AVERTISSEMENT

Ce document est le fruit d'un long travail approuvé par le jury de soutenance et mis à disposition de l'ensemble de la communauté universitaire élargie.

Il est soumis à la propriété intellectuelle de l'auteur. Ceci implique une obligation de citation et de référencement lors de l'utilisation de ce document.

D'autre part, toute contrefaçon, plagiat, reproduction illicite encourt une poursuite pénale.

Contact : ddoc-theses-contact@univ-lorraine.fr

LIENS

Code de la Propriété Intellectuelle. articles L 122. 4
Code de la Propriété Intellectuelle. articles L 335.2- L 335.10
http://www.cfcopies.com/V2/leg/leg_droi.php
http://www.culture.gouv.fr/culture/infos-pratiques/droits/protection.htm

UNIVERSITÉ DE LORRAINE

École doctorale IAEM Lorraine

Systèmes Cyber-Physiques autonomes et communicants en milieux inconnus : Application à l'exploration par robots mobiles.

THÈSE

présentée et soutenue publiquement le 1er décembre 2021

pour l'obtention du

Doctorat de l'Université de Lorraine

(mention informatique)

par

Virgile Daugé

Composition du jury

Présidents :	Jean-François Couchot	Pr, Université de Bourgogne Franche-Comté
Rapporteurs :	Philippe Bonnifait	Pr, Université de Technologie de Compiègne
	Frédéric Le Mouël	Pr, INSA Lyon, Université de Lyon
Examinateurs :	Franck Gechter	Mcf, Université de Technologie de Belfort-Montbéliard
	Isabelle Guérin Lassous	Pr, Université Claude Bernard Lyon 1
Directeurs de thèse :	Sylvain Contassot-Vivier	Pr, Université de Lorraine, Nancy
	Laurent Ciarletta	Mcf, Université de Lorraine, Nancy

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Mis en page avec la classe thesul.

Sommaire

Table des figures vii

Liste des tableaux xi

Liste des codes source xiii

Partie I Présentation du sujet 3

Chapitre 1
Introduction

1.1 Après-mine . . . 7
1.2 Systèmes Cyber-Physiques . . . 9
1.2.1 Définition . . . 9
1.2.2 Domaines d'applications . . . 10
1.3 Environnement inconnu . . . 12
1.4 Autonomie : définition . . . 14
1.5 Perception, les raisons de l'échec . . . 16
1.5.1 Stimulations, Données ou informations ? . . . 18
1.6 Synthèse . . . 19
1.7 Plan de la thèse . . . 20

Chapitre 2
État de l'art

2.1 Définition du problème . . . 21
2.1.1 Caractérisation du positionnement . . . 21
2.1.2 Vers quels modèles ? . . . 23
2.2 Quels outils ? . . . 27
2.2.1 Multi-niveaux . . . 27
2.3 Estimateurs (*front-end*) . . . 29

2.3.1 Odométrie Inertielle . . . 29
2.3.2 Odométrie Visuelle . . . 29
2.3.3 Odométrie Lidar . . . 33
2.3.4 Problèmes récurrents . . . 41
2.4 Prendre en compte l'incertitude (*back-end*) . . . 41
2.4.1 Filtrage Bayésien . . . 42
2.4.2 Lissage (*smoothing*) . . . 43
2.5 Où contribuer ? . . . 45

Partie II Contribution : positionnement **47**

Chapitre 3
NAPS : Un système de positionnement nomade

3.1 Détermination de la nouvelle approche . . . 51
3.1.1 Vers un maillage de positionnement . . . 54
3.1.2 Vers une mesure en coopération active . . . 55
3.2 NAPS . . . 57
3.2.1 Réalisation du maillage temporel . . . 57
3.2.2 Détermination de la position globale . . . 58
3.3 Implémentation proposée . . . 60
3.3.1 Architecture de NAPS . . . 60
3.3.2 Architecture de validation . . . 62
3.3.3 Un développement difficile . . . 63
3.4 Expérimentations . . . 65
3.4.1 Protocoles expérimentaux . . . 65
3.5 Résultats . . . 67
3.5.1 Analyse qualitative . . . 67
3.5.2 Analyse quantitative . . . 70
3.6 Conclusion . . . 71

Chapitre 4
Un cas d'usage : la cartographie 3D

4.1 Vers une solution de SLAM complète . . . 75
4.2 Lidar, nuage, et recalage . . . 75
4.2.1 D'un ensemble de nuages vers un modèle . . . 75
4.2.2 Opération de recalage . . . 76

4.3 Architecture du système de cartographie . . . 77
4.4 Logique de recalage et d'amélioration des nuages . . . 79
4.4.1 Mécanisme d'association temporelle . . . 79
4.4.2 Filtrage en fonction de la vitesse . . . 79
4.4.3 Diminution du bruit statique . . . 79
4.4.4 Filtrage des éléments du système . . . 80
4.5 Expérimentations et Résultats . . . 80
4.5.1 Étude qualitative . . . 80
4.5.2 Étude quantitative . . . 84
4.6 Améliorations . . . 84
4.6.1 ICP et raffinement Bayésien . . . 84
4.6.2 Vers d'autres marqueurs . . . 86
4.6.3 Contrôle robotisé du processus d'exploration . . . 88
4.7 Conclusion . . . 89

Partie III Contribution : planification **91**

Chapitre 5
Génération de trajectoire

5.1 Lecture de la carte . . . 95
5.2 Création de zones restreintes . . . 96
5.3 Géométrisation des zones restreintes . . . 97
5.4 Calcul du diagramme de Voronoï . . . 100
5.5 Filtrage des frontières . . . 103
5.6 Structuration des segments . . . 103
5.7 Organisation en allées . . . 107
5.8 Plus court chemin . . . 109
5.9 Construction de la solution . . . 113
5.10 Ré-échantillonage des points . . . 114
5.11 Lissage de la trajectoire . . . 116
5.12 Approche discrète . . . 121
5.12.1 Données du problème pour le calcul de la trajectoire du robot . . . 121
5.12.2 Processus de calcul du chemin . . . 121
5.13 Constat et évolutions . . . 123

Chapitre 6
Calcul des placements des Phares

6.1 Modélisation du problème local . . . 128
6.2 Cas idéal . . . 131
6.3 Sentiers adaptés à l'environnement . . . 132
6.4 Réduction de l'espace de solutions . . . 133
6.5 Vers une heuristique d'orientation des phares . . . 136
6.6 Un modèle des occlusions de l'environnement . . . 138
6.7 Fonction d'évaluation de solution locale . . . 139
6.8 Optimisation sur les sentiers . . . 142
6.9 Construction d'une solution globale . . . 142
6.10 Prendre en compte plus de solutions . . . 145
6.11 Une meilleure solution locale . . . 148
6.12 Échantillonage multi-échelle . . . 148
6.13 Une meilleure solution globale ? . . . 151
6.14 Approche alternative discrète . . . 153
6.14.1 Données du problème . . . 153
6.14.2 Résultats attendus . . . 153
6.14.3 Processus itératif de calcul . . . 153
6.15 Constat et évolutions . . . 155

Conclusions et perspectives **159**

1 Contexte . . . 159
2 Contributions . . . 159
3 Perspectives . . . 161
3.1 Amélioration des propositions . . . 161
3.2 Étendre le champ d'application . . . 161

Bibliographie **165**

Table des figures

1.1 Carte de France des sites miniers fermés 8
1.2 Mine de calcaire, amménagée pour acceuil de visiteurs. Petr Brož 9
1.3 Exemple de système Cyber-Physique 10
1.4 Redondance possible de l'environnement. Florent Ruyssen, ... 14
1.5 Photo d'un pillier de mine de calcaire. Père Igor 15
1.6 Un chercheur en informatique lorrain à récemment bu 20cl de café sans se brûler 17

2.1 Convention sur les axes de rotation, par Tubezlob 22
2.2 Visualisation d'une grille d'occupation 3D, réalisée avec Octomap. 24
2.3 Amers humains, illustration générée avec le jeu Townscaper, de Oskar Stålberg . 24
2.4 Graphe de facteurs, stockant les informations spatiales et leur probabilités associées 25
2.5 Vue 2D d'un nuage de points généré par nos algorithmes 26
2.6 Flux de traitement des données dans un système SLAM classique 28
2.7 Détection d'amers avec ORB sur deux images du même environnement 31
2.8 Détection des correspondances entres les amers des deux images 32
2.9 Point de vue de la première image projeté sur la seconde calculée par RANSAC et Five-point 33
2.10 Nuage de points issu d'un unique scan lidar de mon bureau, visualisation avec Open3D 34
2.11 Deux scans lidars d'une mine de calcaire simulée 35
2.12 Itérations successives de l'algorithme ICP classique 37
2.13 Modèlisation probabilite de l'incertitude 42
2.14 Filtre à particules appliqué au jeu de plateau *Captain Sonar* 44

3.1 Carte summérienne gravée dans une tablette d'argile, Penn Museum objet CBS13885. 52
3.2 Borne du RGF au sommet du Mont Ventoux, Arbautjc 53
3.3 Processus de maillage temporel, les flèches oranges représentant des mouvement d'éléments 55
3.4 Photographie des éléments du système de suivi local 57
3.5 Processus de construction du maillage temporel 58
3.6 Poses des éléments du système exprimées dans un repère local 59
3.7 Arbre de transformations après déplacement des phares 59
3.8 Arbre de transformations après déverrouillage 60
3.9 Architecture générale de la solution logicielle proposée 62
3.10 Ajout du groupe d'acquisition de la vérité terrain 63
3.11 Vue du dessus de la trajectoire GM2 68
3.12 Vue du dessus de la trajectoire RM2 69

3.13 Erreur de translation . . . 70
3.14 Erreur de rotation . . . 70
3.15 Distribution des erreurs en translation par longueur de segment . . . 72
3.16 Distribution des erreurs en rotation par longueur de segment . . . 72

4.1 Équipement du lidar avec un marqueur . . . 76
4.2 Arbre de transformations permettant de déterminer la pose du lidar . . . 77
4.3 Interfaçage du module de cartographie dans NAPS . . . 78
4.4 Vue du dessus du pire cas d'erreur de lacet . . . 81
4.5 Vue du dessus du modèle reconstruit, jeu *Bureaux* . . . 82
4.6 Vue du dessus du modèle du couloir du bâtiment C du LORIA . . . 83
4.7 Vue de biais du modèle du couloir du bâtiment C du LORIA . . . 83
4.8 Modèle obtenu par ICP et Factor-graph, à partir des poses fournies par NAPS . . . 85
4.9 Modèle obtenu par ICP et Factor-graph, sans les poses fournies par NAPS . . . 86
4.10 Diminution de l'erreur avec un plus grand capteur . . . 87
4.11 Calcul de la pose avec un *meta-marqueur* . . . 88

5.1 Exemple de carte sous forme d'image représentant une maison . . . 96
5.2 La même carte, une fois les obstacles dilatés . . . 97
5.3 Procédure de détermination des contours. . . . 98
5.4 Contours obtenus avec l'agorithme marching squares . . . 99
5.5 Diagramme de Voronoï brut obtenu, avec zoom . . . 101
5.6 Segments frontières des régions d'influences . . . 102
5.7 Ensemble des segments frontières valides, avec zoom . . . 104
5.8 Graphe brut obtenu, centré sur un coin de la carte initiale. . . . 107
5.9 Graphe d'allées obtenu à la suite de l'opération d'étiquetage . . . 108
5.10 Visualisation des distances minimales à la source . . . 113
5.11 Visualisation du chemin optimal reconstruit . . . 115
5.12 Détails sur la trajectoire brute . . . 116
5.13 Vue des sous-chemins ré-échantillonnés . . . 117
5.14 Trajectoires obtenues en faisant varier l'intensité du lissage . . . 119
5.15 Trajectoires obtenues en faisant varier l'intensité du lissage . . . 120
5.16 Carte des distances aux obstacles. . . . 122
5.17 Chemin initial (en jaune) et chemin final (en bleu). . . . 123

6.1 Modèles représentant la zone couverte par un phare, θ variant. . . . 129
6.2 Détermination de la zone de tolérance . . . 130
6.3 Comparaison des largeurs de bandes . . . 131
6.4 Sentiers idéaux, représentés dans l'environnement . . . 132
6.5 Sentiers adaptés à l'environnement, trajectoire sous-échantillonnée . . . 134
6.6 Sentiers adaptés calculés en utilisant tous les points . . . 135
6.7 Deux sections distinctes générées indépendamment . . . 136
6.8 Determination de l'orientation du phare . . . 138
6.9 Seconde heuristique de calcul de l'orientation du phare . . . 139
6.10 Visualisation du modèle "zones d'occlusions" de l'environnement . . . 140
6.11 Visualisation de l'algorithme d'agrandissement du secteur virtuel . . . 141
6.12 Tests de visibilité par intersection de triangles . . . 143
6.13 Scores obtenus pour un section arbitraire . . . 144

6.14 Succession d'étapes d'optimisations locales constituant une solution globale . . . 145
6.15 Solution globale obtenue à partir des sentiers . . . 146
6.16 Génération d'une surface de solutions depuis deux points distincts . . . 147
6.17 Échantillonnage régulier et optimisation locale . . . 149
6.18 Visualisation de l'échantillonage progressif . . . 150
6.19 Succession d'étapes locales, avec résolution faible pour visualisation . . . 151
6.20 Solution globale obtenue à partir des sentiers . . . 152
6.21 Première recherche de position du phare. La couleur rouge indique les contours interdits des murs. Les points blancs sont les points interpolés du chemin. La zone de recherche est définie par le disque partiel autour de la position initiale du robot de référence. Le cercle rose est la position trouvée. Les segments roses indiquent l'orientation de l'ouverture du phare. . . . 154
6.22 Résultat obtenu avec l'approche discrète . . . 156

Liste des tableaux

1.1 Comparaison des environnements inhérents aux différentes applications 12

3.1 Jeux de données collectés durant les expérimentations 67
3.2 RMSE de translations et d'orientation de l'estimation fournie par NAPS 71

4.1 Jeux de données produits 80

5.1 Caractéristiques d'algorithmes simple-source statiques de plus court chemin . . . 109
5.2 Temps d'exécution minimaux observés 124
5.3 Caractéristiques de la machine de test 124

6.1 Évolution du nombre de solutions en fonction de la discrétisation d'angle 137
6.2 Comparatifs en coûts et performance pour les différentes approches. 152

Liste des codes source

5.1 Peuplements possibles de notre graphe 106
5.2 Implémentation de Dijkstra par tas binaire minimum 112
5.3 Reconstruction du chemin obtenu 114

Remerciements

Tout d'abord, je tiens à remercier Sylvain pour sa disponibilité et la richesse de l'expérience humaine que nous avons partagée durant ces années de thèse. Je remercie aussi Laurent pour son co-encadrement, qui à travers un lourd travail prospectif, a rendu ce projet possible.

Je tiens à remercier également l'ensemble des membres de l'équipe SIMBIOT, au sein de laquelle j'ai effectué ces travaux. Bien plus que des collègues, ils sont rapidement devenus des amis. Les rencontres effectuées dans le cadre plus large du laboratoire se sont également révélées indispensables à la conduction d'une recherche au long cours. La richesse et la fraîcheur des idées partagés lors des *pauses cafés* ont en effet eu un fort impact positif sur le déroulement de ma thèse. Je remercie également *Caro*, notre barista préférée, pour le soutien moral apporté tout au long de mon passage au LORIA.

Je remercie Adrien Guénard pour le soutien apporté autant par le *Créativ' Lab* que par ces précieux conseils techniques.

Ces années ont été ponctuées de phases d'enseignements plus où moins prononcées, durant lesquelles j'ai eu l'occasion de rencontrer de nombreux universitaires et étudiants. Je tiens à remercier l'équipe pédagogique et administrative de la *Faculté des Sciences et Technologies* pour son accueil chaleureux lors de cette dernière année d'ATER.

Je remercie ma famille et mes amis proches, qui ont su me soutenir dans les pires moments traversés ces dernières années. Ce sont en effet des choses simples —les BBQs, l'escalade, les soirées jeux— partagées avec les gens que j'aime qui m'ont permis de surmonter les périodes de renoncement. Sans eux, et particulièrement sans le soutien sans faille de Carla, il ne m'aurait en aucun cas été possible de mener cette aventure à son terme. Merci à vous.

