour parcourir les transitions en nous intéressant séparément à chaque arête et aux transitions qu'elle est susceptible d'engendrer.

Il est possible, connaissant l'espacement entre les segments et la pente de l'arête, d'estimer géométriquement le nombre de segments qui viendront buter contre l'arête et de calculer les points orientés d'entrée et de sortie des segments. De plus, comme observé lors de la résolution du GTSP au paragraphe 2.3.2, la trajectoire optimale joue avec les transitions pour réduire le temps de parcours en choisissant, pour le mieux, de connecter un segment directement à son segment voisin ou d'enjamber un ou plusieurs segment(s) suivant le rayon de virage de l'UAV et l'espacement entre les segments. Le sens dans lequel nous envisageons les transitions (par la droite ou par la gauche) est également un paramètre important, notamment en présence de vent.

Notre approche consiste donc, pour chaque arête extérieure, à :

1. estimer le nombre de segments qui viendront buter contre cette arête ;
2. déterminer le meilleur type de transition (nombre de segments enjambés et dans quel sens) pour effectuer les allers-retours sur cette arête, le nombre de segments qu'il est possible d'enjamber étant bien sûr limité par le nombre de segments estimés à l'étape 1.

Les différents types de transition sont comparés en calculant leur durée, avec le vent, par la méthode d'intégration numérique présentée au paragraphe 2.1.

La figure 4.15 illustre trois types de transitions vers la gauche possibles sur l'arête (1) de l'exemple précédent. La transition de durée minimale correspond à la deuxième transition qui enjambe un segment, par la gauche. Il faudrait maintenant faire la même comparaison pour des transitions par la droite pour s'assurer de retenir la meilleure.

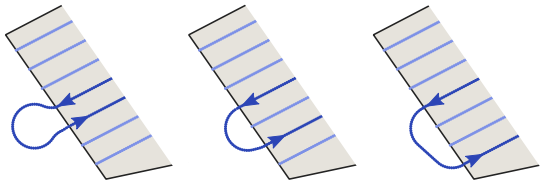


FIGURE 4.15 – Comparaison de trois transitions par la gauche pour l'arête (1)

La somme des durées des transitions retenues pour chaque arête nous donnera alors une estimation de la durée $d_{transitions}$. Cette approche présente l'avantage de fonctionner quelle que soit la géométrie du polygone (convexe ou concave). Elle a cependant des limites évidentes puisque, en s'intéressant à chaque arête séparément, nous n'avons aucun contrôle sur « ce qui se passera en face » en terme de transitions. En choisissant le meilleur pour chaque arête, sans se soucier de la compatibilité des transitions à l'autre extrémité des segments, nous sommes donc résolument optimistes et nous y reviendrons plus tard.

Transitions induites par les zones sans intérêt Nous n'avons toujours pas réglé le cas des zones sans intérêt. Lors de la résolution du GTSP au paragraphe 2.3.2, nous avons constaté qu'il était parfois plus avantageux de « couvrir » une zone sans intérêt plutôt que de la contourner systématiquement. Autrement dit, puisque notre but est ici d'anticiper au mieux les décisions du GTSP, nous ne devrions pas trop nous laisser influencer par ces zones.

Pour chaque zone sans intérêt, deux options s'offrent alors :

1. la zone est suffisamment petite et il serait préférable de l'ignorer ;
2. la zone est suffisamment grande pour qu'il soit plus optimal de la contourner.

Pour prédire quelle stratégie sera adoptée par le GTSP, nous calculons les durées induites par les transitions dans les deux cas.

Dans le premier cas, nous calculons la durée nécessaire pour couvrir la zone Z dans la direction θ_k de la même manière que nous avons estimé la durée de couverture de l'intérieur du polygone, c'est-à-dire en calculant $d_{segments}(Z, \theta_k)$. Cela revient en fait à réintégrer la zone

sans intérêt dans le calcul de l'aire de P_k, en faisant comme si la zone n'existait pas et faisait partie intégrante du polygone.

Dans le deuxième cas, nous calculons la durée nécessaire pour contourner la zone en déterminant les meilleures transitions sur chacune de ses arêtes, de manière analogue à ce que nous avons fait au paragraphe précédent pour les arêtes extérieures du polygone.

Pour ne pas surestimer d_{est} par rapport à d_{GTSP}, nous retenons alors le scénario le plus profitable et ajoutons à $d_{transitions}$ la durée correspondante.

3.2.2 Détermination de la direction optimale

Il nous reste enfin à déterminer la direction des segments $\theta_k^* \in [0, \pi[$ qui minimise la durée de la trajectoire nécessaire pour couvrir le polygone P_k, soit $\theta_k^* = \text{argmin}_{\theta_k} d_{est}(P_k, \theta_k)$. Malheureusement, il n'existe pas de forme analytique pour calculer $d_{est}(\theta_k, P_k)$ et l'idée la plus simple est alors d'essayer plusieurs valeurs de θ_k et de retenir celle qui produit la durée estimée minimale.

Itérer sur les valeurs de $[0, \pi[$ avec un pas de 1° permettrait d'approcher précisément l'optimal mais serait coûteux en calcul.

Intuitivement, nous souhaiterions que l'orientation des segments reflète localement la forme du champ. Comme le démontre HUANG, 2001, la direction qui permet de minimiser le nombre de segments et donc le nombre de transitions est nécessairement la direction d'une des arêtes extérieures ou intérieures (dues aux zones sans intérêt) de P_k (i.e. les segments doivent être parallèles à l'une des arêtes). Bien que notre critère d'optimisation – la durée de la trajectoire – soit plus complexe que le simple nombre de segments ou de demi-tours, nous supposons que cela reste néanmoins une bonne approximation et nous pouvons donc facilement trouver θ_k^* en testant une à une la direction θ_k de chaque arête de P_k et retenir celle qui minimise d_{est}.

Néanmoins, la présence de vent ne permet plus de garantir ce résultat.[5] En supposant que la moitié des segments sera couverte dans la direction θ_k et l'autre moitié dans la direction $\theta_k + \pi$, l'équation 4.11 nous donnait une estimation de la durée de parcours des segments. Il est alors possible de constater que cette durée est minimale lorsque l'UAV vole avec un vent

[5] En présence de vent, il peut parfois être préférable, selon la mission, d'oublier toute optimisation et de placer les segments dans la direction du vent afin que le cap et donc les photographies soient alignées avec la route. Néanmoins, nous considérons ici que l'espacement entre les segments produit un taux de recouvrement suffisant pour produire une carte du champ sans trous, quel que soit le cap.

exactement de travers ($\theta_k = \theta_w \pm \frac{\pi}{2}$). Nous ajouterons donc la direction $\theta_w \pm \frac{\pi}{2}$ aux directions possibles et à tester pour θ_k.

Nous disposons maintenant de tous les ingrédients nécessaires pour résoudre le problème d'optimisation 4.7 : une manière de générer des décompositions $(P_k)_k$ et une méthode permettant d'estimer la durée de la trajectoire pour une décomposition $(P_k)_k$, qui cherche pour chaque polygone P_k une direction de parcours θ_k optimale. La décomposition optimale du champ peut alors être obtenue en appliquant une métaheuristique telle que le recuit simulé présenté ci-dessous.

3.3 Algorithme du recuit simulé 5

En physique et plus particulièrement en métallurgie, le recuit consiste à chauffer un matériau puis à le refroidir de manière à ce qu'il atteigne une nouvelle configuration stable présentant de nouvelles propriétés physiques (recristallisation). Cela peut par exemple permettre d'augmenter la ductilité du métal (capacité à se déformer sans se rompre) en vue de son utilisation future.

Durant le processus de recuit, la structure cristalline du métal évolue du fait de la mobilité des atomes provoquée par le réchauffement. Alors qu'un fluide a tendance à privilégier les états de plus faible énergie, l'apport de chaleur peut entraîner le système vers un état d'énergie supérieure. À chaque niveau de température correspond cependant un équilibre thermodynamique et il existe une température minimale pour laquelle cet équilibre n'évolue plus : la structure du métal est alors figée, l'énergie est minimale et les cristaux ne présentent, en théorie, aucun défaut. Il est parfois nécessaire de maîtriser la phase de refroidissement en s'assurant qu'elle soit suffisamment lente pour que les différents équilibres thermodynamiques puissent être successivement atteints. Autrement dit, le refroidissement ne doit pas être brutal, auquel cas les atomes se figeraient dans une configuration instable.

En 1983, des chercheurs de IBM voient un parallèle intéressant entre la physique statistique, qui permet de comprendre le comportement d'un système à une température donnée vis-à-vis de son équilibre thermodynamique malgré le nombre impressionnant de degrés de liberté à l'échelle atomique, et l'optimisation mathématique. Ils proposent alors l'algorithme du recuit simulé ou *simulated annealing* (KIRKPATRICK et al., 1983), qui constitue encore aujourd'hui une métaheuristique de choix, notamment pour sa facilité d'implémentation.

L'algorithme du recuit simulé est une métaheuristique probabiliste qui s'inspire directement du recuit. Il manipule un état de l'espace de recherche (vecteur de décisions), initialement arbitraire et qui représente le matériau du système, et contrôle un paramètre de température,

initialement élevée, qui décroît tout au long de la procédure. L'analogie consiste à considérer la fonction objectif, que l'on cherche à minimiser, comme l'énergie du système. À chaque itération, le système aura la possibilité d'évoluer vers un autre état, dans le voisinage de l'état courant (petite modification facilement réversible du vecteur de décisions), reproduisant ainsi la mobilité des atomes dans le métal. En se déplaçant de proche en proche, le système pourra donc s'éloigner plus ou moins loin de son état initial. Néanmoins, ces transitions sont soumises à un critère d'acceptation fonction de la température et des niveaux d'énergie (valeurs de la fonction objectif) de l'état courant et du nouvel état envisagé : plus la température diminue, moins il devrait être facile de passer à un état présentant une énergie supérieure. Le critère communément appliqué est issu de l'algorithme de Metropolis-Hastings (METROPOLIS et al., 1953) et s'appuie sur la distribution de Boltzmann : une transition vers un état d'énergie inférieure, i.e. diminuant la fonction objectif, sera toujours acceptée alors qu'une transition vers un état d'énergie supérieure, i.e. augmentant la fonction objectif, sera acceptée avec la probabilité

$$p = \exp(-\Delta E / T) \tag{4.12}$$

où ΔE correspond à la différence (positive) d'énergie apportée au système, i.e. l'amplitude de la détérioration de la fonction objectif, et T à la température courante.

Contrairement à d'autres méthodes d'optimisation, le recuit simulé autorise donc des transitions détériorant la fonction objectif, ce qui lui permet de s'extraire des minima locaux. On observe également que plus la température diminue, plus ces transitions seront difficilement acceptées. Ainsi, les premières itérations encourageront l'*exploration* de l'espace d'état avec une probabilité d'acceptation élevée, alors que les dernières itérations s'apparenteront davantage à une recherche locale où presque seules les transitions améliorantes seront acceptées, ce qui se traduit par l'*exploitation* d'un voisinage local et, finalement, une stabilisation du système et donc la convergence de l'algorithme.

Des preuves théoriques montrent que l'algorithme du recuit simulé est capable de converger vers *un* minimum global (ou *le*, s'il est unique) à partir de n'importe quel état initial, sous certaines conditions (HENDERSON et al., 2003). En particulier, tout état doit être atteignable depuis un autre en un nombre fini d'étapes (chaîne de Markov ergodique) et les schémas de chauffage et de refroidissement doivent suivre des hypothèses drastiques afin d'éviter une convergence prématurée vers un minimum local, hypothèses que l'on ne peut pas garantir en pratique. Il faut par exemple s'assurer que le système atteint l'équilibre thermodynamique correspondant à chaque valeur de température avant de diminuer cette dernière. Algorithmiquement, on tentera de s'en approcher en effectuant un certain nombre d'itérations pour chaque niveau de température (palier). La fonction de décroissance de la température est également un paramètre délicat. Bien que plusieurs schémas de refroidissement existent, la

décroissance géométrique est la plus couramment utilisée car elle offre un bon compromis entre la vitesse et la qualité de la convergence. À l'issue du palier i, la nouvelle température T_{i+1} pour le palier $i+1$ sera ainsi définie par

$$T_{i+1} \leftarrow \alpha T_i \tag{4.13}$$

avec habituellement $\alpha \in [0.97, 1[$. [6]

Le choix de la température initiale est aussi un paramètre important puisqu'elle doit permettre une exploration suffisamment large de l'espace d'état. On la détermine souvent de telle sorte qu'une transition détériorant la fonction objectif ait initialement une probabilité p_0 d'être acceptée, par exemple 70 %. En générant un certain nombre d'états aléatoires et leur voisinage, il est possible d'estimer sur les transitions non améliorantes la détérioration moyenne ΔE_{est} de la fonction objectif. En inversant l'équation 4.12, on peut alors estimer la température initiale T_0 :

$$T_0 = -\frac{\Delta E_{est}}{\ln p_0} \tag{4.14}$$

Enfin, il reste à définir un critère d'arrêt. Il serait en théorie possible d'attendre que la température devienne inférieure à un certain ϵ suffisamment petit : la probabilité d'accepter une transition non améliorante serait alors quasi nulle et la décroissance géométrique étant de plus en plus lente, on pourrait considérer que le voisinage local a été suffisamment exploité. Néanmoins, la pratique montre, pour des problèmes relativement bien structurés (modélisation et opérateur de voisinage pertinents, fonction objectif qui ne présente pas trop de pièges), qu'il n'est pas nécessaire d'attendre si longtemps puisque la convergence semble souvent acquise bien avant. Il serait par exemple plus pertinent de stopper l'algorithme dès que le système est resté parfaitement stabilisé sur les n derniers paliers de température (aucune transition améliorante, aucune transition non améliorante acceptée). Un autre critère d'arrêt, moins fort et contenu dans le premier, serait de stopper si les n derniers paliers n'ont pas permis de découvrir un nouvel état d'énergie inférieure (aucune transition améliorante). Pour avoir une garantie de performance dans le pire cas, on peut enfin allouer un temps maximal à l'algorithme et le combiner avec un autre critère d'arrêt.

Le pseudo-code du recuit simulé est donné dans l'algorithme 1. La figure 4.16 donne, pour trois état initiaux différents, trois photographies du système à différents instants, avec un schéma de refroidissement géométrique.

Le recuit simulé est particulièrement adapté aux problèmes dits sans dérivées et « boîte

[6] Il est possible de faire évoluer la longueur d'un palier (nombre d'itérations à température constante) et le facteur α au cours de la procédure, en fonction de la température, de la stabilité observée…

Algorithme 1 Recuit simulé pour la minimisation

```
1: procedure SIMULATED ANNEALING
2:     T ← <estimated temperature to accept deteriorating moves with probability p_0>
3:     x ← <initial random state>
4:     obj ← OBJECTIVE VALUE(x)
5:     while <stopping criteria not satisfied> do
6:         for i ← 1 to nb_iterations do
7:             x' ← RANDOM NEIGHBOR(x)
8:             obj' ← OBJECTIVE VALUE(x')
9:             ΔE ← obj' − obj
10:            if ΔE ≤ 0 or random number from U(0,1) ≤ exp(−ΔE/T) then
11:                x ← x'
12:                obj ← obj'
13:            end if
14:        end for
15:        T ← αT
16:    end while
17:    return x
18: end procedure
```

noire » dans lesquels la fonction objectif n'est pas explicite, sa valeur étant le résultat d'une phase de simulation. Lorsque la combinatoire associée au problème est élevée, il n'existe alors généralement pas de méthode exacte permettant de résoudre le problème en un temps raisonnable car celle-ci s'apparenterait, faute de connaissances, à une recherche exhaustive par force brute, souvent de complexité exponentielle. Dans de telles conditions, l'algorithme du recuit simulé est suffisamment générique pour pouvoir être implémenté dès lors que la modélisation de l'état permet de construire une structure de voisinage pertinente.