Première partie

Présentation du sujet

Table des matières

Chapitre 1
Introduction

1.1 Après-mine . . . 7
1.2 Systèmes Cyber-Physiques . . . 9
1.2.1 Définition . . . 9
1.2.2 Domaines d'applications . . . 10
1.3 Environnement inconnu . . . 12
1.4 Autonomie : définition . . . 14
1.5 Perception, les raisons de l'échec . . . 16
1.5.1 Stimulations, Données ou informations ? . . . 18
1.6 Synthèse . . . 19
1.7 Plan de la thèse . . . 20

Chapitre 2
État de l'art

2.1 Définition du problème . . . 21
2.1.1 Caractérisation du positionnement . . . 21
2.1.2 Vers quels modèles ? . . . 23
2.2 Quels outils ? . . . 27
2.2.1 Multi-niveaux . . . 27
2.3 Estimateurs (*front-end*) . . . 29
2.3.1 Odométrie Inertielle . . . 29
2.3.2 Odométrie Visuelle . . . 29
2.3.3 Odométrie Lidar . . . 33
2.3.4 Problèmes récurrents . . . 41
2.4 Prendre en compte l'incertitude (*back-end*) . . . 41
2.4.1 Filtrage Bayésien . . . 42
2.4.2 Lissage (*smoothing*) . . . 43
2.5 Où contribuer ? . . . 45

Chapitre 1

Introduction

Ce manuscrit décrit les recherches effectuées dans le cadre de ma thèse, dont l'objectif est de lever un maximum de verrous théoriques et technologiques, qui rendent actuellement difficile la réalisation de systèmes cyber-physiques autonomes d'exploration et de cartographie. Dans le cadre du projet *Grone* dans lequel s'inscrivent ces travaux, des difficultés particulières ont été identifiées dans la surveillance de mines abandonnées à des fins d'entretien et de protection de la population. Nous avons donc choisi ce fil conducteur pour guider nos recherches, dans le but de répondre à une problématique réelle, et ainsi d'avoir un impact concret sur le monde.

Avant de rentrer dans le vif du sujet, nous allons prendre le temps de définir les contours du domaine abordé, qui traverse différents champs de la recherche. Nous commencerons par décrire le concept de systèmes cyber-physiques en dressant un bref historique. Nous verrons ensuite les nombreux domaines d'applications, tous en plein essor, de ces systèmes.

Vient enfin la problématique centrale de la thèse, l'autonomisation des systèmes Cyber physiques mobiles.

Par la suite, nous définirons ce qu'est un environnement inconnu, et les contraintes que cela implique. Cela permettra de cerner au mieux les problématiques auxquelles nous avons dû faire face durant ces années de thèse.

1.1 Après-mine

L'après-mine est un terme faisant référence à la fois à l'évolution d'un site d'activité minière après la fin de son exploitation, et à l'étude des impacts —sociaux, environnementaux, économiques et sécuritaires— de ce site. La législation de nombreux pays, dont la France, prévoit une analyse de risques et la mise en place de mécanismes compensatoires avant toute fermeture de sites miniers. Cependant, ces législation contraignantes ne sont entrées en vigueur que récemment, il existe donc de nombreux sites dont la fermeture n'a fait l'objet ni de contrôles, ni de précautions particulières. La figure 1.1 présente l'ensemble des sites miniers fermés connus, recensés par le cadastre minier gouvernemental *Camino*[1]. Chacun de ces sites présente des risques d'effondrements, d'inondations, de pollution, de remontées de gaz, où encore d'explosions. On comprend alors que ces sites sont une source de périls importants pour la population.

1. Le cadastre minier numérique ouvert : https://camino.beta.gouv.fr/

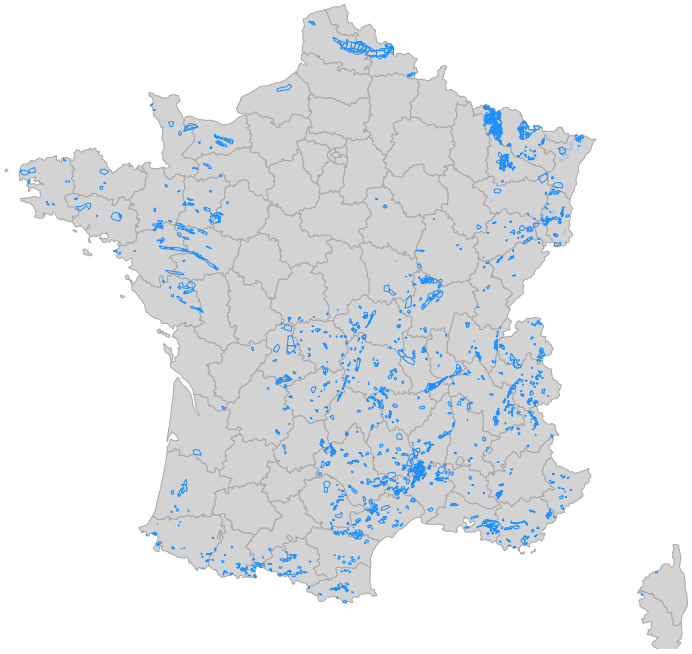

FIGURE 1.1 – Carte de France des sites miniers fermés

Afin d'atténuer ces menaces, l'INERIS (L'Institut national de l'environnement industriel et des risques) a développé une méthodologie d'évaluation des risques [Christian Franck, Romuald Salmon, Christophe Didier, Yves Paquette,]. La phase préliminaire de relevé d'informations prévue nécessite l'intervention de nombreux opérateurs spécialisés pour la collecte des données de terrain. Or, au vu de l'ampleur de la tâche à accomplir —rien que dans le bassin Mosellan, 13000 puits ont été recensés, et seulement 8000 d'entre eux sont approximativement localisés— cette dépendance à des experts chevronnées est un frein majeur à l'identification et la maîtrise de ces phénomènes dangereux.

Ainsi, l'emploi de solutions robotiques autonomes a émergé comme solution au problème de passage à l'échelle des solutions de veille actuelles. De tels systèmes permettraient de réduire les coûts et le temps nécessaire à l'analyse de risques pour un site, permettant logiquement de couvrir plus rapidement l'ensemble des territoires concernés. C'est ce contexte spécifique —missions d'exploration et de surveillances de mines— qui a dirigé nos recherches, principalement par l'apport de contraintes opérationnelles fortes, pour lesquelles les systèmes existants n'étaient pas adaptés. La figure 1.2 permet d'apprécier ce contexte, en particulier une mine de calcaire qui est toutefois équipée —notamment en luminaires— spécialement pour l'accueil de visiteurs. Le travail que nous avons effectué durant ces années de recherche a donc pour objectif de lever les divers verrous théoriques et techniques qui s'opposent au déploiement de systèmes autonomes dans des environnements souterrains, qui à bien des égards s'éloignent des milieux où les robots sont classiquement mis en oeuvre.



FIGURE 1.2 – Mine de calcaire, amménagée pour acceuil de visiteurs. Petr Brož

1.2 Systèmes Cyber-Physiques

1.2.1 Définition

Un système Cyber-Physique est un système informatique, dans lequel des éléments physiques —des capteurs, des actionneurs, des robots— sont contrôlés et supervisés par des algorithmes informatiques. Ces systèmes on été développés et reposent sur l'intégration transparente entre algorithmes et systèmes physiques, sur le couplage fort entre ces éléments, prévus dés la conception pour fonctionner de concert.

La figure 1.3 illustre un exemple de système Cyber-physique, dans lequel des robots, partiellement autonomes —nous reviendrons sur ce terme plus tard— évoluent sur une autre planète. Ces robots comportent des composants matériels : des moteurs, des caméras, des antennes, des unités de calculs… mais également des composants logiciels : des boucles de contrôle pour les moteurs, le traitement de l'image des caméras produisant des modèles de l'environnement, les protocoles de communication pour échanger des informations avec la terre, et enfin, une simulation permettant à des opérateurs —ou des algorithmes— de superviser et de planifier les prochaines étapes de la mission. Il devient alors évident que l'interconnexion de ces composants réels et "virtuels" est une condition *sine qua non* du fonctionnement de ces systèmes complexes. En effet, si l'on prend l'exemple de l'antenne et du protocole, leur développement est souvent réalisé conjointement. Une antenne est conçue pour un protocole de communication particulier, et ce même protocole ne peut souvent fonctionner que sur des antennes spécialisées. Cette interdépendance explique par exemple la nécessité de changer de téléphone pour pourvoir utiliser les derniers protocoles comme la 5G.

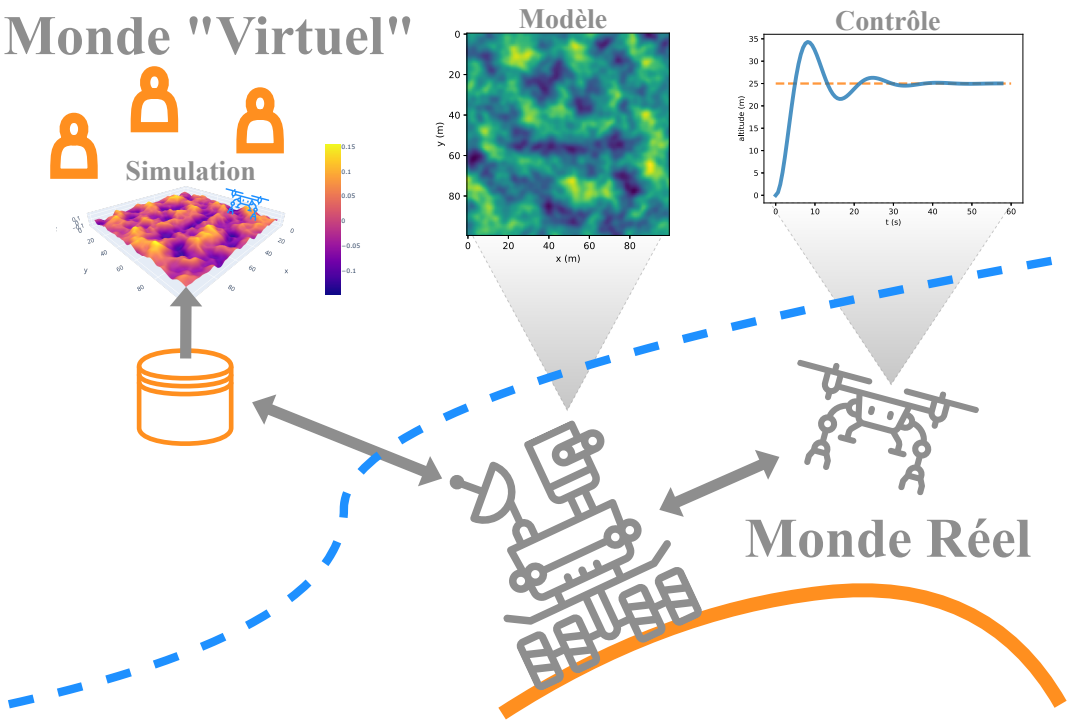

FIGURE 1.3 – Exemple de système Cyber-Physique

Téléphone qui est maintenant devenu un système cyber-physique, depuis qu'on l'affuble de l'adjectif *intelligent*. Il est d'ailleurs difficile de définir une limite ferme entre leurs précurseurs, les systèmes embarqués, et les systèmes cyber-physiques. L'accent est toutefois mis sur les capacités d'interaction entre les différents éléments du système, agissants plus comme un réseau collaboratif que comme des entités indépendantes les unes des autres. Un exemple du monde réel est le *Jardin distribué* (Distributed Robot Garden [Sanneman et al., 2015]) du MIT, où une équipe de robots collaborent pour entretenir des plants de tomates. Cette application utilise un réseau de capteurs distribué, où chaque plante est équipée de capteurs permettant le suivi de ses paramètres vitaux. Des robots permettent quant à eux d'effectuer des opérations de *manutention* et de récolte des données.

Par rapport à ce contexte, notre travail de thèse se focalise sur l'analyse et le traitement des données issues des capteurs pour planifier la mission donnée indépendemment du mode de locomotion des robots (roulant, volant,…).

1.2.2 Domaines d'applications

Il existe de nombreux domaines d'application des systèmes cyber-physiques, exploitant les capacités nouvelles apportées par ces derniers comme la robustesse et précision (multiplicité des capteurs), la réactivité (présence accrue dans l'environnement, il est finalement possible d'avoir le don d'ubiquité), et les caractéristiques qui en découlent : fiabilité et efficacité.

Nous présentons ici une brève sélection des utilisations faites des systèmes cyber-physiques :

Smart grids Ou réseaux électriques intelligents composés de capteurs et d'actionneurs, permettant le contrôle et l'optimisation de la production et de la distribution d'énergie en temps réel, parfois sur des territoires étendus CITE. Ces réseaux se manifestent par l'équipement des grandes infrastructures de l'énergie, mais également dans le quotidien de la population par le déploiement des compteurs dits intelligents.

Supervision médicale Domaine allant de la simple collecte de données à la prédiction, basée sur ces dernières, des interventions nécessaires. La chirurgie à distance, où deux robots —parfois situés dans différentes localités— coopèrent pour effectuer un acte médical. Un chirurgien, humain, manipule une machine à une extrémité du système, qui communique ces informations à l'autre machine, reproduisant les gestes avec une précision salvatrice.

Systèmes de contrôle industriels Bien que déjà bardés de robots et capteurs en tout genre, l'industrie actuelle ne fait qu'aborder les systèmes cyber-physiques, qui seront les piliers de l'industrie du futur plus communément nommée par le terme tapageur *Industrie 4.0*. Le concept se base sur le pilotage (et plus seulement le contrôle) des lignes de production par les informations tirées des données collectées à tous les niveaux de la chaîne de production et d'approvisionnement.

Pilote automatique Il existe de très nombreux systèmes de pilotages automatiques, de trains, de métros, d'engins de chantiers, etc… Le plus célèbre de tous, n'est autre que le pilote automatique d'un avion, qui depuis sa première implémentation en 1912 —un gyroscope et un *horizon artificiel*, contrôlant hydrauliquement les gouvernes de direction et de profondeur— n'a cessé d'intégrer de nouveaux éléments améliorants ses capacités et sa fiabilité. De nos jours, en plus des informations issues des capteurs embarqués dans l'avion, les pilotes automatiques utilisent les informations communiquées par les autres avions, ainsi que par des stations au sol, comme des positions, la météo, l'encombrement aérien etc…

Voiture "autonome" Sujet phare des dernières années, cristallisant les débats sur les questions de responsabilités accordables à des machines, les voitures autonomes circulent depuis maintenant quelques années au milieu des infrastructures humaines. L'Europe, plus conservatrice sur ces questions, investit dans l'équipement des infrastructures publiques de capteurs communicants, permettant de transmettre des informations supplémentaires aux utilisateurs de la route, ainsi qu'évidemment aux futures voitures autonomes, qui formeront alors un gigantesque système cyber-physique avec ces infrastructures.

Robotique mobile Enfin, après avoir vu divers domaines, concentrons nous sur celui qui fera l'objet du reste de ce document, la robotique. Bien que précurseur dans les systèmes embarqués, et systèmes cyber-physiques par essence, l'intégration de toujours plus d'informations et d'algorithmes eux mêmes de plus en plus complexes, ouvre la voie vers une nouvelle ère industrielle et sociale. En effet, La formation de systèmes multi-agents, groupes de robots —homogènes ou hétérogènes— coopérant pour remplir une tâche, ou un ensemble de tâches, permet de réaliser de plus en plus de missions et ce d'une manière de plus en plus fiable.

Un système cyber-physique peut donc avoir de nombreux visages, mais il est toujours basé sur l'interaction forte entre éléments physiques et algorithmiques. La tendance actuelle repose sur l'intégration de toujours plus d'éléments —ou d'agents— afin de doper la réactivité, la précision, la robustesse des systèmes ainsi proposés. De plus en plus d'applications sont désormais réalisables grâce à l'impressionnante croissance observée ces dernières années à la fois des capacités et de la fiabilité de ce type de systèmes.

1.3 Environnement inconnu

Nous avons listé précédemment des exemples d'utilisation de systèmes cyber-physiques, aussi nombreux que variés. Nous n'avons pas en revanche abordé un paramètre essentiel qui varie très fortement entre ces applications : leur contexte. Nous utiliserons le terme **environnement** pour désigner ce contexte, il est en effet plus approprier pour désigner la réalité dans laquelle doivent évoluer les différents agents. Il existe de nombreuses façons de caractériser ces environnements, mais nous nous concentrerons sur la notion de **maîtrise**, c'est-à-dire à quel point les propriétés de l'environnement sont connues.

Je vous invite à une petite expérience, qui consiste à classer les environnements des domaines d'applications listés dans la section précédente 1.2.2, dans l'ordre décroissant de la maîtrise de l'environnement. De manière à placer l'application dont l'environnement est le mieux connu en première position et en dernière position celle dont l'environnement est le plus incertain.

Le tableau 1.1 présente une rapide évaluation de ces contextes selon des critères subjectifs. Le premier étudie la disponibilité d'un lien de communication dédié, moins susceptible d'être perturbé que le réseau internet, par exemple. Le second critère concerne la présence de spécialistes humains lors de la mission à effectuer. C'est un critère fort car ces opérateurs garantissent souvent de nombreuses propriétés. Ils vérifient que le matériel est en place, connecté, règlent la température, la luminosité etc… Autre point important, la présence d'autres agents —robots ou humains— susceptibles d'interférer avec la mission. En effet, ces éléments dynamiques aux comportements non prévisibles complexifient énormément l'accomplissement d'une tâche donnée. Je peux par exemple citer mon chat *Némo* pour qui mon clavier devient l'endroit le plus confortable de la maison dès que je m'installe pour rédiger. Bien que son comportement soit prévisible, il conserve sa propriété handicapante.