Comme la plupart des métaheuristiques de recherche globale, le recuit simulé ne donne aucune garantie quant au caractère local ou global du minima trouvé et seule l'expérience, par essais successifs, peut permettre d'apprécier la qualité de la solution. Une autre classe de métaheuristiques dites à population permettent de réduire ce risque de solution sous-optimale en manipulant plusieurs individus qui pourront converger vers des états différents, par diversification de la recherche. Les algorithmes génétiques, qui reposent sur des opérateurs de croisement, de mutation et de sélection en font partie et pourront être préférables à l'algorithme du recuit simulé selon la modélisation adoptée, le paysage de la fonction objectif, etc.

L'application de l'algorithme du recuit simulé à notre problème ne pose pas de problème, la fonction objectif (3.2) et l'opérateur de voisinage (3.1.3) ayant été définis. Concernant l'état initial, bien que nous aurions pu créer une décomposition aléatoire valide (i.e. sans coupes

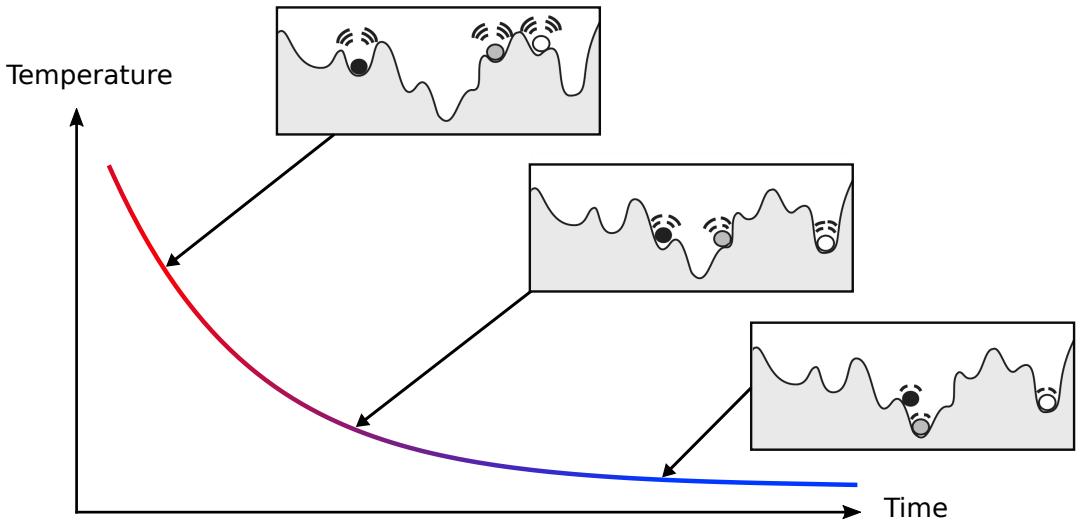


FIGURE 4.16 – Algorithme du recuit simulé (inspiré de : source inconnue)

L'algorithme est lancé en parallèle avec trois états initiaux différents représentés par les boules noire, grise et blanche qui évoluent sur le paysage de la fonction objectif. Autour de la boule est symbolisée l'amplitude de la détérioration qu'elle peut franchir avec une probabilité élevée, disons 80 %. Les boules noire et grise atteindront le minimum global alors que la boule blanche restera bloquée dans un minium local.

qui s'intersectent), nous choisissons d'utiliser le vecteur décision nul partout. À l'issue de l'optimisation, nous pouvons espérer obtenir la décomposition optimale *pour notre fonction objectif.* 7

3.4 *Génération des segments* 8

Une fois la décomposition optimale $(P_k)_k$ et les directions optimales associées $(\theta_k)_k$ connues, nous générons les segments d'acquisition (jusqu'à présent imaginaires) qui permettront de construire la trajectoire avec le GTSP présenté dans la section 2.

Les segments sont construits de manière indépendante pour chaque sous-champ P_k en recouvrant l'enveloppe du polygone de lignes orientées dans la direction θ_k et régulièrement espacées selon l'espacement imposé. Les segments sont ensuite créés en calculant l'intersection de ces lignes avec le champ ce qui présente l'avantage de fonctionner quelle que soit la forme du champ et des zones sans intérêt (figure 4.17).

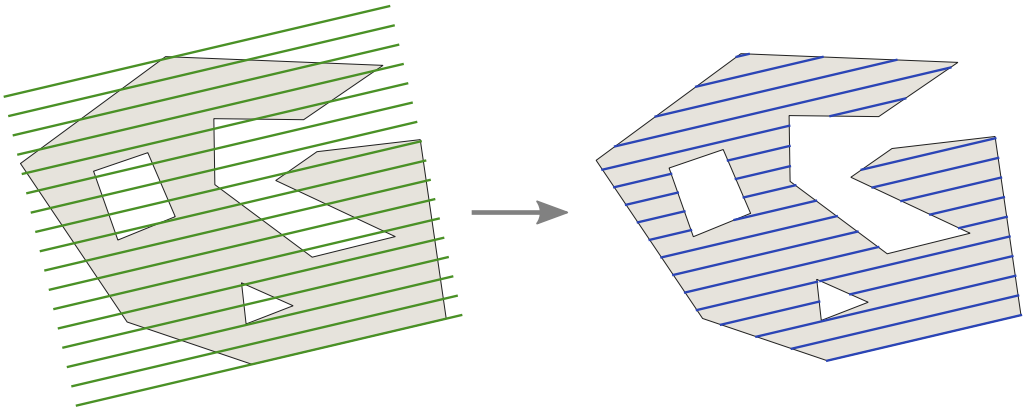


FIGURE 4.17 – Création des segments orientés dans la direction θ_k pour un polygone P_k

4. Extension à la planification en ligne

Cet algorithme de génération de plan de vol initial offrant des performances propices à une utilisation en ligne, nous choisissons de l'adapter et de l'intégrer au cœur de la planification en ligne.

À un instant donné, l'*état de la mission* est modélisé par une liste de segments qu'il reste à parcourir. La connexion de ces segments à l'aide du GTSP (9, section 2), dans un ordre optimisé et par des transitions de Dubins, définira le plan de vol. La replanification en ligne de la mission consistera in fine à supprimer, remplacer ou ajouter des segments dans la liste des segments d'observation et à recalculer le plan de vol optimal.

Une boucle de supervision, alimentée par les positions estimées et l'état courant de la carte (en couverture et qualité) construite par le module de SLAM, enverra des requêtes à la planification en ligne. Il existe différents types de requêtes qui nécessitent des actions différentes.

Mise à jour du vent ou de l'espacement entre les segments Une modification de l'environnement ou des paramètres de la mission ayant des conséquences directes sur la structure du plan de vol et/ou sa durée, il est préférable de replanifier l'ensemble de la mission. La liste des segments qu'il reste à parcourir est vidée et un nouveau plan de vol est alors généré (1 à 9), recouvrant la ou les zones qui n'ont pas encore été perçues d'après la carte actuelle.

(Re)visiter une zone De nouveaux segments recouvrant la zone sont générés (1 à 8, section 3) puis insérés dans la liste des segments de l'état de la mission. Le plan de vol est alors mis à jour (9).

Ajout d'une fermeture de boucle L'ajout d'un chemin permettant de fermer une boucle dans le SLAM correspond à l'insertion d'un ou plusieurs segments dans l'état de la mission puis à la mise à jour du plan de vol (9).

Idéalement, il faut prendre en compte l'intervalle de temps écoulé pendant les calculs et durant lequel le drone, toujours en mouvement, aura suivi son ancien plan de vol. La fenêtre d'anticipation – de l'ordre de quelques secondes – permettra donc par exemple, lors de la mise à jour du plan de vol, de ne pas considérer comme non couverte une portion de segment qui l'aura été durant la phase de calculs.

5 Résultats

Après quelques détails d'implémentation, nous présentons et analysons des exemples de plans de vols générés avec notre méthode. À l'heure où sont écrites ces lignes, seule la planification hors ligne est implémentée de bout en bout.

1. Implémentation

L'algorithme a été implémenté en C++ au sein de l'architecture ROS présentée au paragraphe 2.4.

Il utilise notamment les bibliothèques GDAL, pour lire et écrire des Shapefile et gérer les projections et GEOS, pour la géométrie algorithmique.

Tout au long du stage, un travail de profilage de code a été réalisé pour accélérer autant que possible les étapes coûteuses et appelées un grand nombre de fois comme :

- le découpage du champ à l'aide de coupes (étape 3) ;
- la fonction objectif estimant la durée du plan de vol (étape 4) ;
- la boucle du recuit simulé.

Toutes les décompositions du champ générées lors de l'optimisation sont mises en cache. Certaines opérations pour déterminer les trajectoires de Dubins optimales sont également mises en cache et l'intégration numérique permettant de prendre en compte le vent dans le calcul de la durée de la trajectoire n'est réalisée qu'une seule fois pour produire une table de valeurs approchées.

Les tests ont été lancés sur un ordinateur de bureau équipé d'un processeur Intel Core i7 quadricœur à 3.6 GHz et de 16 Go de RAM.

2. Paramètres

Nous utilisons les paramètres suivants pour configurer le recuit simulé :

- Probabilité initiale p_0 d'accepter une transition détérioriante : 70 %
- Frontière entre exploration et exploitation : < 20 % de transitions détérioriantes acceptées
- Nombre d'itérations par palier de température : 200 en exploration, 300 en exploitation
- Facteur α de décroissance de la température : 0.93 en exploration, 0.99 en exploitation
- Critère d'arrêt : aucun état améliorant la meilleure solution connue jusqu'à présent n'a été visité sur les 100 derniers paliers (soit 30 000 itérations) et 30 secondes maximum

Le solveur GLKH est paramétré ainsi :

- Configuration des heuristiques internes : par défaut
- Nombre maximum d'itérations : 2000

Enfin, certaines caractéristiques propres au drone ou à la mission seront conservées constantes dans tous les tests :

- Vitesse air v_a du drone : 15 m/s
- Rayon de virage minimal r : 40 m
- Champ de vision FOV de la caméra : 60°
- Altitude h : 100 m

3. Tests et analyses

Chaque champ est représenté avec ses coupes potentielles générées à l'étape 1 (en rouge) afin de donner un aperçu de l'espace de recherche et de la combinatoire associée. Sont également précisés :

- la surface du champ, en hectares ;
- son nombre d'arêtes, extérieures ou intérieures ;
- le nombre de coupes potentielles.

Chaque scénario de test est défini par :

- le recouvrement latéral ou *sidelap* imposé entre les différents segments ;
- la vitesse et la direction du vent.

Le plan de vol résultant sera représenté par la trajectoire obtenue à l'issue de l'algorithme, accompagné de ces informations :

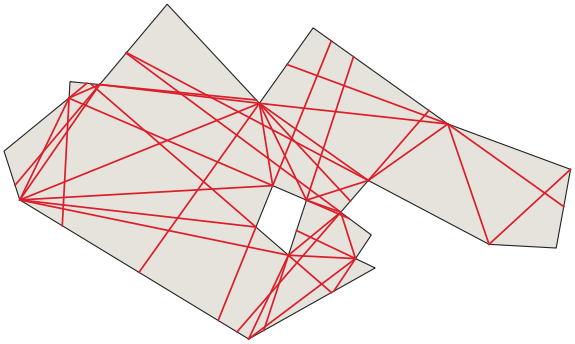
- nombre de segments : nombre de tronçons rectilignes dans la trajectoire ;
- durée réelle : durée de la trajectoire, que le GTSP aura cherché à minimiser à l'étape 9 ;
- durée estimée : il s'agit de la valeur de la fonction objectif 4 utilisée pour évaluer la décomposition dans la boucle du recuit simulé et entre parenthèses, l'erreur relative par rapport à la durée réelle ;
- temps de calcul : entre parenthèses, nous distinguerons le temps de calcul dû aux étapes 1 à 8 (recherche de la décomposition par la métaheuristique) et le temps de calcul dû au seul GTSP 9.

3.1 Influence du taux de recouvrement

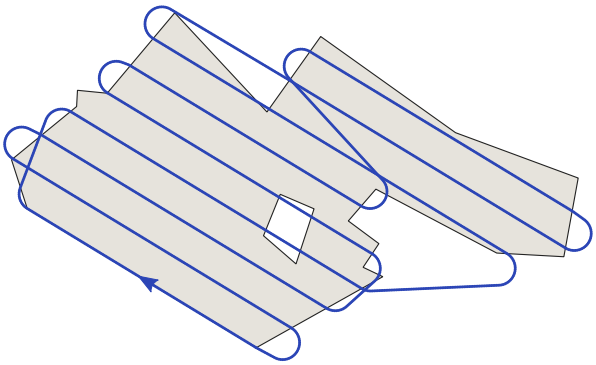
La figure 5.1 détaille les résultats obtenus sur un même champ pour trois taux de *sidelap* différents : 30 %, 60 % et 80 %. Comme attendu, plus le recouvrement augmente et plus les segments sont nombreux et rapprochés. Deux observations sont intéressantes.

Le temps de calcul dû à la recherche de la décomposition par notre métaheuristique est relativement stable (3 s, 3 s et 4 s). Il est légèrement supérieur lorsque le recouvrement augmente car la fonction objectif doit tester davantage de transitions possibles sur les arêtes (enjambement de segments). En revanche, le temps de calcul dû au seul GTSP augmente significativement, passant de près de 0 s à 2 s, ce qui s'explique par la complexité exponentielle en nombre de segments de l'espace de recherche des trajectoires.

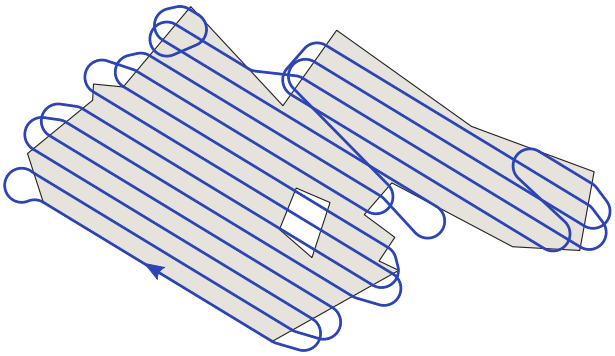
D'autre part, l'erreur relative entre la durée estimée par la métaheuristique et la durée de la trajectoire finale diminue lorsque le taux de recouvrement – et donc le nombre de segments – augmente, jusqu'à atteindre seulement 3 secondes. En effet, plus le nombre de segments augmente, plus le plan de vol sera structuré et régulier, rendant les « irrégularités » négligeables dans la durée totale. L'autre explication provient de notre approche statistique au moment d'écrire la fonction objectif. La durée de parcours des segments avait été estimée en fonction de l'aire du champ, or davantage de segments permet de mieux apprécier la forme du polygone et donc rendre compte de son aire. Enfin, plus il y a de segments venant buter contre une arête, plus nos prédictions quant aux formes des transitions se vérifient.