Table 1.1 – Comparaison des environnements inhérents aux différentes applications

Application	Médical	Smart Grids	Pilote	Indus.	Voiture	Robot
Communication dédiée	✓					
Spécialistes humains	✓					
Accès restreint	✓	✓				
Équipements pré-installés	✓	✓	✓	✓		
Équipements étalonnés	✓	✓	✓	✓		
Infrastructures dédiées	✓	✓	✓	✓	✓	
Éléments ext. coopérants			✓			

D'autres facteurs entrent en ligne de compte, comme le fait que les équipements soient pré-installés et étalonnés pour répondre à une problématique précise. Ainsi, un ampèremètre dans une station électrique est installé précisément là où il doit effectuer ses mesures. De plus, les grandeurs physiques à mesurer sont connues et limitées à une plage de valeurs pour lesquelles le capteur est adapté. Cette contrainte peut paraître être une évidence et la base de tout système complexe. Cependant, l'exemple du *vol 501* d'*Ariane 5*, qui a pris fin brutalement à cause d'un capteur à qui on a demandé de lire des valeurs pour lesquelles il n'était pas prévu démontre le contraire. Dans ce cas, des vérifications auraient dues être faites en simulation avant décollage. Il n'est toutefois pas toujours possible de vérifier ces propriétés, notamment lorsque l'on manque

de connaissances sur l'environnement où va se dérouler l'expérience. Cette méconnaissance empêche également de réaliser un étalonnage sur place, étant donné que sans **vérité terrain**, il n'est pas possible d'ajuster les valeurs produites par un capteur en fonction de l'erreur, qui est elle même inconnue.

Enfin, les *éléments extérieurs coopérants* signifie simplement dans le cas de l'auto-pilote que tous les éléments perturbateurs —ici les autres avions— transmettent en permanence leurs informations de navigation aux avions proches. Il est ainsi plus aisé de prévoir et d'éviter une éventuelle collision.

Ce classement peut être discuté, mais l'essentiel est de comprendre que l'environnement est un facteur prépondérant dans le développement d'un système cyber-physique. De plus, nous ne disposons que d'une fenêtre —un modèle— de l'environnement, construit à la fois à partir des données issues des capteurs et des informations que l'on en tire, et des connaissances *a priori* disponibles. Or ces connaissances à priori sont difficiles à garantir en pratique, c'est pourquoi de nombreuses approches en robotiques se basent sur des hypothèses sur l'environnement. Ces hypothèses représentent un risque fort pour le système. En effet, dans le cas où le monde réel ne répond pas à ces contraintes, le comportement du système défini n'est pas non plus garanti. C'est la principale raison pour laquelle des éléments dont le comportement a été validé en simulation —modèle simplifié d'un environnement réel, qui ne sera probablement jamais capable d'en imiter toutes les spécificités— agissent parfois de manière imprévisible lorsqu'ils sont confrontés au monde réel. On comprend alors que la robustesse d'un système est inversement proportionelle au nombre et à la force des hypothèses posées sur l'environnement dans lequel il devra évoluer.

En robotique, de nombreuses hypothèses sont classiquement utilisées, dont :
— Exposition lumineuse suffisante et constante
— Présence de repères visuels/sensoriels
— Variété permettant de distinguer deux localités différentes
— Un espace libre ou l'agent peut évoluer
— Milieu permettant la propagation d'ondes

Or, il s'avère que ces contraintes sont loin d'êtres garanties dans le monde réel. L'existence même d'une horloge solaire démontre l'invalidité de la première (son principe de fonctionnement se base sur le déplacement de l'ombre formée par l'horloge). La présence de repères distinguables de manière unique par une combinaison capteur/algorithme est également difficile à garantir, bien qu'en pratique et pour un environnement cible bien délimité il est possible de paramétrer spécifiquement ce couple capteur/algorithme de manière satisfaisante. Quant à la redondance de l'environnement, la simple vue d'un aqueduc ou d'un pont suffit à se convaincre qu'il existe des environnements qui se répètent spatialement. La photo 1.4, n'est que l'une de ces structures qui ponctuent nos paysages. Ces propriétés sont difficiles à garantir, ce qui fait de l'hypothèse de leur présence une hypothèse forte.

L'hypothèse sur la capacité du milieu à diffuser des ondes est une hypothèse faible (facile à garantir en pratique) lorsque l'on évolue sur terre ou dans les airs. C'est heureux car sans cela, aucune communication sans fil possible, et de nombreux capteurs ne fonctionneraient pas non plus.

Lorsque l'on s'intéresse à une problématique d'exploration d'environnements inconnus — comme c'est notre cas— il est par définition impossible de garantir les propriétés de ce dernier.



FIGURE 1.4 – Redondance possible de l'environnement. Florent Ruyssen, CC BY SA

Par conséquent, et dans le but d'obtenir un système le plus robuste possible, il faudrait ne faire aucune hypothèse lors de la conception de ce dernier. Toutefois, aucun environnement n'est totalement inconnu, nous savons par exemple que sur terre, il y a une forte probabilité pour que nous soyons soumis à la gravité. Nous verrons plus tard que cette propriété évidente peut se révéler utile.

Si l'on transpose les contraintes de notre problématique de surveillance de mines abandonnées, il devient rapidement évident qu'il faudra se passer de lumière. De plus, comme on peut le constater sur la figure 1.5, les piliers de supports —régulièrement espacés pour soutenir le poids des voûtes— forment un environnement très redondant. Ces spécificités nous contraignent donc à nous écarter des hypothèses fortes classiquement adoptées, ne faisant alors que l'hypothèse de l'espace libre et de la propagation des ondes. Ces contraintes beaucoup plus fortes sur le système à concevoir nous poussent à chercher un nouveau paradigme d'exploration autonome.

1.4 Autonomie : définition

> **Myth 4 : Autonomous systems are autonomous.** Strictly speaking, the term "autonomous system" is a mis-nomer. No entity—and, for that matter, no person—is capable enough to be able to perform competently in every task and situation. On the other hand, even the simplest machine can seem to function "autonomously" if the task and context are sufficiently constrained.
> — J.M. Bradshaw, R.R. Hoffman, M. Johnson, and D.D. Woods

> **Mythe 4 : Les systèmes autonomes sont autonomes.** À proprement parler, le terme "système autonome" est inapproprié. Aucune entité, ni aucune personne d'ailleurs, n'est suffisamment capable pour accomplir avec compétence toutes les tâches et ce dans toutes les situations. D'un autre côté, même la machine la plus simple peut sembler fonctionner de manière "autonome" si la tâche et le contexte sont suffisamment contraints.
> — J.M. Bradshaw, R.R. Hoffman, M. Johnson, and D.D. Woods - *traduit de l'anglais*

Nous avons, par plusieurs fois déjà, évoqué la notion d'autonomie. Ce concept bien que familier en apparence, ne fait pas consensus quand à ce qu'il représente. Parfois durée de fonctionnement, parfois indépendance voire liberté, ce terme à divers usages et il convient d'en



FIGURE 1.5 – Photo d'un pillier de mine de calcaire. Père Igor

préciser les contours.

Nul doute que ce concept que nous cherchons à capturer est multidimensionnel, nous allons néanmoins nous concentrer sur deux axes qui nous paraissent essentiels : celui de l'**auto-suffisance** et celui de l'**auto-détermination**. L'auto-suffisance est entendue au sens de la capacité à effectuer une tâche donnée sans intervention extérieure. L'auto-détermination quant à elle représente du coté système la capacité à générer ses propres objectifs, et du coté humain la liberté que l'on accorde au système. Ainsi, on attend d'un système autonome qu'il ait une autonomie —au sens de l'auto-suffisance— la plus importante possible. En revanche, il n'est pas toujours souhaitable d'accorder à un système une grande liberté décisionnelle, particulièrement dans le cas où son auto-suffisance est réduite, il pourrait alors générer des objectifs qu'il ne sera pas capable de mener à bien.

Parfois, quand le risque est trop grand et pour une question de responsabilité, la décision est prise de limiter la liberté d'auto-direction d'un système, malgré son auto-suffisance. C'est le cas par exemple pour les voitures autonomes, où un humain doit chaperonner le système en permanence, et être prêt à reprendre le contrôle. Autre exemple, les premiers *Mars rovers* de la NASA, qui bien que tout à fait capables de générer leurs propres objectifs, ont été minutieusement dirigés par une équipe dédiée, la NASA ayant quantifié le coût d'une défaillance comme trop important. On parle dans ce cas de sous-utilisation d'un système par manque de confiance en ce dernier.

Enfin, et de manière contre-intuitive, un robot doté d'une grande auto-direction et d'une grande auto-suffisance n'est pas forcément une bonne chose. Il ne s'agit pas ici de considérations alarmistes de type *soulèvement des machines* mais de contraintes bien réelles. En effet, comme le souligne [Bradshaw et al., 2013], un tel robot doit être également capable de communiquer de manière à être compris par les éléments —humains ou non— qui évoluent dans le même univers que lui. Ainsi la prédictibilité d'un système est un critère fort qui détermine la pertinence de l'autonomie —entendue dans les deux axes— d'un système.

1.5 Perception, les raisons de l'échec

Après avoir défini les contours du terme autonomie, nous allons voir comment y parvenir. La première étape, et certainement toujours la plus complexe à ce jour, est d'être capable d'observer le contexte dans lequel on évolue. De percevoir l'environnement qui nous entoure. Cette tâche, que nous êtres vivants effectuons jours après jours peut paraître simple tellement elle nous est naturelle. Il est d'ailleurs assez frustrant de ne pas arriver à reproduire dans un système un comportement adaptatif pourtant si commun dans le vivant. Cela permet néanmoins d'aborder l'incroyable beauté de l'une des multiples capacités du monde biologique.

Nous n'en avons pas conscience, mais notre capacité à percevoir notre environnement est phénoménale. Autant au sens de la quantité d'informations de qualité que nous en tirons, que de notre capacité à les traiter pour adapter nos actions en conséquence. Nous ne faisons pas que percevoir notre environnement, nous le connaissons, nous le comprenons même. Il va sans dire que la robotique est loin, très loin d'égaler la performance et l'efficacité de la cognition humaine.

Afin de mieux apprécier cette capacité incroyable dont vous disposez, je vous propose une

autre expérience. Choisissez un objet quelconque dans votre environnement direct, et munissez vous d'un papier et d'un crayon. Listez maintenant toutes les informations que vous pouvez tirer de cet objet. Maintenant félicitez-vous, vous êtes plus performant qu'un système robotique ne le sera probablement jamais. Vous pouvez visualiser le résultat de cet exercice pratiqué en 5 minutes sur ma tasse de café ce matin 1.6. Vous noterez la diversité des informations relevées et leur pertinence : par exemple, la forme cylindrique fermée à la base, est immédiatement reconnue comme un contenant, fait qu'un algorithme établirait péniblement, surtout s'il doit traiter de nombreux objets différents. Ici les informations les plus difficiles à obtenir sont très probablement les notions de temporalité et de causalité. Ce petit exercice n'a qu'un but, démontrer à quel point nous sommes capables de contextualiser les choses, d'ajouter une quantité formidable d'informations. La réalité augmentée est à la mode, mais la quantité d'informations que nous sommes capables d'ajouter grâce à cette technique est dérisoire par rapport à la quantité disponible sans équipements. C'est plutôt les robots qui rêveraient de disposer de nos *super pouvoirs*.

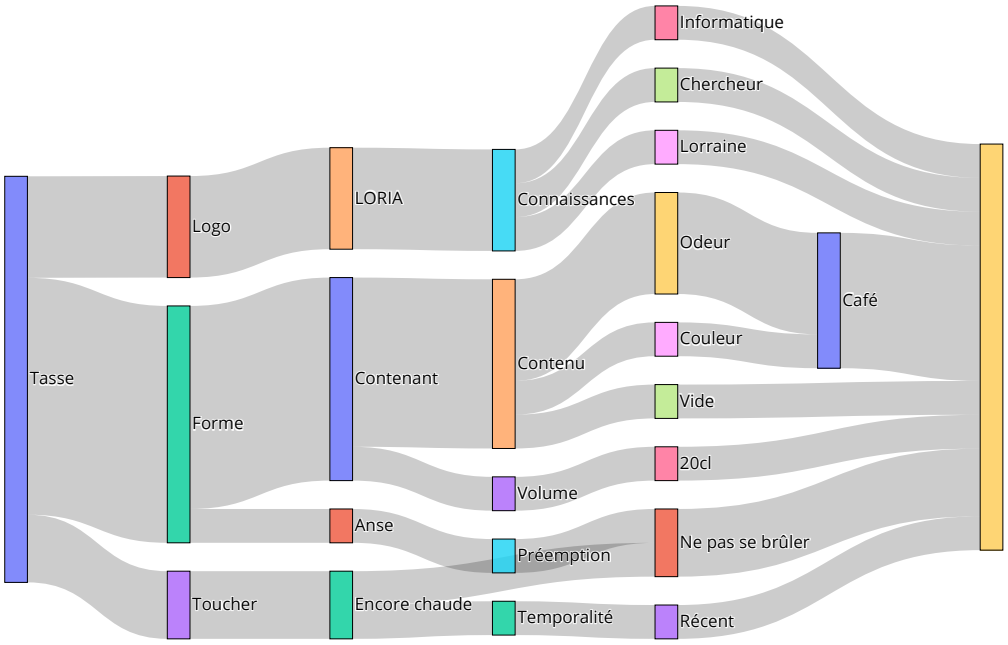

FIGURE 1.6 – Un chercheur en informatique lorrain à récemment bu 20cl de café sans se brûler

Nous l'avons vu, la perception est donc une tâche complexe, que nous accomplissons avec brio. L'ensemble du contexte que nous sommes capables d'ajouter à nos observations nous permet, même inconsciemment de mieux identifier la situation dans laquelle nous nous trouvons. Ainsi, des informations de haut niveau sont en permanence générées par notre cerveau comme la notion de risque, d'utilité, d'intérêt de telle où telle subdivision de notre environnement. Malheureusement, nous ne maîtrisons pas l'ensemble des mécanismes qui permettent une telle efficacité. Certaines théories, comme celle de la *loi universelle de la généralisation* [Shepard, 1987], expliquée plus tard par des approches probabilistes [Griffiths et al., 2010], et étendue par des travaux [Sims, 2018] qui définit comme limite à la perception la capacité d'un être vivant —ou d'un système— à récolter et traiter les stimulations issues de son environnement.

1.5.1 Stimulations, Données ou informations ?

Prenons quelques instants pour poser notre récit afin de préciser un point fondamental pour la compréhension de la suite de l'aventure. Nous avons précédemment utilisé différents termes —données, informations, connaissances— sans en déterminer les contours. Or il est important de bien distinguer le sens de ces mots qui peuvent d'un premier abord faire référence à un concept identique. Nous adopterons par la suite la taxonomie proposée par Russell Ackoff [Russell Ackoff, 1989].

> [...] Furthermore, the distinction between data, information, and so on up to wisdom are seldom made in the educational process, leaving students unaware of their ignorance. They not only don't know, they don't know what they don't know.
> — Russell Ackoff

> [...] En outre, la distinction entre les données, les informations, et ainsi de suite jusqu'à la sagesse, est rarement faite dans le processus éducatif, laissant les étudiants inconscients de leur ignorance. Non seulement ils ne savent pas, mais ils ne savent pas ce qu'ils ne savent pas.
> — Russell Ackoff - *traduit de l'anglais*

Données Un son arrive à vos oreilles.

Informations Il semblerait que votre voisin joue du violon.

Connaissances Vos ignoriez l'existence de telles notes.

Discernement/Compréhension Votre voisin débute le violon.

Les données sont des stimulations qui représentent des propriétés d'objets ou d'événements. Les informations modélisent également des propriétés mais de manière plus concise et utile que les données dont elles sont issues. La différence entre données et informations est fonctionnelle, pas structurelle. En effet, un ensemble de données structurées ne contient pas nécessairement plus de sens que ce même ensemble sans structure. L'opération de traitement permettant de convertir les données en informations ajoute du sens.

La connaissance est une construction faite d'une multitude d'informations, elle peut être vue comme la synthèse des expériences accumulées au fil du temps —et de l'espace. En informatique, elle est souvent implémentée sous forme d'instructions, de règles à suivre. Ces dernières peuvent êtres mises à jour afin de prendre en compte l'évolution de l'environnement, produisant ainsi des systèmes plus robustes.

Enfin la compréhension est la capacité d'expliquer le lien de causalité à partir des effets mesurés. C'est évidemment l'objectif ultime de la perception, qui n'est toutefois pas évident. Il existe de nombreux faits constatés que l'on ne saura probablement jamais expliquer. On peut citer l'*efficacité déraisonnable des mathématiques dans les sciences naturelles* [Courant and Wigner, 1960] soulignée par E. P. Wigner, que l'on ne parviendra probablement jamais à comprendre :

> The miracle of the appropriateness of the language of mathematics for the formulation of the laws of physics is a wonderful gift which we neither understand nor deserve. We should be grateful for it and hope that it will remain valid in future research and that it will extend, for better or for worse, to our pleasure even though

> perhaps also to our bafflement, to wide branches of learning.
> — E. P. WIGNER
>
> Le miracle de la pertinence du langage mathématique pour la formulation des lois de la physique est un cadeau merveilleux que nous ne comprenons ni ne méritons. Nous devrions en être reconnaissants et espérer qu'il restera valable dans les recherches futures et qu'il s'étendra, pour le meilleur ou pour le pire, à de vastes branches du savoir, pour notre plus grand plaisir, mais peut-être aussi pour notre plus grand désarroi.
> — E. P. WIGNER - *traduit de l'anglais*

Une meilleure compréhension d'un phénomène permet d'interagir avec de manière plus efficace. D'ailleurs, nous n'avons toujours pas défini l'efficacité. Nous l'entendons au sens du rendement, c'est-à-dire de la capacité à mieux répondre à une problématique avec des ressources données, ou encore minimiser la quantité de ressources nécessaires pour trouver, en reprenant les termes de Leslie Valiant [Valiant, 2013], une solution *probablement approximativement correcte*, répondant à nos objectifs initiaux.

1.6 Synthèse

Nous avons abordé dans ce chapitre la nécessité de développer de nouvelles solutions autonomes d'exploration d'anciens sites miniers, sites qui comportent des caractéristiques spécifiques. Après avoir défini l'autonomie, nous avons identifié comme verrou principal à sa réalisation la capacité de perception, c'est-à-dire de collecter des données issues de l'environnement et de les traiter pour produire des informations pertinentes pour notre application.

Cette notion de pertinence est évidemment dépendante de la mission considérée, mais il existe des informations fondamentales, nécessaires au fonctionnement de la majorité des systèmes, à plus forte raison lorsque ces derniers sont mobiles. Il s'agit du modèle géométrique de l'environnement —une carte par exemple— ainsi que de la position du système cyber-physique dans cette carte. En effet, ces deux informations sont aussi complémentaires qu'indispensables à la réalisation de n'importe quelle mission autonome comprenant un agent mobile. Assurément, une position sans référence associée n'a pas de sens, et l'absence d'informations de position rend une carte plus difficile à exploiter : Imaginez l'utilité d'une carte au trésor sans l'iconique ✗…

Ayant identifié le problème de positionnement et de cartographie comme le verrou majeur de la robotique autonome, nous avons concentré nos travaux sur ce thème. Toutefois, nous avons également évoqué dans ce chapitre les spécificités de l'environnement liées au cas pratique de l'après-mine que nous avons identifié dans la section 1.3. Ces particularités ne permettent pas de suivre les paradigmes actuellement développés en robotique autonome. Nous avons donc centré notre travail de recherche sur la construction d'une approche nouvelle, permettant de générer —en réduisant au maximum la dépendance à l'environnement— les précieuses informations de cartographie et de positionnent strictement nécessaires à l'exécution de toute mission autonome par un système cyber-physique évoluant dans un environnement inconnu.

1.7 Plan de la thèse

Nous allons d'abord commencer par situer nos travaux parmi les nombreux développements récents de la robotique autonome (2). Ce chapitre sera l'occasion de faire un tour d'horizon des solutions existantes, et d'identifier l'originalité de notre approche.

Nous décrirons ensuite nos contributions à travers quatre chapitres. Les deux premiers, traitant respectivement du système de positionnement (3) que nous avons conçu et de son application à la cartographie (4) formeront une deuxième partie. Les deux chapitre suivants, seront dédiés à la présentation des algorithmes permettant le déploiement de notre système dans un environnement réel. Ils présenterons respectivement la génération de trajectoire adaptée à la cartographie (5) et le calcul du placement optimal des phares (6) et composeront notre troisième et dernière partie.

Bien que synthèses et analyses seront proposées en fin de chapitres, nous recomposerons ces dernières dans une conclusion plus générale, qui tentera de prendre du recul par rapport au travail proposé et à l'expérience vécue lors de ces années de thèse.

Chapitre 2

État de l'art

Nous avons vu précédemment que l'obtention des informations de positionnement et de cartographie étaient aujourd'hui un défi majeur de la robotique autonome. Nous nommerons par la suite **SLAM** (pour Simultaneaous Localisation And Mapping ou positionnement et cartographie simultanée) les systèmes permettant d'estimer ces informations.

Avant d'aborder la pléthore d'approches différentes de SLAM, nous allons présenter les divers types d'informations —de modèles— qui sont classiquement générées par ces approches.

Nous pourrons par la suite aborder le coeur de ce chapitre, la présentation et l'analyse des approches explorées par la communauté scientifique pour tenter de répondre à cette problématique complexe.

2.1 Définition du problème

2.1.1 Caractérisation du positionnement

Nous évoquons depuis quelques pages déjà la notion de positionnement, sans pour autant en avoir précisé les contours. Remédions donc rapidement à cela. Une position est un lieu où une chose est placée, sa situation. Dans la vie courante, cela peut être représenté par une zone —une ville, une localité—, une adresse ou encore des coordonnées comme le couple $(Latitude°Nord, \quad Longitude°Ouest)$. Si c'est bien à des coordonnées que nous nous intéressons, il s'agit pour nous de trois coordonnées cartésiennes permettant de représenter un point p dans l'espace euclidien tri-dimensionnel tel que $p = \begin{bmatrix} x & y & z \end{bmatrix}$. Ces deux systèmes de coordonnées ont un point commun, ils sont exprimés par rapport à une référence —l'équateur et le méridien de Greenwich pour Lat, Lon. Dans notre cas cependant, la position p sera exprimée en fonction du point de départ de la mission, qui sera l'origine de notre système de coordonnées cartésiennes.

Bien qu'étant une position relative à la position initiale —l'origine— d'une mission, nous noterons cette position **Globale**, par contraste avec les positions **Locales**, qui elles sont exprimées par rapport à une autre référence que l'origine. Un exemple de position locale est la position relative de deux robots, qui ne renseigne pas directement sur la position de ces deux robots dans le monde.

Nous avons défini la position, toutefois, le terme positionnement englobe une autre composante que la position. Il s'agit de l'orientation, qui désigne dans le langage courant la direction de l'orient —d'où se lève le soleil à l'équinoxe– et plus largement les points cardinaux (Nord, Sud, Est, Ouest). Cette notion se transpose dans notre système cartésien par la combinaison de rotations autour des trois axes X, Y et Z. Cette combinaison peut être exprimée par une **matrice de rotation 3x3** nommée R :

$$R = R_z(\alpha)\, R_y(\beta)\, R_x(\gamma) = \underbrace{\begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\text{lacet (yaw)}} \underbrace{\begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix}}_{\text{tanguage (pitch)}} \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\gamma & -\sin\gamma \\ 0 & \sin\gamma & \cos\gamma \end{bmatrix}}_{\text{roulis p(roll)}}$$

Où les angles α, β, γ représentent respectivement le **lacet**, le **tangage** et le **roulis**, termes utilisés aussi bien dans l'aviation que dans la robotique terrestre la figure 2.1 permet de visualiser ces axes de rotations .

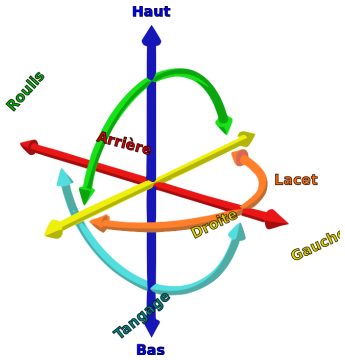

FIGURE 2.1 – Convention sur les axes de rotation, par Tubezlob

Nous avonc donc déterminé comment se concrétise la notion de positionnement, à travers la position et l'orientation. Ces deux informations permettent de représenter le positionnement d'un modèle indéformable dans notre monde physique, en couvrant tous les degrés de libertés de ce dernier. On parle alors de positionnement **6Dof** (pour Degrees Of Freedom).

Il est possible de regrouper ces deux notions à travers un seul outil mathématique, une matrice de transformation (application affine) homogène. Il est possible de définir simplement une telle matrice T à partir du vecteur de position p et de la matrice de rotation R de la manière suivante :

$$T = \begin{bmatrix} R & p \\ 0 & 1 \end{bmatrix}$$

En plus de combiner ces informations, cette matrice permet d'appliquer aisément des opérations de composition et d'inversion —entre autres— dont nous exploiterons le potentiel dans le chapitre suivant.

Le problème de positionnement est donc de fournir, au fur et à mesure de la mission, une estimation de l'état des différents éléments suivis, c'est à dire leurs positions et orientations respectives, et ce de la manière la plus précise possible. Une autre métrique de la performance

d'un système de positionnement, souvent mise de côté dans la littérature, est la fréquence à laquelle le système est capable de produire cette estimation.

Cette problématique de positionnement peut paraître dépassée, alors que la majorité d'entre nous est équipée d'appareils **GPS** (*Global Positioning System* ou système de positionnement global). Cette accessibilité masque le fait qu'il a fallu mettre en orbite 31 satellites (30 pour le système Galileo) et que sa précision reste limitée $\approx 5m$, bien que la version haute précision de Galileo $\approx 20cm$ —voire $\approx 10cm$ avec utilisation d'une autre base terrestre— devrait voir le jour en 2024[2]. Il est important de préciser qu'il existe une grande variété d'environnements dans lesquels un tel système de positionnement global n'est pas disponible. En effet, les ingénieurs en aérospatial ne disposent pas de cette commodité pour leurs rovers. Sans aller si loin, il existe de nombreux lieux où ce signal n'est pas disponible, comme l'intérieur de grands bâtiments, ou encore des souterrains, comme les mines auxquelles nous nous intéressons.

2.1.2 Vers quels modèles ?

Nous l'avions évoqué dans l'introduction, la cartographie consiste à construire, à partir de mesures de l'environnement, un modèle de ce dernier. Nous sommes familiers avec le concept de carte, qui n'est pourtant que l'une des nombreuses options qui permettent de modéliser l'environnement. Nous allons premièrement aborder les catégories principales de modèles utilisés dans la littérature :

Grille d'occupation (occupancy grid) Une grille d'occupation [Thrun, 2003] est un tableau de cellules stockant la probabilité de trouver un obstacle à un point donné de l'espace. Cette grille peut également être en trois dimensions [Hornung et al., 2013], on parle alors de grille de voxels, mais le principe reste exactement le même. La figure 2.2 présente l'image générée à partir d'une grille d'occupation 3D. Ce modèle a l'avantage d'être **dense**, il peut donc être utilisé pour de nombreux points clés de la navigation robotique que sont la planification de parcours, l'évitement d'obstacles ou encore l'interprétation directe par un humain. Ce dernier point permet en effet de diriger le robot au cas où son auto-détermination ne soit pas suffisante, mais également l'analyse du terrain ainsi reconstitué, élément nécessaire à de nombreuses applications de la cartographie.

Ensemble d'amers Ce terme issu de la marine —amers— désigne les points de repères —comme les tours et clochers le long de la côte— permettant de naviguer à vue. Ce modèle de l'environnement est un ensemble d'amers associés à leurs positions supposées dans l'espace. Ces amers sont sélectionnés pour leur caractère unique, reconnaissable —ils doivent pouvoir êtres identifiés parmi les autres amers. Ainsi cela dépend largement de l'environnement, nous parlions de clocher précédemment, mais cela n'est valide que s'il est reconnaissable, ainsi, une tour dans un village composé uniquement de petites maisons sera un bon amer, alors que cette même tour ne constituera pas une bonne solution si elle fait partie d'un système défensif comportant des dizaines d'autres tours identiques. Le choix de l'amer se portera alors sur l'unique maisonnette devant ces remparts. Comme le montre la figure 2.3, cette représentation, bien que suffisante pour la navigation, n'est qu'une représentation partielle de l'environnement, ou encore **éparse**. La figure reprend les visuels dans l'ensemble d'amers alors qu'en réalité ils ne sont représentés que par

2. Site officiel du projet Galileo https://www.gsc-europa.eu

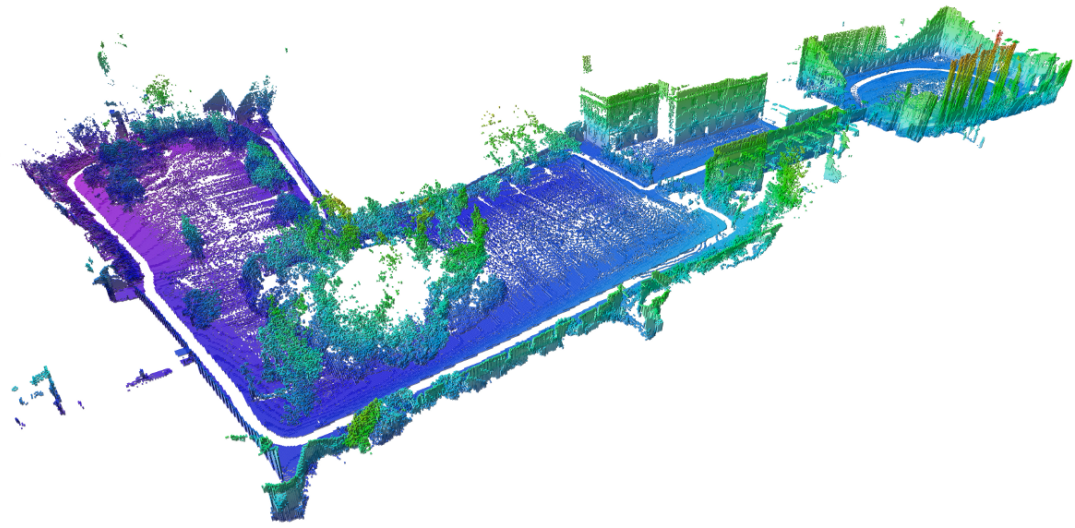

FIGURE 2.2 – Visualisation d'une grille d'occupation 3D, réalisée avec Octomap.

des points, avec lesquels sont stockés le modèle de l'amer qui dépend de l'algorithme et du capteur utilisé. Cette représentation ne permet donc ni l'évitement d'obstacles, ni l'interprétation directe du modèle par un humain. Si vous cherchiez une maison dans ce paisible hameau, ce modèle serait bien inutile, il vous faudrait ajouter d'autres données pour en tirer les informations pertinentes au sens de votre recherche.

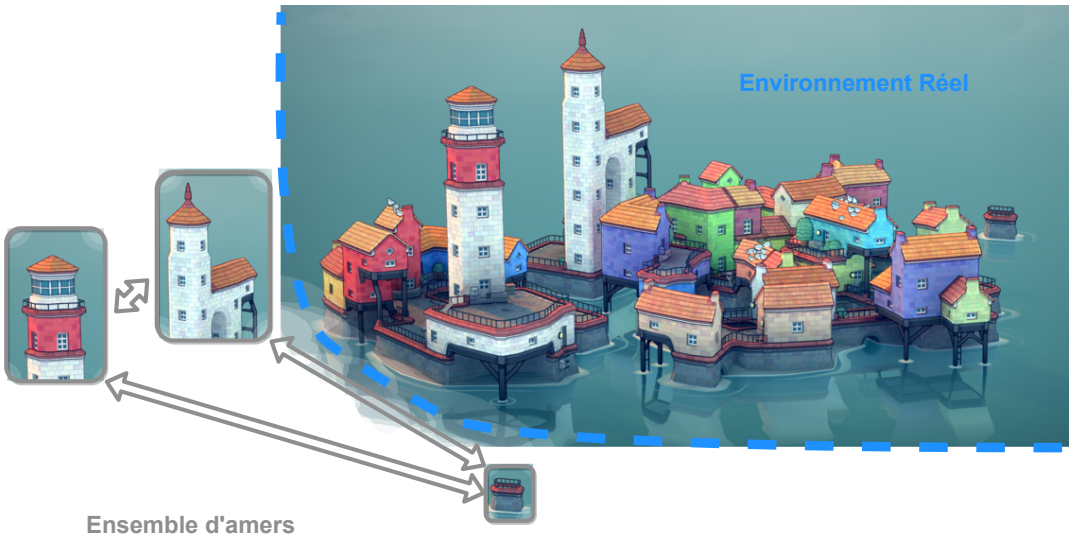

FIGURE 2.3 – Amers humains, illustration générée avec le jeu Townscaper, de Oskar Stålberg

Graphe de facteurs (factor-Graph) Un graphe de facteurs [Loeliger, 2004, Dellaert and Kaess, 2006], est un graphe qui permet de modéliser l'incertitude que l'on a sur l'ensemble des états successifs du robot. Dans le cadre du problème de positionnement, ces états successifs sont en réalité la trajectoire effectuée par le robot. Il est également pos-

sible d'inclure de manière transparente les observations faites lors de ce parcours, formant ainsi une *carte* de l'environnement. Ce modèle permet d'inférer en temps réel les positions du robot ou des objets observés (voir section 2.4.2). La figure 2.4 présente un tel graphe, où les ronds bleus sont les positions x_i successives du robot et les carrés noirs les observations l_i —ici des amers. Li s'agit bien des mêmes amers, en effet, ce graphe est un modèle permettant d'intégrer un ensemble d'amers en y ajoutant des probabilités, permettant de réduire les erreurs de mesure comme nous le verrons par la suite. Toutefois, les informations contenues dans ce graphe sont des modèles d'observations éparses, qui en soit ne constituent pas un modèle exploitable par un humain. Une étape supplémentaire sera donc nécessaire à la création d'un tel modèle par ajout d'images ou de nuages de points par exemple.