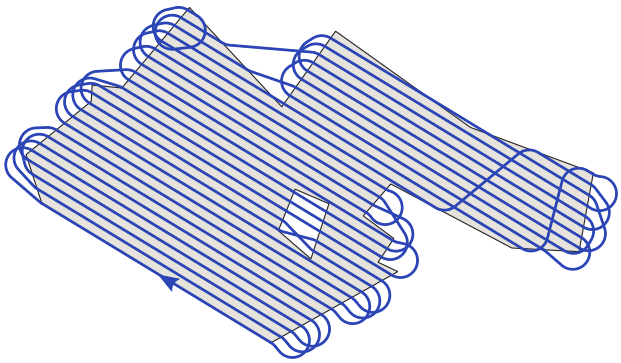
Surface : 51 ha
Nombre d'arêtes : 22
Nombre de coupes potentielles : 44



Sidelap : 30 % **Vent** : /
Nombre de segments : 12
Durée réelle : 624 s (10 min)
Durée estimée : 583 s (−6.69 %)
Temps de calcul : 3 s (3/0)



Sidelap : 60 % **Vent** : /
Nombre de segments : 21
Durée réelle : 1014 s (17 min)
Durée estimée : 991 s (−2.22 %)
Temps de calcul : 4 s (3/1)



Sidelap : 80 % **Vent** : /
Nombre de segments : 42
Durée réelle : 1931 s (32 min)
Durée estimée : 1934 s (+0.16 %)
Temps de calcul : 6 s (4/2)

FIGURE 5.1 – Influence du taux de recouvrement

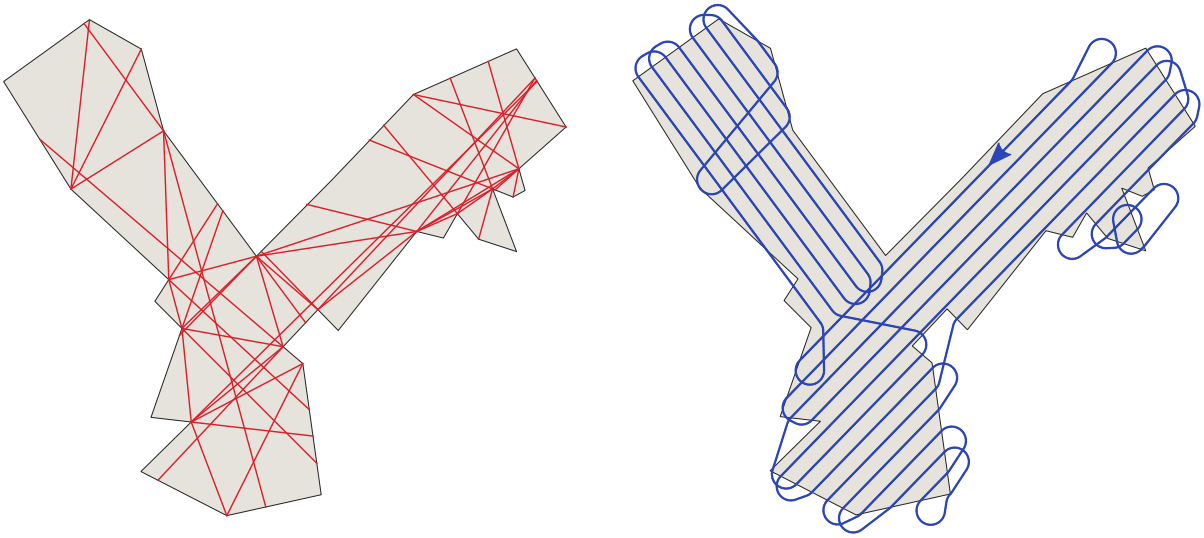
3.2 Influence du nombre de coupes potentielles

Si la complexité du GTSP est exponentielle en nombre de segments, la complexité de notre métaheuristique pour calculer la décomposition est elle exponentielle en nombre de coupes potentielles. Elle est donc bien moins sensible aux dimensions du champ qu'à sa forme. Le nombre de coupes potentielles augmente avec :

- la présence de zones sans intérêt, comme nous pouvons le voir très nettement sur le champ précédent (figure 5.1) ;
- le nombre d'arêtes extérieures, d'autant plus si elles créent de la concavité (e.g. le champ de la figure 5.2).

Sur ces champs, la métaheuristique de décomposition nécessite entre 3 et 7 s.

Sur des champs à la géométrie moins compliquée, comme ceux illustrés sur la figure 5.3, le nombre de coupes potentielles est limité et la décomposition se montre très rapide, de l'ordre de la seconde voire moins.

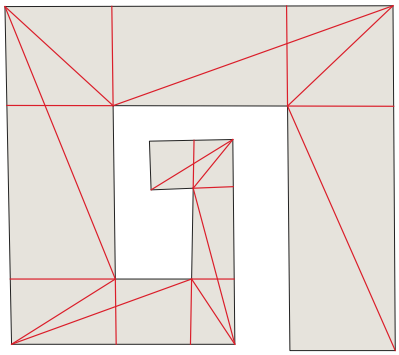


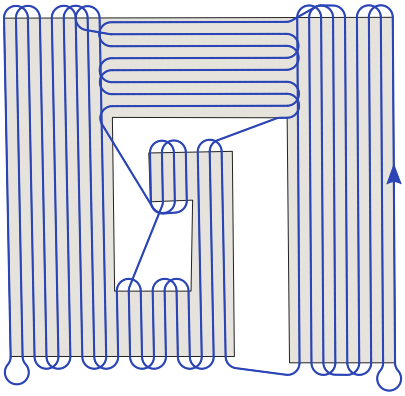
Surface : 78 ha
Nombre d'arêtes : 30
Nombre de coupes potentielles : 49

Sidelap : 60 %
Nombre de segments : 32
Durée réelle : 1516 s (25 min)
Durée estimée : 1565 s (+3.22 %)
Temps de calcul : 8 s (7/1)

Vent : /

FIGURE 5.2 – Champ présentant un grand nombre de coupes potentielles

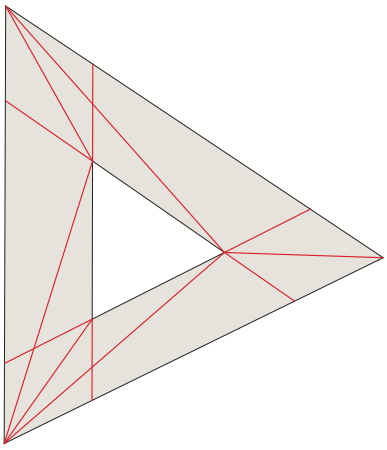


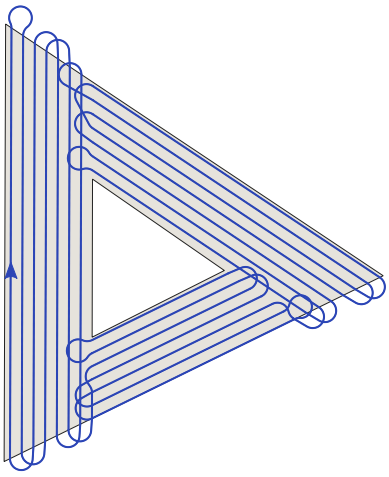


Surface : 118 ha
Nombre d'arêtes : 14
Nombre de coupes potentielles : 21

Sidelap : 65 %
Nombre de segments : 41
Durée réelle : 2411 s (40 min)
Durée estimée : 2433 s (+0.94 %)
Temps de calcul : 2 s (1/1)

Vent : /





Surface : 89 ha
Nombre d'arêtes : 6
Nombre de coupes potentielles : 12

Sidelap : 65 %
Nombre de segments : 23
Durée réelle : 1735 s (29 min)
Durée estimée : 1781 s (+2.63 %)
Temps de calcul : 2 s (0/2)

Vent : /

FIGURE 5.3 – Champs à la géométrie simple

3.3 Influence du vent

En observant l'équation 4.11, qui donne la durée d'un aller-retour sur un segment en fonction de son orientation et du vent, nous avions remarqué que cette durée est minimale lorsque le vent est perpendiculaire au segment et maximale lorsque le drone vole dans le vent. Cette hypothèse d'un parcours en allers-retours est au cœur de la fonction objectif et nous avions pour cette raison ajouté aux orientations possibles des segments la direction perpendiculaire au vent.