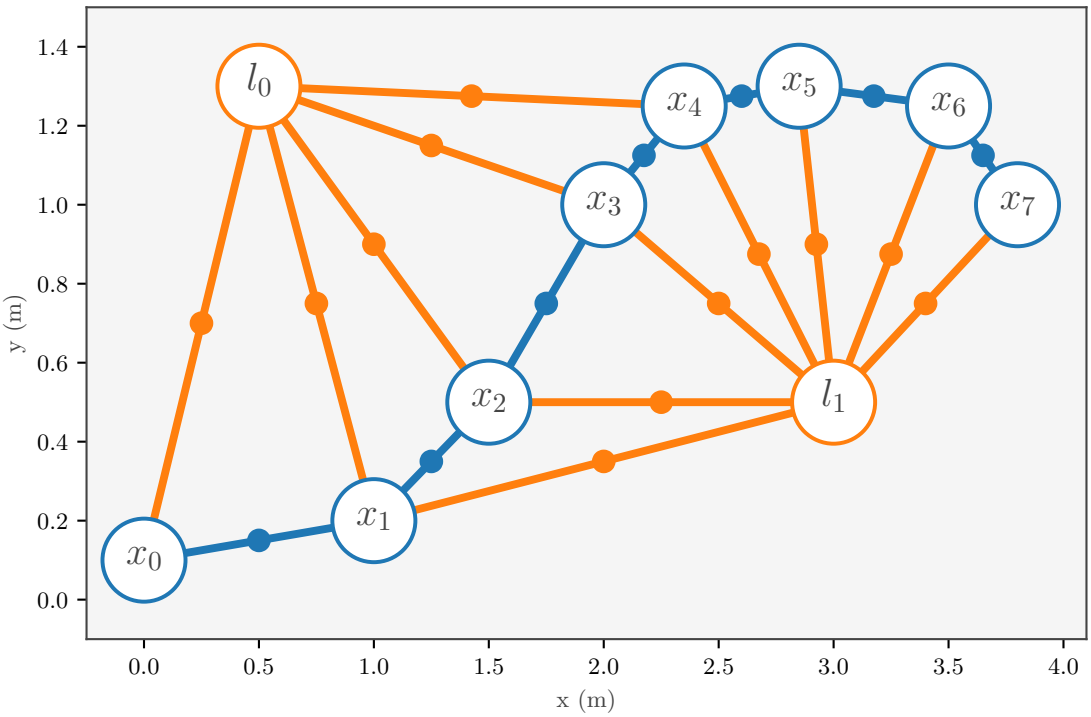

FIGURE 2.4 – Graphe de facteurs, stockant les informations spatiales et leur probabilités associées

Nous l'avons vu, les modèles de l'environnement classiquement créés par les algorithmes de SLAM ne sont pas nécessairement directement utilisables, que cela soit par le robot pour sa navigation autonome, ou par un humain ou un algorithme pour l'exploitation des informations acquises lors de la mission. Nous y ferons désormais référence à travers le terme **modèles internes**. On peut alors se demander quels sont les modèles attendus d'un système de cartographie. Il existe presque autant de modèles que d'applications, mais nous allons citer les principaux. Le premier et certainement le plus utilisé est l'image. Il s'agit à la fois d'un modèle relativement simple à construire, et qui porte de nombreuses informations, facilement exploitables par un humain, presque comme s'il était sur place. Ces images peuvent être des compositions de plusieurs photographies, corrigées —et enrichies— avec les informations acquises par divers capteurs.

Un autre modèle moins fréquent, mais largement adopté dans le bâtiment est le nuage de

points. Composé uniquement de points 3D, auxquels l'on peut associer des informations supplémentaires —une couleur par exemple—. Ce modèle est dense, il est donc directement exploitable par un humain, comme le montre la figure 2.5. Il est également facile de se projeter dans cet environnement, même si le visuel s'éloigne de celui que l'on aurait sur place.

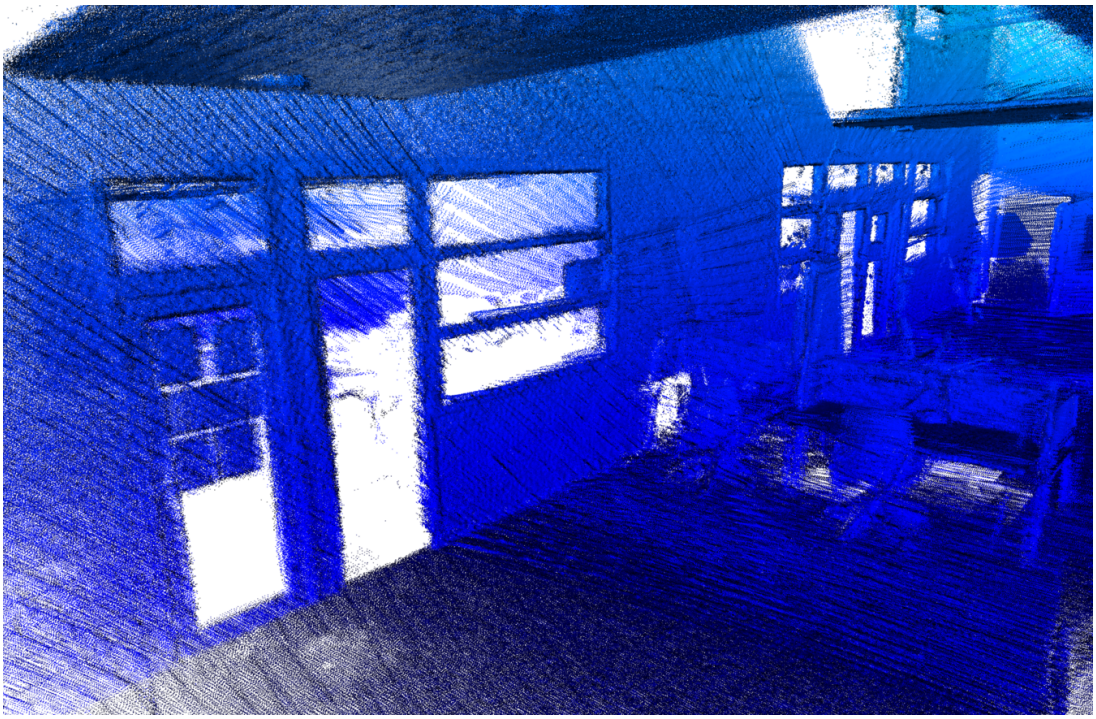

FIGURE 2.5 – Vue 2D d'un nuage de points généré par nos algorithmes

Enfin, un modèle combinant les avantages des deux derniers est un maillage 3D, composé de triangles formant des surfaces, sur lesquelles il est possible d'ajouter des textures —potentiellement issues de photographies de l'environnement réel. Ce modèle à été très largement adopté dans le monde du jeu vidéo, car il permet de renforcer le sentiment d'immersion du joueur. L'image *environnement réel* de la figure 2.3 est une vue d'un maillage 3D, dans lequel il est possible d'évoluer, permettant une analyse du terrain, une application possible étant la préparation d'une intervention de pompiers par exemple. Ce type de modèle est globalement plus difficile à obtenir, et demande le chaînage de nombreux algorithmes, appartenant parfois à des domaines de recherche différents.

Les modèles de l'environnement déterminés et utilisés par les systèmes classiques de SLAM ne sont pas nécessairement des modèles exploitables pour d'autres tâches. La construction de tels modèles *humainement compréhensibles* demande dans une grande majorité des cas des étapes supplémentaires.

2.2 Quels outils ?

Après avoir précisé le type d'informations attendues pour répondre à notre problématique de positionnement et de cartographie, nous allons nous intéresser aux différents outils disponibles dans la littérature. Le domaine de recherche est assez vaste, et comporte énormément d'approches qui sont souvent des monstres de Frankenstein, composées de nombreux algorithmes interconnectés d'une manière spécifique pour répondre à une problématique particulière. Je dois ici faire l'aveu de la difficulté rencontrée en début de thèse, où il n'était pas évident de faire la part des choses. En effet, la compréhension des tenants et aboutissants du domaine s'est révélé être une épreuve de longue haleine, ponctuée par l'incapacité occasionnelle d'identifier l'intérêt et les objectifs spécifiques de certaines contributions publiées, tant le paysage est varié. De plus, l'effort de vulgarisation ne semble pas une priorité dans la rédaction scientifique, ayant pour conséquence de rendre difficile d'accès des documents qui ont pourtant pour vocation le partage de la connaissance. C'est cette raison qui motive, dans l'intégralité de ce document, la volonté et l'application mises en oeuvre pour rendre les différents concepts présentés le plus intelligible possible.

Nous allons donc dans un premier temps nous intéresser à l'architecture d'une solution de SLAM, afin d'identifier des catégories —ou des niveaux— d'algorithmes. Cela permettra de donner une vue globale d'un processus classique.

Après avoir identifié ces niveaux, nous étudierons chacun d'entre eux ainsi que les briques qui les composent. Cette stratégie devrait permettre de donner une vision d'ensemble du paysage algorithmique du SLAM.

2.2.1 Multi-niveaux

Les divers algorithmes constitutifs d'un système de SLAM réalisent des tâches souvent bien différentes. Il est toutefois possible de les catégoriser d'après leur positions dans le flux de traitement des données. La figure 2.6 présente un flux classique d'informations. Les entrées du système sont les données produites par le(s) capteur(s) et son (leurs) pilote(s). Bien que nous ayons représenté deux capteurs ici, certaines approches en utilisent plus [Shao et al., 2019], au prix toutefois d'une augmentation de la complexité du système et des coûts en calculs, pour obtenir des résultats similaires à l'état de l'art.

Estimateurs (*front-end*) La première catégorie est celle des estimateurs de déplacement ou de position. Souvent qualifiés de *front-end* dans la littérature, ces algorithmes produisent, à partir des données issues d'un capteur, une estimation des déplacements effectués où encore de la position actuelle du robot. Ils sont intimement liés à la nature des données qu'ils traitent, et leur complexité, leur précision et leur robustesse varient donc fortement selon les approches.

Intégrateurs (*back-end*) La seconde catégorie englobe les algorithmes qui exploitent les résultats d'un ou de plusieurs estimateurs, afin d'optimiser la précision de manière globale, en prenant en compte l'ensemble des mesures faites par le système. La littérature y comporte des références sous le terme *back-end*. Ils s'appuient sur les probabilités pour modéliser l'état du système considéré ainsi que l'incertitude associée. Bien que leur fonctionnement soit indépendant de celui des estimateurs, la précision et la robustesse des

résultats obtenus par les *back-end* sont tributaires de la qualité des données à leur disposition.

Adaptation La troisième catégorie est celle du **post-traitement**, de l'utilisation de l'information de positionnement et des données des capteurs dans le but de produire un modèle **applicatif**, adapté à une mission donnée. Cela peut être la production par exemple d'un nuage de points colorés, permettant l'exploration virtuelle d'un monument historique, objectif final d'une mission de cartographie pour la préservation du patrimoine par exemple. Les solutions remplissant ces traitement d'adaptation s'éloignent du domaine de la robotique autonome, c'est pourquoi nous ne les aborderons pas dans ce chapitre.

Détection de boucle (*loop-closure*) Le but est simple, reconnaître et identifier un lieu déjà visité lorsque on y retourne. Ces algorithmes de détection sont intimement liés à la fois à l'intégrateur, et au capteur qui fournit les données initiales. Développer un tel outil demande donc d'avoir un système complet de SLAM en place, et il sera difficile de l'utiliser pour un autre système. Nous avons pour cette raison rapidement écarté l'éventualité de nous diriger vers ce domaine spécifique. Nous n'aborderons par conséquent pas en détails ce vaste sujet. Globalement, une détection de boucle a pour objectif d'associer deux observations d'une même réalité physique, mais faites à des temporalités différentes. Ce lien permet de mettre à jour le modèle de l'intégrateur et ainsi réduire la dérive de positionnement. Ces approches sont généralement très coûteuses en calculs —et en mémoire— car il s'agit de comparer chaque nouvelle observation avec l'ensemble des précédentes. De plus, une association erronée peut avoir un impact dramatique sur la précision de l'estimation de position.

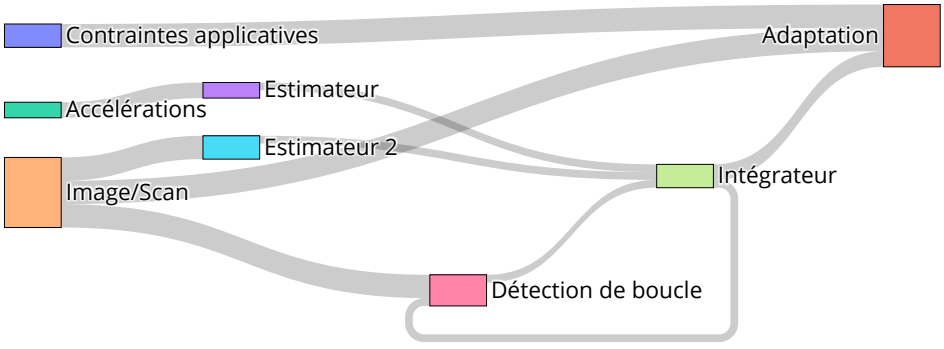

FIGURE 2.6 – Flux de traitement des données dans un système SLAM classique

Ces catégories sont poreuses, certains algorithmes étant à cheval sur ces dernières, mais elles décrivent bien les grandes étapes du processus de SLAM, et permettent de mieux identifier la portée des différentes contributions que nous allons présenter.

Une autre façon de catégoriser les solutions de SLAM se base sur le moment où ils sont

employés. Certains algorithmes ne sont en effet pas capables de fournir et de maintenir le positionnement et la carte en temps réel. On parle alors d'algorithmes *offline* ou encore de *full SLAM*, car ils utilisent parfois l'intégralité des mesures faites durant l'expérience. Ces approches ne permettent évidement pas de rendre le robot autonome, car les données ne sont disponibles qu'après de lourds calculs, réalisés après coup. De plus, ces approches ont tendance à disparaître avec l'augmentation de la performance des algorithmes et de la puissance des machines embarquées. Pour ces raisons, nous ne nous intéresserons pas à ces approches.

2.3 Estimateurs (*front-end*)

Première étape algorithmique de la chaîne de traitement des données issues de l'environnement, les estimateurs utilisent en entrée les données produites par les capteurs et leurs pilotes associés. Ils sont par conséquent intimement liés à ces derniers. Il existe de très nombreux capteurs et estimateurs associés, que nous ne pourrons pas tous traiter, nous nous concentrerons sur trois grandes catégories, les Centrales inertielles (*IMU en anglais*), les caméras, et enfin, les Lidars —acronyme de l'anglais *laser imaging detection and ranging*. La majorité des développements récents ont été réalisés à partir de ces appareils, et se concentrent particulièrement sur les caméras.

Bien que nos contributions n'appartiennent pas à ces domaines respectifs, il est important d'aborder les grand principes de leur mise en oeuvre. Cela permettra de bien identifier les problématiques inhérentes à ces systèmes, et aux défis qu'ils doivent encore relever.

2.3.1 Odométrie Inertielle

Les centrales inertielles sont utilisées dans une grande diversité d'appareils, du bateau à la fusée. Ces *centrales* sont en réalité une collection de plusieurs capteurs, généralement trois gyromètres et trois accéléromètres. Les gyromètres mesurent les vitesses angulaires et les accéléromètres les accélérations linéaires sur les trois axes. L'intégration des données permet de déterminer la position et l'orientation du système, relativement à une position et une orientation initiales. Cette construction récursive —basée sur l'estimation précédente— de l'estimation conduit à l'accumulation d'erreur, aussi appelée **dérive**. Il existe de nombreux modèles de centrales inertielles, mais même les meilleurs sont soumis à cette dérive, intrinsèque à ces systèmes. L'estimation fournie devient donc rapidement imprécise, et les centrales inertielles sont utilisées principalement en support d'autres méthodes.

2.3.2 Odométrie Visuelle

Inutile de présenter le fonctionnement d'une caméra tant elles ont pénétré notre quotidien. Nous n'aborderons ici que les caméras monoculaires, premièrement parce que les algorithmes sont *relativement* similaires pour les autres types de caméra mais également car les développements récents semblent se concentrer sur ces dernières en raison de leur commodité d'emploi. En effet, les caméras monoculaires ne sont pas chères, ne consomment que peu d'énergie, et sont légères. Elles sont donc des candidates naturelles pour la robotique, et tout particulièrement pour les drones volants.

Le principe de base de l'odométrie visuelle repose sur la génération d'une estimation de transformation entre les points de vue de deux images différentes, prises soit par des appareils différents, soit à des temporalités différentes. Pour cela, trois grandes étapes sont généralement employées :

— Détection d'amers
— Appariement d'amers
— Déduction de la transformation

1. *Détection d'amers :* Le point de départ de cet algorithme et d'être capable de reconnaître des amers. Nous l'avons vu, un amer est un point reconnaissable de l'environnement. Cette étape consiste à déterminer quels points dans l'image sont reconnaissables, et à en produire une description, de manière à être capable de l'identifier à nouveau dans une autre image capturant la même scène —depuis un autre point de vue évidemment. Pour reprendre l'analogie maritime, nous pourrions décrire le clocher comme *un clocher dominant le village, surplombant une tour rouge et coiffé d'un toit en ardoise noire.* Les descripteurs produits par —et pour— des algorithmes sont en réalité des vecteurs multi-dimensionnels représentant au mieux les caractéristiques de l'amer détecté.