La figure 5.4 illustre quatre plans de vol générés en présence d'un vent venant du sud :

1. Le vent a une vitesse de 5 m/s et nous constatons, par rapport au cas sans vent (figure 5.3), que la décomposition a privilégié les cellules supérieure et inférieure, en raccourcissant les tronçons de trajectoire verticaux. Compte tenu de ce que nous venons de rappeler, cela n'est pas étonnant : la décomposition cherche du vent (quasi) traversier et fuit l'alignement avec le vent.
2. Avec un vent de 9 m/s, la fonction objectif décrète qu'il est désormais plus économe d'éliminer les segments verticaux en agrandissant la cellule inférieure, bien que le nombre de segments et donc de demi-tours augmente.
3. En poussant le vent à 12 m/s, la fonction objectif semble habitée par la recherche du vent traversier, la décomposition ne trouvant plus aucun intérêt à épouser les formes du champ alors que le nombre de demi-tours s'envole.
4. Dans ce scénario irréaliste où nous faisons tendre le vent vers la vitesse air du drone, les demi-tours deviennent si laborieux que le GTSP, cette fois, décide de traverser entièrement la zone sans intérêt.

Si tout cela semble cohérent, deux points sont à éclaircir.

Nous pouvons nous interroger sur l'hypothèse (que l'on constate très forte puisque c'est elle qui a dirigé la décomposition) selon laquelle la trajectoire optimale consisterait, pour n'importe quel champ, à le couvrir à l'aide de segments parcourus « statistiquement » en allers-retours. Nous avons essayé, pour ce champ, de forcer la cellule verticale à être couverte exclusivement en vent de dos, en utilisant les deux cellules obliques pour créer une spirale, en vain. Si cela aurait pu profiter à la cellule de gauche, les deux autres cellules auraient, elles, souffert du vent sur l'ensemble du parcours…

Enfin, comment expliquer que l'erreur relative entre la durée estimée et la durée post-GTSP ne fasse que croître lorsque nous augmentons la vitesse du vent ? Les durées des transitions

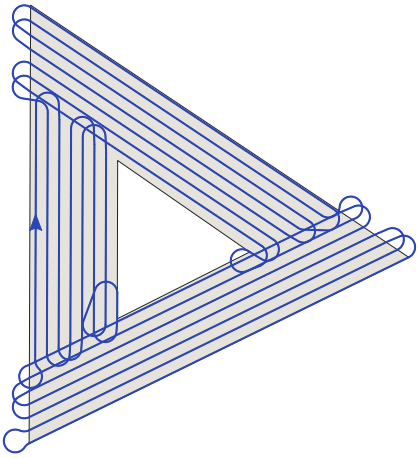
sont estimées dans la fonction objectif en considérant chaque arête individuellement et en prédisant les meilleures transitions dont elle sera le support. Avec un vent du sud, les meilleures transitions, dans notre cas, semblent être de connecter chaque segment à son voisin en se dirigeant vers le nord, toutes les transitions ayant lieu dans le vent. Nous ne nous préoccupons à aucun moment de « ce qu'il se fera sur les autres arêtes » et s'il sera possible de boucler le tour facilement. Dans notre cas, il manque dans l'estimation le long chemin de retour qu'il faudrait réaliser avec un vent de face pour revenir fermer le tour. Le calcul montre d'ailleurs que la durée de ce chemin correspond plus ou moins à l'ordre de grandeur de l'erreur constatée. Le GTSP a choisi l'autre alternative qui consiste à connecter un segment sur deux, ce qui lui permet de boucler le tour sans longue transition, bien que les demis-tours vers le sud soient coûteux.

3.4 Sous-estimation et sur-estimation

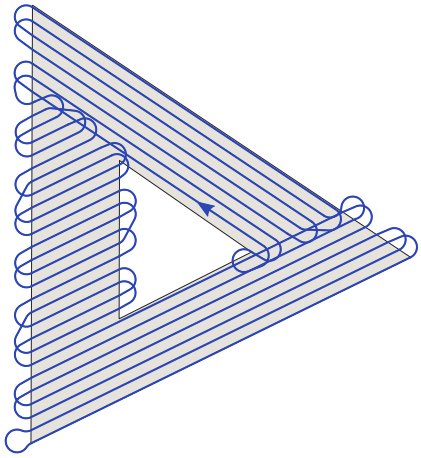
Il a été observé que la métaheuristique de décomposition sous-estime ou sur-estime toujours la durée réelle connue une fois les segments générés et le GTSP résolu. Cela ne serait pas problématique si l'erreur affectait de la même manière toutes les décompositions : l'ordre serait préservé. Or, il arrive que la fonction objectif juge une décomposition meilleure qu'une autre alors que c'est en réalité le contraire. L'algorithme peut par exemple être tenté d'ajouter une coupe, augmentant ainsi le nombre de polygones dans la décomposition, sans toutefois pouvoir en mesurer le réel impact en terme de transitions ; c'est notamment le cas de petits polygones.

Alors que nous possédons en cache toutes les décompositions calculées et les valeurs de la fonction objectif associées, il serait dommage de retourner une décomposition dont la durée a été sous-estimée au détriment d'une meilleure décomposition dont la durée aura elle été sur-estimée. Nous ajoutons alors un petit test : si la meilleure décomposition, aux yeux de la métaheuristique, comporte n polygones, nous calculons le GTSP sur la meilleure décomposition connue de $n, n-1, \ldots, 1$ polygone(s) et nous sélectionnons celle qui produit la durée réelle minimale. Nous pouvons en effet nous permettre de résoudre quelques instances de GTSP à l'issue de l'optimisation sans que cela ne pèse trop sur les performances.

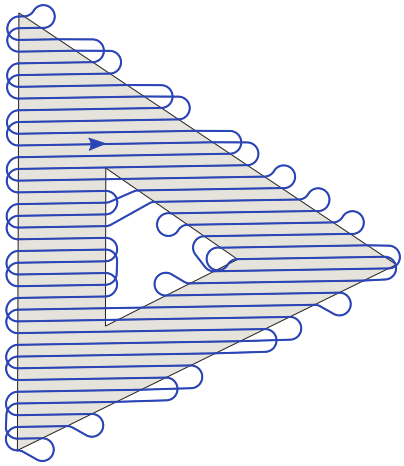
Enfin, notons que l'algorithme, relancé à plusieurs reprises sur un même champ, retourne presque toujours la même décomposition, ce qui nous rassure sur l'adéquation du modèle à la métaheuristique.



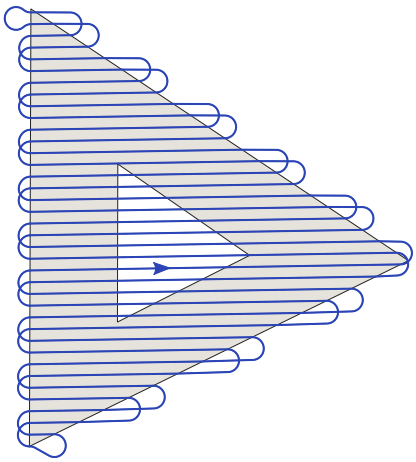
Sidelap : 65 % **Vent** : 5 m/s, du sud
Nombre de segments : 23
Durée réelle : 1992 s (33 min)
Durée estimée : 1892 s (−5.03 %)
Temps de calcul : 2 s (0/2)



Sidelap : 65 % **Vent** : 9 m/s, du sud
Nombre de segments : 35
Durée réelle : 2615 s (43 min)
Durée estimée : 2343 s (−10.42 %)
Temps de calcul : 3 s (0/3)



Sidelap : 65 % **Vent** : 12 m/s, du sud
Nombre de segments : 51
Durée réelle : 3682 s (61 min)
Durée estimée : 3165 s (−14.03 %)
Temps de calcul : 2 s (0/2)



Sidelap : 65 % **Vent** : 14.5 m/s, du sud
Nombre de segments : 51
Durée réelle : 10 595 s (176 min)
Durée estimée : 6844 s (−35.40 %)
Temps de calcul : 2 s (0/2)

FIGURE 5.4 – Influence du vent

6 Bilan et perspectives

Nous concluons ce rapport en dressant le bilan des contributions et en proposant des perspectives pour améliorer et étendre ce travail.

1. Contributions

Notre algorithme de planification hors ligne pour la cartographie d'un champ convexe ou concave, à l'aide d'un drone à voilure fixe, est divisé en deux étapes :

1. détermination d'une décomposition exacte du champ puis génération de segments ;
2. génération du plan de vol par connexion des segments avec le GTSP.