   Les premières recherches d'amers se sont penchées sur la détection de *coins (corners en anglais et dans la littérature)* [Harris and Stephens, 1988, Shi and Tomasi, 1994]. Ils présentent l'avantage de rester des coins lorsque le point de vue change. Cependant, ces amers sont sensibles aux changements d'échelle, phénomène systématique si l'on s'approche où l'on s'éloigne d'une scène. C'est pour corriger cela que la méthode *SIFT (Scale-Invariant Feature Transform)* [Lowe, 2004], basée sur la construction d'un vecteur —d'un descripteur— à haute dimension représentant les gradients locaux de l'image a été développée. Cette dimensionnalité a un coût, c'est la raison pour laquelle d'autres travaux ont porté sur la réduction des calculs comme *SURF (Speeded Up Robust Features)* [Bay et al., 2006], où encore une approche basée sur l'apprentissage machine *FAST (Features from Accelerated Segment Test)* [Rosten and Drummond, 2006]. La réduction de la mémoire nécessaire a également fait l'objet de travaux aboutissant à la publication *BRIEF (Binary robust independent elementary features)* [Calonder et al., 2010]. Enfin, l'approche proposée par *ORB* —disponible dans la librairie *OpenCV*[3]— [Rublee et al., 2011] est, comme l'approche *BRISK* [Leutenegger et al., 2011] une composition des solutions *FAST* et *BRIEF*, formant des solutions permettant la détection et la description d'amers en temps réel et cela dans une enveloppe mémoire réduite.

   La figure 2.7 présente la localité dans l'espace 2D de chaque image de chacun des amers détectés sur notre environnement jouet. Nous avons employé l'implémentation de ORB mise à disposition par la bibliothèque OpenCV via son API python. La figure présente les amers dans les deux images, mais pas leurs descripteurs, qui ne sont de toute manière pas interprétables visuellement par un humain.

2. *Trouver les correspondances :* Passons maintenant à la détection des correspondances —l'appariement— entre les amers des deux images. L'idée est de déterminer si deux observations, sont en réalité des modèles d'un même objet physique. Il s'agit ici d'évaluer la similitude —la distance entre les deux vecteurs descripteurs— des amers. Si l'on juge

3. https://opencv.org/

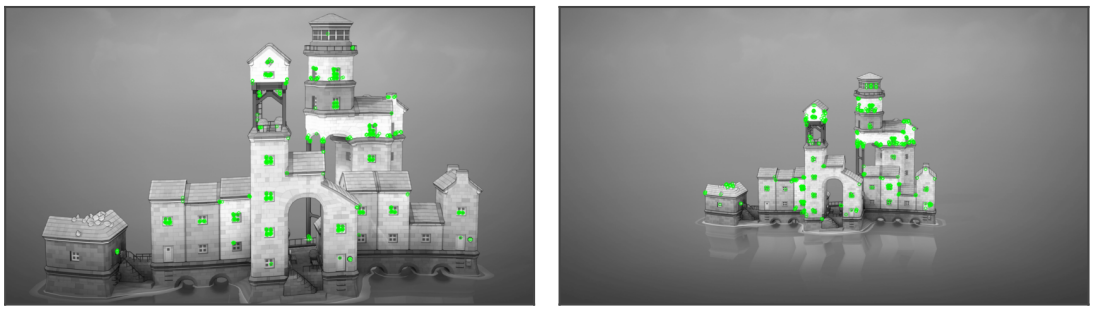

FIGURE 2.7 – Détection d'amers avec ORB sur deux images du même environnement

qu'ils sont suffisamment ressemblants, ils sont appairés. Ici, *suffisamment* signifie que leur distance est inférieure à un seuil arbitrairement défini. On touche ici à un écueil important des systèmes classiques de SLAM. En effet, il est possible de créer des associations entre deux observations, alors qu'elles ne représentent en réalité pas le même objet dans le monde réel. Cela mène inévitablement à l'ajout d'erreur de manière irrémédiable. En effet, il ne sera plus possible de corriger directement cette erreur, il faudra compter sur la compensation apportée par d'autres observations. L'accumulation de ces erreurs peut parfois conduire à une défaillance critique du système de SLAM.

Nous savons quel mécanisme permet de décider si deux observations font référence à un même objet et doivent donc êtres associées. Il reste toutefois à déterminer quels couples d'amers tester. La version naïve consiste à comparer chacun des descripteurs d'une image avec tous ceux de la seconde image. Cet algorithme de force brute est évidemment très coûteux, on va en effet tester tous les couples possibles, même si certains n'ont aucune chance d'êtres associés —car trop distants dans l'espace de description—. En réalité, il suffirait de tester l'amer de la seconde image dont la distance —toujours dans l'espace de description— est la plus faible. On pense alors immédiatement à l'algorithme de recherche de plus proche voisin comme étant une solution idéale pour sélectionner le candidat à tester. Cependant, la grande dimensionnalité des vecteurs descripteurs —128 dimensions pour SIFT— rend inefficaces ces algorithmes, dont la complexité augmente de manière exponentielle avec le nombre de dimensions. En effet, plus il y a de dimensions, moins chacune d'entre elle est discriminante. Une approximation de la recherche de proches voisins [Beis and Lowe, 1997], permet de surmonter cette difficulté. Une autre approche [Nistér and Stewénius, 2006] propose des résultats intéressants, permettant d'accélérer le processus. Elle est inspirée par le concept du *bag of visuals words* initialement développé pour récupérer de manière efficace une image similaire à l'image d'entrée dans une base de données d'images.

La figure 2.8 présente les meilleures associations d'amers réalisées entre nos deux images : Les associations ont été classées par distance, seules les n plus proches sont affichées. Nous pouvons constater plusieurs associations erronées, sources d'erreur pour la suite du processus. Toutefois, il s'agit d'un exemple jouet, et les algorithmes utilisés ont été utilisés sans paramétrages spécifiques. De plus, l'écart entre les deux points de vue est supérieur à ceux rencontrés classiquement entre deux *frames* de caméra. Il est important de noter que la similitude étant étudiée dans l'espace de description, l'erreur dans l'espace physique peut être importante. La performance obtenue ici est donc loin des standards de

la littérature, mais il est important de souligner que ces standards sont obtenus à grand renforts de micro-réglages. Ces étapes d'adaptation manuelles sont un obstacle à l'emploi de telle techniques dans des environnements inconnus, dont nous ne connaissons pas les propriétés.

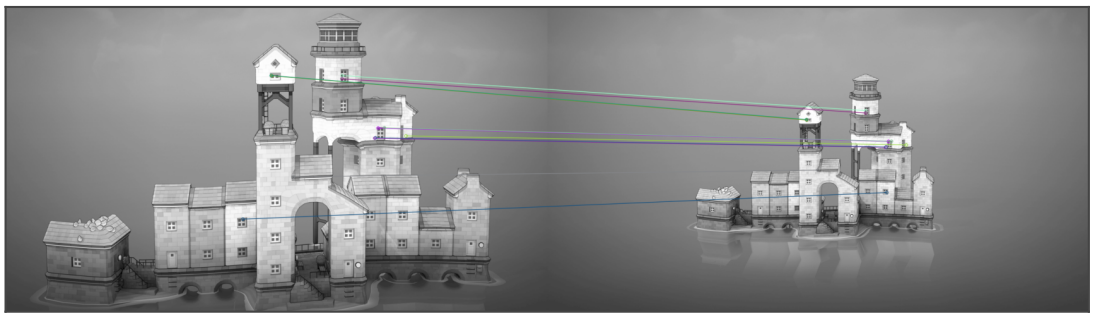

FIGURE 2.8 – Détection des correspondances entres les amers des deux images

3. *Déduction de la transformation :* Après avoir déterminer les correspondances, il faut déterminer la transformation entre les deux origines de caméra. Chaque correspondance est une transformation, déterminée par la différence entre les coordonnées dans la première et la seconde image. Cependant, comme nous l'avons vu précédent, il existe des erreurs d'association, qu'il faut éviter au maximum d'utiliser. Communément, la méthode RANSAC pour *RANdom SAmple Consensus* [Fischler and Bolles, 1981] est appliquée pour déterminer quelles correspondances utiliser pour calculer cette transformation. L'hypothèse est la suivante : il existe un ensemble de correspondances qui fait consensus, autrement dit qui sont approximativement d'accord sur la transformation entre les deux images. D'un autre coté, il existe des valeurs aberrantes —des *outliers*, probablement issus d'erreurs d'associations— qu'il ne faut pas prendre en compte pour calculer la transformation finale. Cet algorithme itératif repose sur deux étapes qui se répètent : Sélection aléatoire d'un candidat —ici une correspondance, donc une transformation— puis dénombrement des échantillons qui sont d'accord avec ce candidat —dont la transformation est proche dans notre cas—. On répète l'opération un certain nombre de fois, puis on sélectionne parmi tous ces essais celui dont le consensus était le plus fort.

   Bien que largement utilisé, RANSAC présente une sensibilité au seuil utilisé pour déterminer le consensus —qui permet de déterminer si deux correspondances sont d'accord—. En effet, si ce seuil est large, les *outliers* ne seront pas correctement filtrés, et à l'inverse, si le seuil est trop petit, le système devient instable car la transformation est alors calculée avec un petit nombre de correspondances. La littérature comprend de nombreuses extensions de RANSAC, qui tentent de résoudre ces problèmes, mais également de réduire le coût en calcul de l'approche.

   Par la suite, la détermination de la transformation entre les deux caméras est un problème classique de géométrie épipolaire connu sous le nom de *five-points Relative Pose problem*, pour laquelle une solution proposée par Nister [Nistér, 2004] est devenue une référence. la figure 2.9 présente les épilignes —droites qui passent par l'origine de la caméra et par les amers— et l'épipoint —l'intersection de ces lignes— qui correspond à la position

de la caméra lors de la première prise de vue. Elles sont affichées sur le second cliché, permettant ainsi d'apprécier la transformation obtenue entre les deux points de vue.

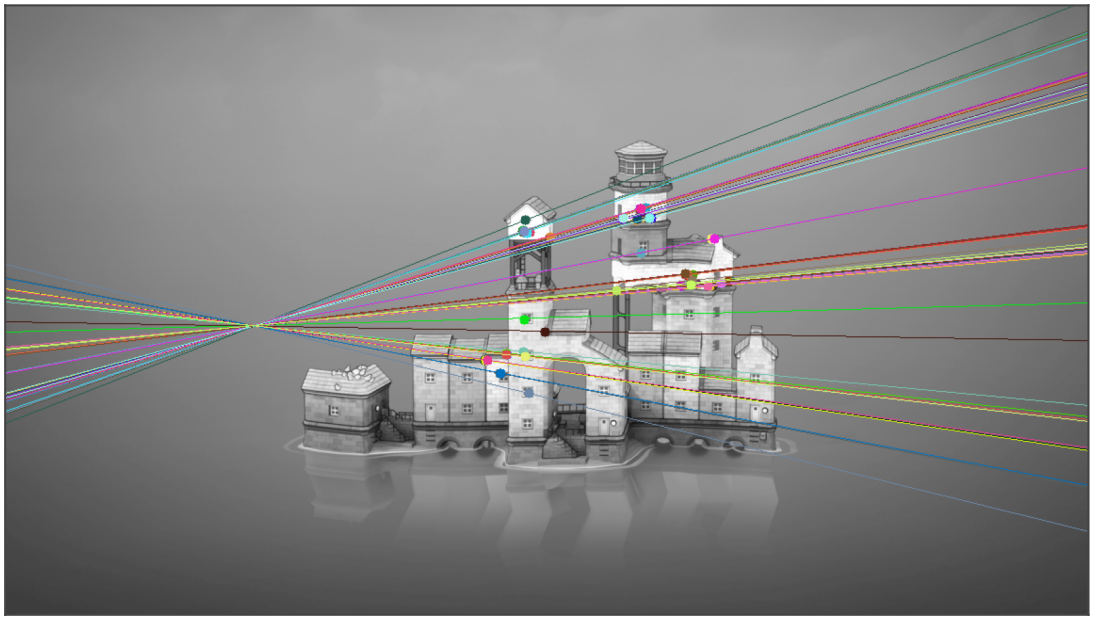

FIGURE 2.9 – Point de vue de la première image projeté sur la seconde calculée par RANSAC et Five-point

4. *Une estimation fiable de la position ?* La répétitions de ces étapes —qui fournissent un déplacement entre deux images— à chaque nouvelle image provenant de la caméra permet de construire une estimation de position, basée sur l'ajout à l'estimation actuelle du nouveau déplacement. On comprend alors qu'en utilisant ce principe, la moindre erreur sur un déplacement sera conservée dans l'estimation de position, et ce jusqu'à la fin de la mission. Ce phénomène est nommé ***dérive***, car à force d'accumulation de petites erreurs, l'estimation de positionnement du robot est de moins en moins précise.

   En plus de cette considération d'accumulation d'erreur, l'utilisation de caméras nécessite une hypothèse forte sur l'environnement : la présence de lumière. Cela ne correspond donc pas aux objectifs que nous nous sommes fixés, et l'emploi d'un estimateur visuel s'avère impraticable dans le contexte des mines abandonnées. Ces dernières années ont vu l'émergence d'études sur les caméras basées sur les événements. Chaque pixel opérant indépendamment, ces caméras promettent la capture d'informations à une fréquence élevée, qui plus est dans des environnements très peu lumineux. La disponibilité et le prix exorbitant des ces unités nous a découragé d'explorer cette piste.

2.3.3 Odométrie Lidar

Un lidar est un télémètre laser, mesurant généralement la distance d'un objet en envoyant dans sa direction une impulsion laser, et en mesurant le délai avec laquelle il capte la réflexion de cette impulsion. Plusieurs de ces cellules sont montées ensemble sur un berceau rotatif, permettant d'acquérir des scans 3D de l'environnement à partir d'un seul capteur. La figure 2.10

présente un unique scan obtenu à partir d'un lidar rotatif. Les points sont exprimés dans le repère du capteur, et non dans le repère global du monde. Ainsi, deux scans —nuages de points— étant collectés à des positions différentes en réalité seront exprimées dans le même repère, conduisant à l'impossibilité de les combiner directement pour former un nuage de points cohérent, fidèle à la réalité. Il faut donc dans un premier temps déterminer la transformation entre les deux prises de vue. Cette opération est appelée **recalage**.

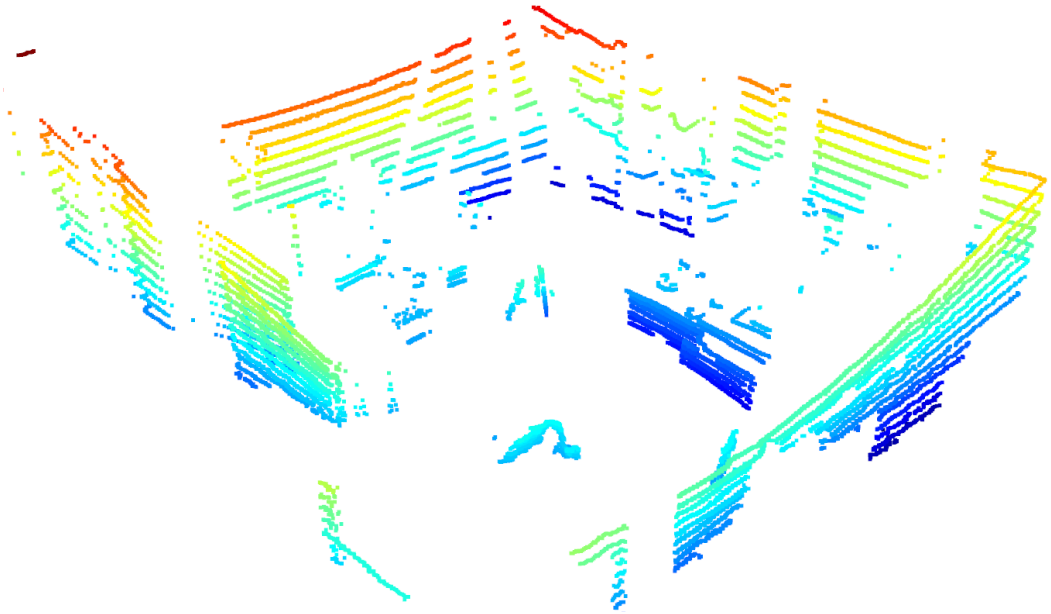

FIGURE 2.10 – Nuage de points issu d'un unique scan lidar de mon bureau, visualisation avec Open3D

Le concept de base de cette opération est simple : on manipule un nuage —en lui appliquant des rotations et des translations—, jusqu'à ce que sa superposition avec l'autre nuage soit convaincante. Ce processus de recalage de deux nuages de points suit généralement les étapes suivantes :

1. **Sélection** d'un ensemble de points d'un ou des deux nuages.
2. **Association** de ces points à des échantillons de l'autre nuage.
3. **Pondération** de ces paires de correspondances.
4. **Exclusion** de certaines paires, considérées comme aberrantes au regard du reste des autres paires.
5. **Minimisation** d'une **métrique d'erreur** basée sur les paires de points.
6. **Répétition** de ces étapes tant que l'algorithme n'a pas convergé, c'est à dire tant que l'erreur globale estimée est supérieure à un seuil fixé. Autrement dit, on boucle jusqu'à ce que la superposition des nuages soit satisfaisante.

Afin de mieux illustrer ce processus, nous allons le réaliser à travers un exemple. La figure 2.11 présente un environnement simulé, composé de quatre piliers de mine, ainsi que deux scans

donnant les nuages P et Q. Il est important de préciser que dans cet exemple jouet, aucun bruit de mesure n'est simulé, c'est le terrain qui est irrégulier (généré de manière à mieux coller à la réalité avec un bruit de Perlin). En effet, la symétrie forte de ce type d'environnement couplée au bruit de mesure peut conduire à des scénarios où il est strictement impossible de distinguer deux nuages de points, pourtant collectés à des endroits différents. Ici, nous connaissons la position relative des deux nuages grâce à la simulation. Mais en réalité, c'est ce que nous cherchons à estimer.

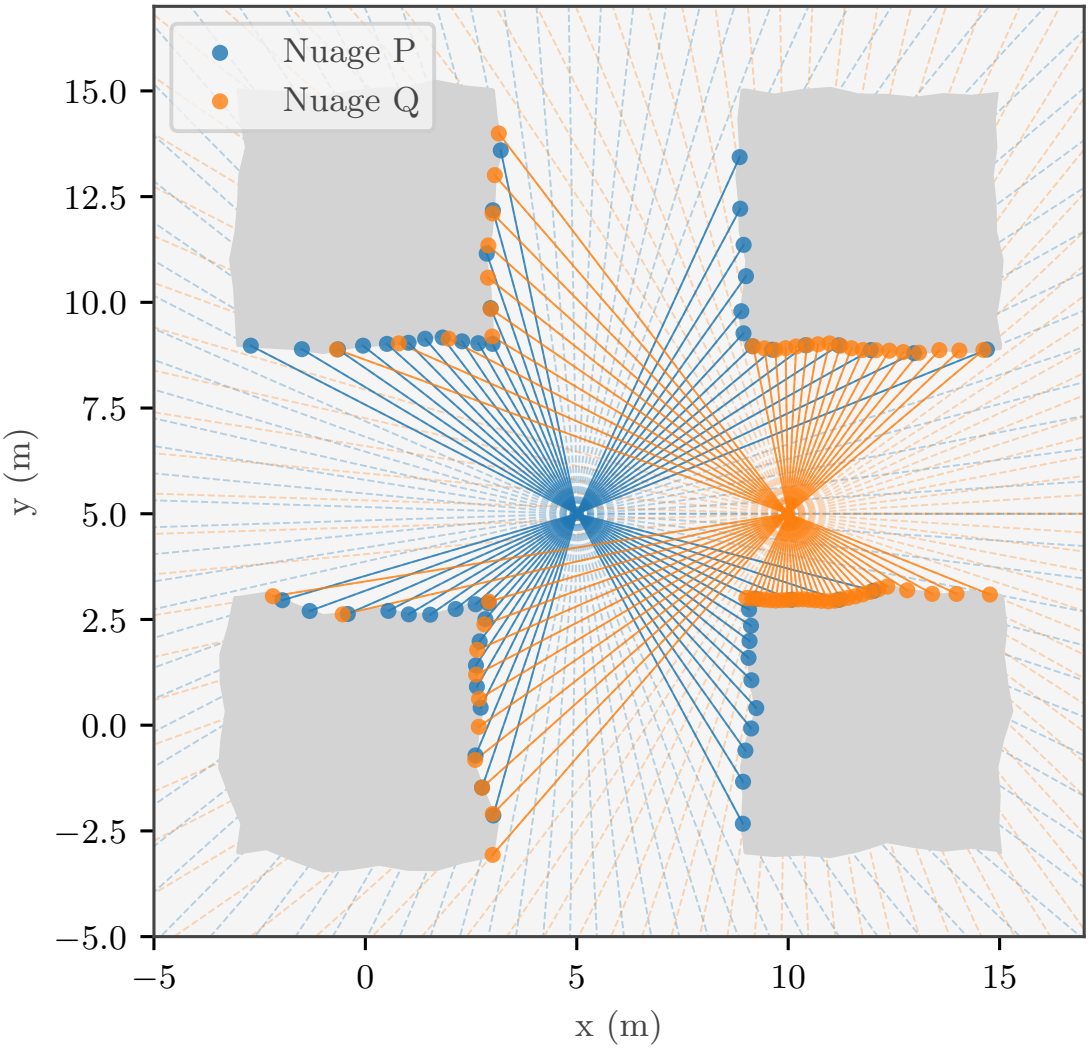

FIGURE 2.11 – Deux scans lidars d'une mine de calcaire simulée

La figure 2.12 montre les étapes successives de l'ICP classique. Dans cet algorithme simple, seule les étapes d'association et de minimisation sont répétées. Pour chaque itération, chaque point du nuage P est associé avec le point du nuage Q le plus proche. Ces correspondances donnant chacune une transformation, il est possible de déterminer une transformation de manière à minimiser la norme de ces transformations —qui sont interprétées ici comme des erreurs — entre paires de points. Chaque erreur d'association ici sera un frein à la bonne estimation de la transformation. Et dans un cas de symétrie évoqué précédemment, l'algorithme pourra

converger vers la mauvaise interprétation si l'estimation initiale est mauvaise.

La transformation déduite des correspondances est ensuite appliquée au nuage P et l'on répète ces deux opérations un nombre fixé de fois. En pratique, on préfère vérifier la convergence pour arrêter l'algorithme. Cette étape consiste soit à vérifier si la somme des erreurs est inférieure à un seuil défini, soit à constater qu'il n'est plus possible de réduire cette somme d'erreurs. Dans l'exemple, les premières itérations sont présentées, ainsi que la dernière, côte à côte avec la *vérité terrain (Gound truth - GT en anglais)* issue de la simulation (sur les deux dernières vignettes). Les triangles représentent les origines des nuages, c'est-à-dire la position du capteur qui les a collecté. Les huit premières vignettes permettent d'apprécier le processus itératif de recalage. Les deux dernières présentent le résultat obtenu par l'algorithme (à gauche) et la vérité terrain (à droite). Cette dernière comporte également la transformation estimée (en rouge) et la transformation réelle entre les deux nuages.

Il existe de **très nombreuses** variations de cette version basique, qui utilisent une combinaison et des paramétrages spécifiques, principalement pour les quatre premières étapes. Leurs objectifs sont divers, mais il s'agit globalement d'améliorer la rapidité, la tolérance au bruit, la capacité de convergence —à quel point l'approximation initiale peut être fausse— ainsi que la précision finale de la transformation calculée. Nous allons maintenant présenter les grandes approches pour chacune de ces étapes.

1. *Sélection des points :*

   Tous Il est parfois possible d'utiliser tous les points [Besl and McKay, 1992] des deux nuages. Par exemple si ces nuages sont éparses et ne contiennent pas beaucoup de points, ou encore si la puissance et le temps de calcul ne sont pas des problèmes pour l'application ciblée.

   Sous-échantillonnage uniforme (*uniform sub-sampling*) L'idée est de choisir régulièrement des échantillons dans l'espace, de manière a obtenir une densité de points similaire sur tout l'espace couvert —évidement uniquement si le nuage initial couvre également l'espace de manière uniforme, ce qui est rarement le cas en pratique. L'inconvénient de cette méthode [Turk and Levoy, 1994] est que cette sélection requiert des calculs supplémentaires, cette uniformité n'étant pas garantie en prenant un point sur dix par exemple.

   Échantillonage aléatoire (*random sampling*) L'avantage de cette approche [Masuda et al., 1996] réside dans son coût réduit, car aucun calcul n'est nécessaire pour effectuer notre sélection. Cette approche est donc largement adoptée, parfois même avant l'utilisation d'une autre méthode de sélection. En effet, la forte densité —et donc le grand nombre de points produits— permise par les capteurs actuels —notamment par les caméras de profondeur— combinée à la contrainte temps réel d'un système de SLAM force l'utilisation d'astuces permettant de ne pas dépasser les capacités de calcul limitées des systèmes cyber-physiques.

   Points distinguables (*feature-based sampling*) Cette approche de sélection [Weik, 1997] se rapproche du principe d'amers présenté dans la section 1. L'idée est de déterminer à quel point chaque point est reconnaissable, à partir des informations connues

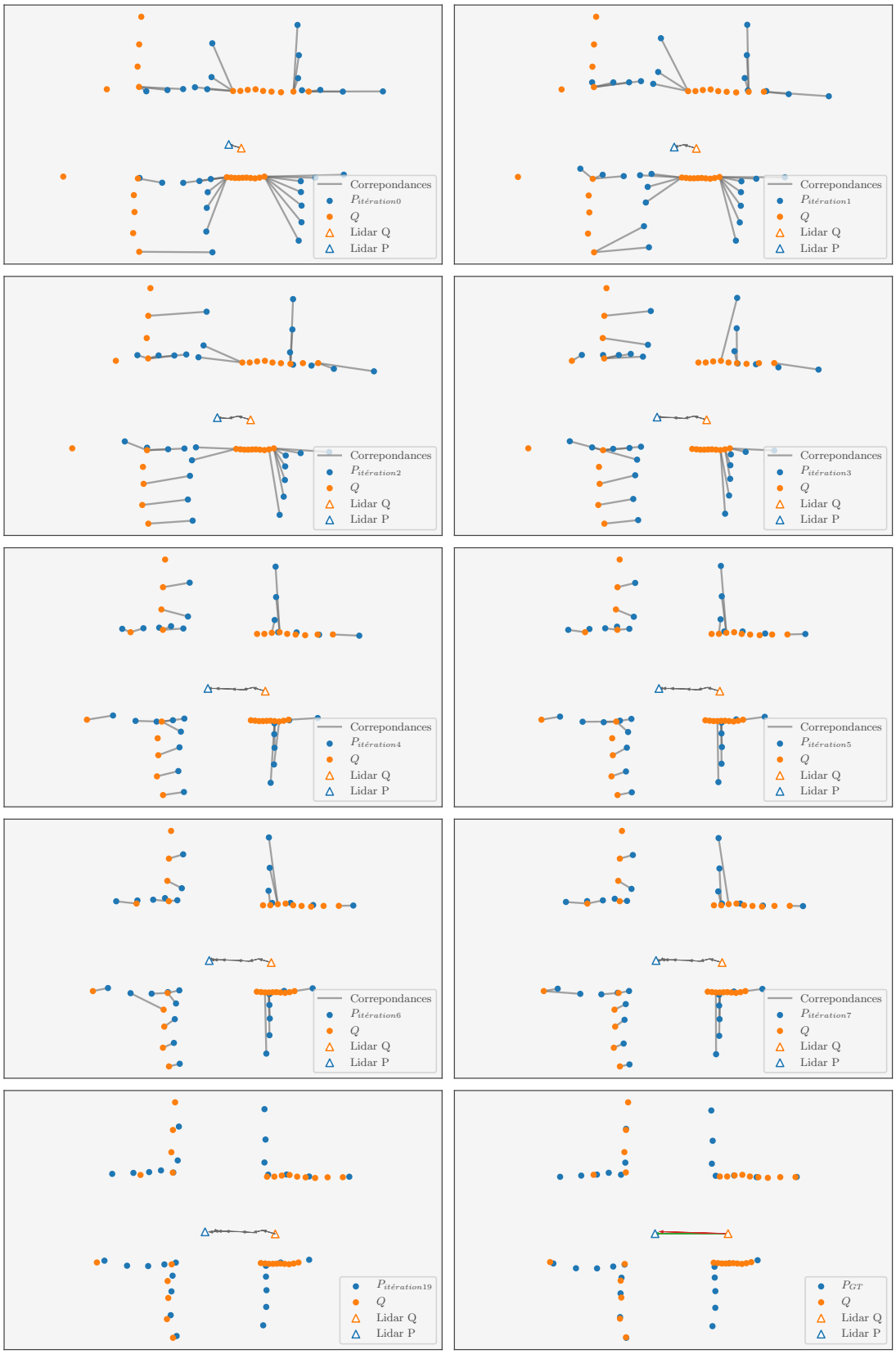

FIGURE 2.12 – Itérations successives de l'algorithme ICP classique

sur les voisins. Bien que permettant potentiellement une sélection plus pertinente de points, cette approche demande beaucoup de calculs à cette étape.

Uniforme dans l'espace normal (*Normal-space sampling*) Cette approche [Rusinkiewicz and Levoy, 2001] est comparable à l'approche uniforme, mais en considérant cette fois l'espace normal au lieu de l'espace physique. Ainsi, on essaie de garder un ensemble de points dont les normales sont uniformément distribuées. Cette approche nécessite évidement de calculer les normales des points, ce qui peut s'avérer coûteux sur de grands nuages de points. Les résultats obtenus par cette sélection sont généralement meilleurs pour des environnements lisses, avec peu de points particuliers.

2. *Association des points en paires :*

Plus proches Approche originelle, qui consiste à appairer les points les plus proches. Cette opération peut être accélérée en utilisant des algorithmes de recherche de type *kd-trees* [Simon, 1996]. Il faut toutefois de nombreuses itérations —sur toutes les étapes— pour converger, et la connaissance d'un alignement initial est nécessaire. Toutefois, il est possible d'aligner les barycentres des deux nuages pour générer cet alignement initial, solution qui en pratique est mauvaise pour un robot qui se déplace. En effet, les deux scans n'ont en réalité pas le même barycentre, car un lidar effectue généralement des mesures dans un disque autour de sa position.

Plus proches, mais compatibles Il s'agit d'appairer les points les plus proches, mais uniquement s'ils sont compatibles. Cela demande d'avoir des informations supplémentaires sur les points 3D, par exemple la couleur [Godin et al., 1994].

Tracé de normales (*normal shooting*) Consiste à tracer un rayon partant du point considéré dans la direction de sa normale, et à choisir comme correspondance l'intersection entre ce rayon et la surface de l'autre nuage de points. Cette méthode [Yang and Medioni, 1992] requiert évidement le calcul préalable de la surface à partir des points du second nuage. Les résultats obtenus sont meilleurs pour des environnements simples et lisses, mais moins bons pour des environnement complexes.

Approches projectives (*projection-based*) Connaissant une approximation initiale de la position du capteur lors du second scan, on trace un rayon partant du capteur et passant par le point que l'on cherche à appairer dans le premier nuage. Le point du second nuage sélectionné comme correspondance sera alors l'intersection entre ce rayon et la surface du second nuage [Dorai et al., 1998, Weik, 1997].

L'association de données est un point clé du processus de recalage, c'est généralement le choix qui influe le plus sur la performance finale de l'ensemble d'algorithmes qui composent l'ICP. Le choix de l'approche est intimement lié aux informations disponibles, et/ou à la puissance de calcul nécessaire à leur obtention. D'une manière générale, les méthodes d'associations utilisant les normales des points ou des surfaces donnent de meilleurs résultats.

3. *Pondération des paires :* Parfois, nos connaissances sur le capteur utilisé permettent d'évaluer la validité des correspondances obtenues. En effet, certains capteurs comme les caméras de profondeur sont plus précises dans l'estimation des objets proches que lointains. Il est ainsi possible d'accorder plus d'importance, de crédit, aux points qui ont été collectés proches du capteur, et à l'inverse moins de crédit aux mesures plus incertaines [Förstner and Wrobel, 2016]. Le calcul de l'erreur globale sera modifié en une somme des erreurs de chaque paire de points, pondérée par la confiance dans la validité de cette correspondance.

4. *Exclusion des paires aberrantes :* L'utilisation de paires erronées induit systématiquement un biais dans l'estimation de la transformation finale. Divers mécanismes ont donc été étudiés afin de limiter leur impact, en essayant de les identifier pour les mettre de coté pour l'étape de minimisation.

   Distance Rejet de la paire si la distance entre ses deux points (*point-to-point*) est supérieur à un seuil.

   Taux Rejet des pires $n\%$ des paires, selon une métrique définie, généralement la distance point-à-point.

   Statistique Rejet des paires dont la distance entre les points est supérieure à n fois la déviation standard [Masuda et al., 1996].

   Cohérence avec les voisins Deux paires (p_1, q_1) et (p_2, q_2) sont rejetées si :

   $$\| \|p_1 - p_2\| - \|q_1 - q_2\| \| > seuil$$

   Ce qui revient à : parmi les quatre points des deux paires, si les distances entres les points appartenant au même nuage ne sont pas équivalentes, on rejette les deux paires [Dorai et al., 1998].

   Ne pas utiliser les extrémités L'idée est simplement de rejeter toutes les paires qui contiennent au moins un point proche de l'extrémité de la surface à laquelle il appartient dans son nuage. Cette méthode [Turk and Levoy, 1994] est particulièrement efficace pour écarter les mauvaises correspondances lorsque les nuages sont issus de scans dont le recouvrement n'est pas complet —ce qui est systématique dans le SLAM.