Zones sans intérêt Nous utilisons la flexibilité offerte par le GTSP pour modéliser des zones sans intérêt mais survolables. Celles-ci permettent potentiellement d'écourter la mission mais sont encore peu évoquées dans la littérature. Elles sont d'ailleurs parfois assimilées à des obstacles infranchissables, ce qui peut être pire encore.

Clarification sur la question de l'optimalité La notion d'*optimalité* d'une trajectoire s'entend toujours au regard d'un critère d'optimisation défini par l'auteur et d'un ensemble de contraintes qui peuvent émerger de la méthode elle-même. Nous avons expliqué pourquoi minimiser le nombre de segments n'était pas satisfaisant. En présence de vent, minimiser la seule durée des transitions ne suffit pas non plus. Les approches les plus prometteuses sont donc celles qui cherchent à minimiser la durée totale de la trajectoire. À notre connaissance,

les auteurs qui s'y intéressent (e.g. COOMBES et al., 2018) se contraignent à des trajectoires de type boustrophedon où l'ordre de parcours des segments est déjà préétabli.

Dans notre cas, la trajectoire sera toujours in fine le résultat de l'application d'un GTSP sur un ensemble de segments. À moins de modifier la deuxième étape, nous n'avons alors pas d'autre choix que d'accepter de transformer les spécificités du GTSP en contraintes sur le plan de vol (i.e. « trajectoire fermée qui doit parcourir chaque segment une et une seule fois, sans pouvoir rien visiter d'autre »). En contrepartie, nous obtenons la garantie que, pour cet ensemble de segments, le plan de vol obtenu sera optimal en temps. À titre de comparaison, si nous raisonnons en termes de contraintes et de contreparties, l'intérêt du boustrophedon classique est, pour une mission comme la nôtre, difficilement défendable. La manière dont sont générés les segments constitue également une contrainte intériorisée par le plan de vol. Dans notre cas, ils sont obtenus en utilisant une décomposition exacte et donc une partition du champ où les cellules et les segments ne peuvent par exemple pas se superposer. Il faudrait alors se demander dans quelle mesure toutes ces contraintes implicitement introduites dans le plan de vol peuvent nous éloigner du plan de vol « réellement » optimal.

Anticipation des choix du GTSP Théoriquement, dès que nous contraignons le plan de vol à être le résultat d'un GTSP et que nous cherchons le plan de vol optimal sous cette contrainte, la première étape doit nécessairement correspondre à la recherche des segments menant au GTSP optimal. Autrement dit, il serait sous-optimal de traiter la première étape de manière indépendante, en cherchant des segments tout en oubliant le problème du GTSP sous-jacent. C'est pourtant ce que font BOCHKAREV et SMITH, 2016 en utilisant, en guise de première étape, une approche gloutonne qui oublie le GTSP à venir dans la seconde étape. Formulé autrement encore, le GTSP est, en soi, une bonne solution pour connecter des segments ; mais quitte à contraindre le plan de vol à avoir la forme imposée par un GTSP, autant orienter la recherche des segments vers l'optimal pour le GTSP. Notre contribution consiste donc ici à proposer des moyens d'anticiper certaines réactions du GTSP (couverture ou non d'une zone sans intérêt, type de transitions sur les arêtes) dans le but d'écrire une première étape qui soit cohérente avec la seconde et dont la motivation soit logiquement fondée et tournée vers la recherche de l'optimal.

Utilisation d'une métaheuristique Notre principale contribution concerne le calcul de la décomposition exacte à l'aide d'une métaheuristique alors que l'état de l'art repose sur des techniques gloutonnes ou exhaustives. Celles-ci sont souvent soit rapides mais sous-*optimales*, soit exhaustives mais extrêmement lentes.

Le modèle que nous présentons est alimenté par un jeu de coupes potentielles, encodées par un vecteur binaire qui se prête bien à l'optimisation. Bien que la fonction objectif soit spécialement conçue pour le GTSP, il est tout à fait possible de l'adapter à un tout autre problème de recherche de décomposition.

L'utilisation d'une métaheuristique comme le recuit simulé est pertinente à plusieurs égards :

- L'espace de recherche des décompositions est souvent de complexité exponentielle et aucune méthode exhaustive ne supporte (efficacement) le changement d'échelle alors qu'une métaheuristique peut s'orienter rapidement vers des décompositions intéressantes.
- Utiliser une métaheuristique permet également de penser son problème de manière globale et de réaliser une vraie optimisation, contrairement à certaines approches gloutonnes qui suivent des procédures, parfois compliquées, dont le fonctionnement risque de nous échapper à tout moment.

Enfin, notre algorithme affiche des performances de 1 à 3 s pour les champs simples et jusqu'à 10 s environ pour les champs les plus complexes, ce qui le rend directement comparable avec certaines approches gloutonnes qui, elles, ne font pas d'optimisation.

Performances et utilisation pour la planification en ligne Les performances de notre algorithme tranchent avec l'état de l'art et rendent donc envisageable son intégration dans une boucle de planification en ligne. Nous espérons avoir des contributions à présenter à ce sujet dans un futur proche.

2. Perspectives

Notre premier objectif reste à court terme l'intégration de l'algorithme pour la planification en ligne.

Avant de l'étendre, nous souhaiterions également mieux comprendre le modèle en réalisant davantage de tests. La première idée serait de remplacer, le temps du test, la fonction objectif du recuit simulé par la résolution même du GTSP. Les résultats, qui seraient disponibles après quelques dizaines d'heures de calcul, fourniraient sans doute des données intéressantes de comparaison entre la durée estimée et la durée réelle qui nous permettraient, peut-être, d'améliorer notre fonction objectif tout en nous offrant un jeu de données utilisable pour l'y confronter.

2.1 Améliorations bienvenues

Nous listons quelques améliorations plus ou moins importantes que nous n'avons pas eu le temps de traiter.

Simplifier la géométrie d'entrée Comme nous l'avons vu, une zone sans intérêt peut engendrer une grande quantité de coupes potentielles. Il pourrait être intéressant d'essayer d'identifier les zones sans intérêt qui, suffisamment petites, seront quoi qu'il arrive toujours couvertes par la trajectoire et de les réintégrer dans le polygone du champ. De la même manière, si l'utilisateur suit trop précisément les contours de sa parcelle au moment de dessiner le polygone, le nombre d'arêtes risque d'augmenter et il faudrait pouvoir détecter ces cas pour réduire la complexité en simplifiant, mais sans trahir, la géométrie d'entrée.

Garantir le recouvrement Jusqu'à maintenant, nous avions passé sous silence l'un des aspects du problème de recouvrement. Afin de garantir que chaque zone du champ sera perçue par deux segments d'observation, il peut être nécessaire de prolonger les segments au-delà des limites du champ et d'ajouter de nouveaux segments quand il le faut (figure 6.1). À première vue, ce problème est compliqué et si la figure peut nous inciter à construire une solution basée sur l'enveloppe convexe du champ, par exemple, il existe toujours des scénarios où cela serait sous-optimal.

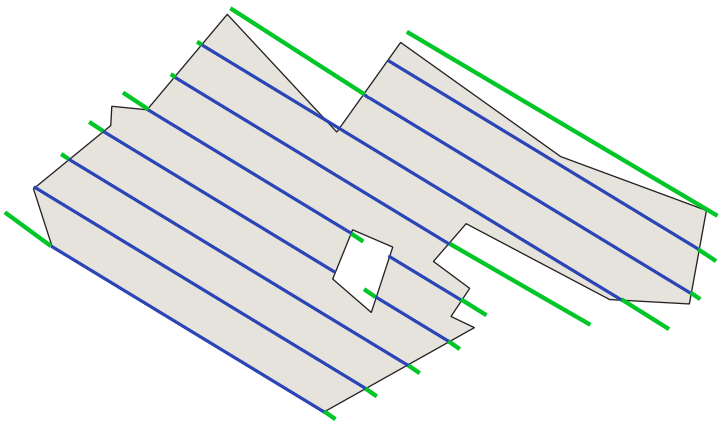


FIGURE 6.1 – Garantir le recouvrement en étendant et ajoutant des segments

Étudier d'autres types de coupes Les coupes potentielles issues de la triangulation sont rarement utilisées par la décomposition *optimale* retournée par l'algorithme. En effet, ces coupes « en diagonale » seraient souvent responsables de transitions coûteuses. Cela s'explique

également par le fait que les coupes issues de prolongements d'arêtes possèdent, elles, une orientation qui est déjà présente dans la géométrie, ce qui limite grandement les transitions le long de cette coupe si cette orientation venait à être choisie par les cellules voisines. Forts de ce constat, nous pourrions ajouter de nouvelles coupes potentielles dont les directions correspondent aux directions des arêtes du champ et mener une expérimentation pour en mesurer les conséquences. Une autre amélioration bienvenue consisterait à détecter puis supprimer une coupe très proche d'une autre coupe.

Diminuer l'erreur sur l'estimation de la durée Comme nous l'avons vu, l'erreur sur l'estimation de la durée croît notamment avec la vitesse du vent. Il s'agirait donc de mieux prendre en compte le vent ou, considérant que toutes les décompositions souffriront du même biais, de ne rien faire. Le risque, à terme, est de ne plus pouvoir écrire une fonction objectif qui reste à la fois pertinente et performante. Par exemple, nous nous sommes toujours interdit pour le moment de générer des segments dans la fonction objectif et nos calculs se basent sur des considérations d'aires, etc. Cela pourra-t-il toujours suffire ?

2.2 Pistes pour étendre le problème

Enfin, il serait souhaitable de pouvoir cartographier plusieurs champs disjoints dans un même secteur, si l'autonomie du drone le permet, pour le compte d'une seule personne ou d'une coopérative. Nous pourrions pour cela calculer des segments d'observation pour chacun des champs et résoudre le GTSP sur l'ensemble des segments si les champs sont suffisamment proches, ce qui pourrait permettre une utilisation optimisée des ressources.

Nous pourrions également réfléchir à la possibilité d'utiliser une flotte de drones pour optimiser la durée de la mission, ce qui nous obligerait à déterminer si et dans quelle mesure les algorithmes présentés dans ce rapport peuvent être réutilisés.

Bibliographie

1. ACAR, E. U., CHOSET, H., RIZZI, A. A., ATKAR, P. N. et HULL, D. (2002). « Morse Decompositions for Coverage Tasks ». In : *I. J. Robotics Res.* 21, p. 331-344.
2. BOCHKAREV, S. et SMITH, S. L. (2016). « On minimizing turns in robot coverage path planning ». In : *2016 IEEE International Conference on Automation Science and Engineering (CASE)*, p. 1237-1242.
3. BOCHTIS, D., VOUGIOUKAS, S. et GRIEPENTROG, H. W. (2009). « A mission planner for an autonomous tractor ». In : *Transactions of the ASABE* 52.5, p. 1429-1440.
4. CHAVES, S. M., KIM, A., GALCERAN, E. et EUSTICE, R. M. (2016). « Opportunistic sampling-based active visual SLAM for underwater inspection ». In : *Autonomous Robots* 40.7, p. 1245-1265.
5. CHOSET, H. et PIGNON, P. (1998). « Coverage Path Planning : The Boustrophedon Cellular Decomposition ». In : *Field and Service Robotics*. Sous la dir. d'A. ZELINSKY. London : Springer, p. 203-209.
6. COOMBES, M., FLETCHER, T., CHEN, W.-H. et LIU, C. (2018). « Optimal Polygon Decomposition for UAV Survey Coverage Path Planning in Wind ». In : *Sensors* 18.7.
7. DUBINS, L. E. (1957). « On Curves of Minimal Length with a Constraint on Average Curvature, and with Prescribed Initial and Terminal Positions and Tangents ». In : *American Journal of Mathematics* 79.3, p. 497-516.

8. G MCGEE, T., SPRY, S. et KARL HEDRICK, J (août 2005). « Optimal Path Planning in a Constant Wind with a Bounded Turning Rate ». In : *Collection of Technical Papers - AIAA Guidance, Navigation, and Control Conference* 5.

9. GALCERAN, E. et CARRERAS, M. (déc. 2013). « A Survey on Coverage Path Planning for Robotics ». In : *Robot. Auton. Syst.* 61.12, p. 1258-1276.

10. HELSGAUN, K. (2015). « Solving the equality generalized traveling salesman problem using the Lin–Kernighan–Helsgaun Algorithm ». In : *Mathematical Programming Computation* 7.3, p. 269-287.

11. HENDERSON, D., JACOBSON, S. H. et JOHNSON, A. W. (2003). « The Theory and Practice of Simulated Annealing ». In : *Handbook of Metaheuristics*. Sous la dir. de F. GLOVER et G. A. KOCHENBERGER. Boston, MA : Springer US, p. 287-319.

12. HUANG, W. H. (2001). « Optimal line-sweep-based decompositions for coverage algorithms ». In : *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*. T. 1, 27-32 vol.1.

13. JIN, J. et TANG, L. (2010). « Optimal Coverage Path Planning for Arable Farming on 2D Surfaces ». In : *Transactions of the ASABE* 53.1, p. 283-295.

14. KIRKPATRICK, S., GELATT, C. D. et VECCHI, M. P. (1983). « Optimization by Simulated Annealing ». In : *Science* 220.4598, p. 671-680.

15. LI, Y., CHEN, H., ER, M. J. et WANG, X. (2011). « Coverage path planning for UAVs based on enhanced exact cellular decomposition method ». In : *Mechatronics* 21.5, p. 876-885.

16. METROPOLIS, N., ROSENBLUTH, A. W., ROSENBLUTH, M. N., TELLER, A. H. et TELLER, E. (1953). « Equation of State Calculations by Fast Computing Machines ». In : *The Journal of Chemical Physics* 21.6, p. 1087-1092.

17. NAM, L. H., HUANG, L., LI, X. J. et XU, J. F. (2016). « An approach for coverage path planning for UAVs ». In : *2016 IEEE 14th International Workshop on Advanced Motion Control (AMC)*, p. 411-416.

18. NEWMAN, P. et HO, K. L. (2005). « SLAM-Loop Closing with Visually Salient Features ». In : *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, p. 635-642.

19. NOON, C. E. et BEAN, J. C. (1993). « An Efficient Transformation Of The Generalized Traveling Salesman Problem ». In : *INFOR : Information Systems and Operational Research* 31.1, p. 39-44.

20. NY, J., FERON, E. et FRAZZOLI, E. (2012). « On the Dubins Traveling Salesman Problem ». In : *IEEE Transactions on Automatic Control* 57.1, p. 265-270.

21. PAULL, L., THIBAULT, C., NAGATY, A., SETO, M. et LI, H. (2014). « Sensor-Driven Area Coverage for an Autonomous Fixed-Wing Unmanned Aerial Vehicle ». In : *IEEE Transactions on Cybernetics* 44.9, p. 1605-1618.

22. TECHY, L. et WOOLSEY, C. A. (2009). « Minimum-time path planning for unmanned aerial vehicles in steady uniform winds ». In : *Journal of guidance, control, and dynamics* 32.6, p. 1736-1746.

23. XU, A., VIRIYASUTHEE, C. et REKLEITIS, I. (2014). « Efficient complete coverage of a known arbitrary environment with applications to aerial operations ». In : *Autonomous Robots* 36.4, p. 365-381.

24. YAKOUBI, M. A. et LASKRI, M. T. (2016). « The path planning of cleaner robot for coverage region using Genetic Algorithms ». In : *Journal of Innovation in Digital Ecosystems* 3.1, p. 37-43.

25. YU, X., ROPPEL, T. A. et HUNG, J. Y. (2015). « An optimization approach for planning robotic field coverage ». In : *IECON 2015 - 41st Annual Conference of the IEEE Industrial Electronics Society*, p. 4032-4039.