Une autre méthode de minimisation de l'impact des valeurs aberrantes consiste à diminuer leur poids lors du calcul de l'erreur. Ces approches, devenues le standard de par leur efficacité —pas uniquement dans le cadre de ce problème— sont basées sur des outils statistiques (*robusts kernels* [Huber, 1992]). De nombreuses versions ont été développées, mais d'après une analyse exhaustive [Babin et al., 2019], leurs performances sont similaires pour peu qu'elles soient paramétrés correctement.

Cette étape de filtrage est généralement basée sur des heuristiques, elles mêmes déduites des connaissances des conditions dans lesquelles le robot va évoluer. Ce qui ne semble pas compatible avec notre problématique d'environnement totalement inconnu. Toutefois, une récente contribution (*Adaptative robust kernels* [Chebrolu et al., 2021]), propose d'adapter en temps réel le filtrage des valeurs aberrantes, permettant à la fois de s'affranchir des paramétrages manuels et de mieux s'adapter à un environnement dynamique.

5. *Minimiser l'erreur :* Cette étape de minimisation est un classique problème des moindres carrés, dans lequel on détermine la transformation en cherchant à minimiser la distance point-à-point de l'ensemble des paires.

$$min \sum \|q_i - p_j\|^2$$

Il existe toutefois une autre façon de calculer l'erreur que la distance euclidienne, qui permet d'y inclure les normales des points. Cette approche [Chen and Medioni, 1991], qui est devenue la référence, propose comme erreur pour une paire (p_i, q_j) donnée, la projection de la distance euclidienne sur la normale du point q_j :

$$min \sum ((q_j - p_i) \cdot n_q)^2$$

Cette méthode de calcul de l'erreur est connue pour sa convergence plus rapide, et nécessite donc moins d'itérations. Une optimisation du calcul de la transformation est proposée dans une approche nommée *Generalized-ICP* [Segal et al., 2009], dans laquelle une modélisation probabiliste de la structure des deux nuages est utilisée.

6. *Une estimation fiable de la position ?*

Nous avons donné un aperçu de l'ensemble des techniques mises en oeuvre pour déterminer la transformation entre deux nuages de points donnés. La profondeur et la diversité des solutions algorithmiques mises en oeuvre à chaque étape du processus souligne le caractère complexe d'une solution d'ICP. La complexité n'est pas nécessairement un problème en soit, mais ici la performance finale d'une solution est directement liée à la pertinence des nombreux choix de conception effectués lors de ces étapes. Renforcée par l'interdépendance des étapes —changer un paramètre/algorithme a des répercussions sur toutes les autres étapes—, cette complexité devient difficile à maîtriser. Ainsi, développer et mettre en oeuvre un tel système demande une adaptation forte —qui ne peut être réalisée que par un spécialiste— aux spécificités du terrain ou il sera déployé.

Ces algorithmes sont-ils employables sur le terrain ? Leur robustesse est elle suffisante pour résister aux conditions des applications réelles auxquelles ils sont destinés ? Ce sont les questions posées dans une publication [Pomerleau et al., 2013] visant à poser les bases de l'évaluation de telles solutions. Leur conclusion est sans appel :

> The point-to-plane solution can be stable for applications where : first, the environment type can be controlled to be highly structured ; second, the overlap is kept high while the robot is moving and third, the state estimation used as initial pose for the registration remains within $10cm$ and 10°. These types of conditions are usual for laboratory experiments but are unlikely to happen in real applications. — François Pomerleau · Francis Colas · Roland Siegwart · Stéphane Magnenat

> La solution point à plan peut être stable pour les applications où : premièrement, le type d'environnement peut être contrôlé pour être hautement structuré ; deuxièmement, le chevauchement est maintenu élevé pendant que le robot se déplace et troisièmement, l'estimation de l'état utilisée comme pose initiale pour le recalage reste dans les limites de $10cm$ et 10°. Ces types de conditions sont habituelles pour les expériences en laboratoire, mais peu probables dans les applications réelles. — François Pomerleau · Francis Colas · Roland Siegwart · Stéphane Magnenat

Nous avons vu comment produire des estimations de déplacements au fur et à mesure d'une expérimentation, mais qu'en est-il de l'estimation de la position actuelle du robot ? De manière analogue à l'odométrie visuelle où inertielle, il est possible de calculer de manière récursive la position actuelle du robot en ajoutant la transformation estimée entre les deux nuages à la précédente estimation de la position. Cette solution souffre donc de la même accumulation d'erreurs d'estimations, cette fameuse dérive.

2.3.4 Problèmes récurrents

A travers l'exploration des grandes approches d'odométrie, nous avons constaté l'existence de problèmes récurrents dans les estimateurs existants. Premièrement, le **problème d'association de données**. Lorsque nous observons l'environnement, nous n'en percevons qu'une image, pas la réalité. il est alors impossible, d'après deux observations différentes d'établir avec certitude qu'il s'agit du même objet. Or nous l'avons vu, une telle erreur de jugement a des conséquences irrémédiables sur l'estimation actuelle et future de la position, et par la même occasion sur la qualité du modèle reconstruit.

Nous l'avons vu, des solutions aussi nombreuses que variées ont été mises en place pour tenter de limiter l'impact de ces associations erronées. Cela a plusieurs conséquences directes : tout d'abord, la **complexification** de ces systèmes d'odométrie. En effet, toutes les approches consistent en un enchevêtrement d'algorithmes, nécessitant de nombreux choix de conception aux conséquences imprécises. Cela implique le recours à des spécialistes, autant pour la mise en oeuvre que pour l'exploitation de ces solutions. De plus, c'est une gageure de garantir le bon fonctionnement de tels systèmes dans un nouvel environnement, pour lequel il faudra probablement une phase d'adaptation et de paramétrage.

Résultat direct de cette complexification, l'**augmentation de la puissance de calcul nécessaire**. Cela n'est pas nécessairement un problème lorsque l'objectif consiste à reconstruire un environnement à partir de données collectées au préalable. Toutefois, quand il s'agit de fournir une estimation de position et un modèle de l'environnement en temps-réel, pour la navigation d'un robot autonome par exemple, ce coût en calculs devient un frein majeur.

2.4 Prendre en compte l'incertitude (*back-end*)

Bien qu'étant basés sur des principes de mesure très différents, tous les capteurs sont sujets au bruit de mesure. Il est aisé de modéliser les problèmes, notamment de robotique, de sorte à ce qu'ils soient simples à résoudre si ces bruits sont statistiquement indépendants. Il suffirait alors de multiplier les mesures pour compenser les effets du bruit. Malheureusement, et particulièrement dans les problèmes de SLAM, les erreurs de mesure sont statistiquement dépendantes. Cela est dû à l'accumulation des erreurs au fil du temps, ce qui influence la façon dont les futures observations seront interprétées. C'est pour tenter d'atténuer ces erreurs qu'une multitude de mécanismes de fusion et d'intégration des données ont vu le jour.

Dans l'écrasante majorité des applications en robotique, l'incertitude, si elle est modélisée, l'est de manière probabiliste. La figure 2.13 illustre cette modélisation dans le cadre d'un problème de positionnement 2D. Sur la vignette de gauche est représentée l'estimation de position

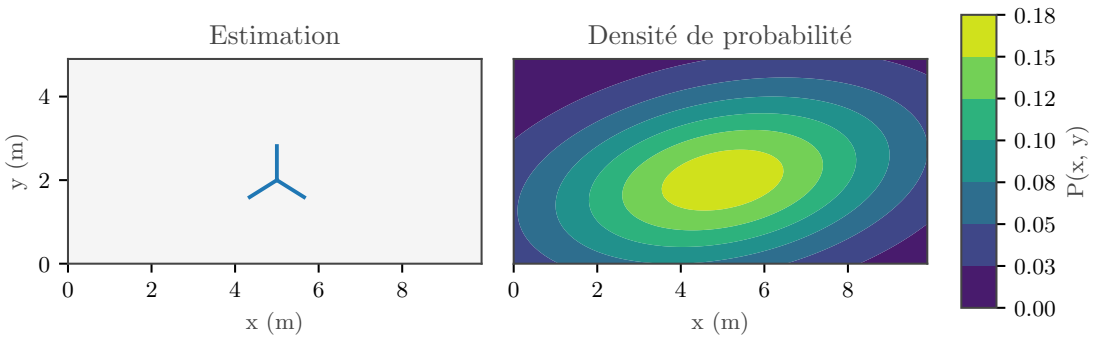

FIGURE 2.13 – Modèlisation probabilite de l'incertitude

fournie par un système quelconque. Cette information est bien évidemment primordiale, on a besoin de cette estimation, toutefois elle ne permet ni de savoir si on se trompe, ni à quel point on se trompe. La modélisation de l'incertitude de la mesure à travers une densité de probabilité —suivant ici une loin normale multidimensionnelle— permet de calculer des intervalles de confiance, représentés par les ellipses dans la vignette de droite. Dans le cas 2D, il s'agit d'une aire, pour laquelle il est possible de déterminer la probabilité que notre robot soit réellement à l'intérieur. Cette densité est centrée sur l'estimation, et ce centre μ est appelé moyenne. Une matrice de covariance Σ donne quant à elle la direction et l'intensité de sa diffusion dans l'espace. La détermination précise —qui représente correctement la réalité— de cette matrice n'est pas nécessairement une chose aisée, surtout quand les estimations sont produites par des algorithmes complexes ou pire, des ensembles complexes d'algorithmes complexes. Pour peu que cette densité de probabilité modélise correctement les erreurs potentielles, il devient alors possible de filtrer les observations selon leur plausibilité.

En plus de cette capacité de filtrage, la modélisation probabiliste permet la combinaison —la fusion— de données issues de différents capteurs, en prenant en compte les spécificités de chacun. Intégrer ces différentes informations permet d'affiner les estimations produites par différents *front-ends* de manière à produire une estimation plus précise que ce qu'aurait pu fournir même la plus précise d'entre elles.

2.4.1 Filtrage Bayésien

De manière analogue à la construction récursive de l'estimation de position, abordée dans les différents estimateurs *front-end*, les algorithmes que nous allons voir dans cette section s'appuient sur l'estimation précédente —ici son modèle probabiliste— et sur le modèle de la nouvelle observation pour mettre à jour l'estimation de position et sa confiance associée. Il existe une large littérature sur ces problèmes dit de *filtrage Bayésien* qui dépasse largement le cadre de la robotique. Nous allons toutefois tenter d'en dessiner les contours, en commençant par un bref historique.

La compétition féroce entre les blocs lors de la guerre froide et la course à l'espace qui en a résulté ont dopé la recherche, particulièrement dans le domaine du contrôle. C'est dans ce contexte que les filtres Bayésiens ont vu le jour. Ces filtres —où estimateurs récursifs d'états— ont pour objectif d'estimer un état inconnu, à partir des observations faites au fil du temps. Il

s'agit donc d'estimer l'état actuel d'un système, par exemple sa position. Le principe consiste dans un premier temps à prédire, à partir des commandes de contrôle et du modèle du système, l'état futur du système. Puis à comparer cette prédiction avec la prochaine observation, afin de corriger le modèle de prédiction.

La plus célèbre de ces méthodes, le filtre de Kalman [Kalman, 1960], a été utilisée dans les années 1960 pour la navigation du projet Apollo, pour laquelle il fallait estimer la trajectoire du vaisseau spatial lors de son aller-retour vers la lune. Produisant des estimations suffisantes pour les standards de l'époque, cette méthode s'appuie cependant sur deux hypothèses fortes : le monde suit une loi normale, et le système étudié est linéaire. Or dans le monde réel, le bruit de mesure ne suit pas nécessairement une distribution normale, et le problème de positionnement n'est pas un problème non plus obligatoirement linéaire. Diverses approches tentent de contourner ces problèmes, comme l'*Extended Kalman Filter (EKF)* qui linéarise localement le système pour pouvoir employer le filtre classiquement.

D'autres approches, basées sur l'échantillonnage des états possibles du système et leur validation ou invalidation par prédictions et corrections successives permettent à la fois de contourner —plus ou moins efficacement— le problème de linéarisation et la limitation originelle aux distributions Gaussiennes. Leur principe est simple, après sélection —aléatoire ou non selon les méthodes— d'un échantillon des états possibles (dans notre cas des points de l'espace) on détermine leur probabilité, c'est à dire à quel point ils sont compatibles avec les observations actuelles. On écarte les points peu probables, et on les *propage* : on leur applique une transformation en fonction de notre prédiction du déplacement actuel, puis on répète le processus. La figure 2.14 permet d'illustrer ce principe à travers un exemple. Ce dernier est inspiré du jeu de plateau *Captain Sonar*, dans lequel deux équipes s'opposent et doivent localiser le sous-marin adverse à partir uniquement des informations de déplacements, criées par les capitaines à leurs équipes. Les sous-marins se trouvent nécessairement sur une grille 2D. À t_0, un échantillonage des états possibles est réalisé, représenté par les points colorés. Une probabilité est associée à chaque échantillon, représentée ici par la taille du point. Puis, on propage dans le temps l'ensemble des échantillons, en suivant le déplacement *Nord* vociféré par le capitaine adverse. Les particules peu probables —un sous-marin terrestre— sont écartées, diminuant le nombre de particules, et donc d'état possibles du système, tout en augmentant la probabilité des particules viables. On répète le processus plusieurs fois (ici le sous-marin fait toujours cap au nord), en espérant que les observations permettent de réduire le nombre de particules jusqu'à atteindre une probabilité suffisante pour ne considérer qu'une seule particule comme celle représentant le système. Le filtre a alors convergé, comme à t_3. Ces approches, comme *sigma-points* [Van der Merwe and Wan, 2003], *Unscented Kalman Filter (UKF)* [Wan and Van Der Merwe, 2000] et plus globalement les *particle-filters* [Arulampalam et al., 2002] ont gagné en popularité dans les dernières décennies.

2.4.2 Lissage (*smoothing*)

Les approches récursives abordées précédemment ne se basent que sur le dernier état connu pour déterminer la nouvelle estimation, ce qui les prive d'une quantité importante d'informations pourtant disponibles. D'autres stratégies s'emploient à utiliser l'ensemble des états précédents du système. Initialement nommées *full-SLAM* [Thrun, 2002] ou encore *bundle-adjustment*, ces approches calculent après coup (*offline*) les corrections sur l'ensemble des estimations faites. Il est en théorie possible de les appliquer à chaque nouvelle observation pour générer une esti-

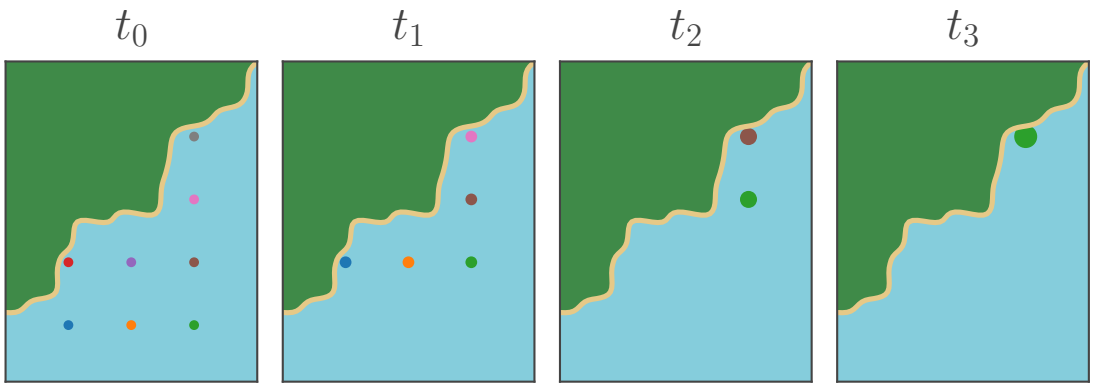

FIGURE 2.14 – Filtre à particules appliqué au jeu de plateau *Captain Sonar*

mation optimale basée sur l'ensemble des observations passées. Cependant, leurs exigences en terme de puissance de calculs prohibe toute approche d'estimation en temps réel. C'est pourquoi bien que fournissant de meilleures estimations que les filtres récursifs, leur emploi n'a pendant longtemps pas été possible dans le cadre de la robotique autonome.

L'idée d'énoncer le problème d'optimisation du *full-SLAM* en terme d'algèbre linéaire éparse [Dellaert and Kaess, 2006] a permis de diminuer de manière drastique la quantité de calculs nécessaire pour les résoudre. En effet, cela ouvre la possibilité de s'appuyer sur les nombreux travaux en algèbre linéaire —eux mêmes liés à la théorie des graphes— pour exploiter pleinement la caractéristique **éparse** du problème de SLAM. Si l'on représente le problème avec un graphe, comme sur la figure 2.4, il devient aisé de se rendre compte que le degré des noeuds reste faible, et que le graphe résultant sera faiblement connecté. Il devient alors possible d'estimer, et ce en temps-réel, l'état actuel d'un système, plus seulement à partir de la dernière estimation mais de tout l'historique. Cette exploitation de toutes les données a cependant des désavantages, le principal étant l'augmentation progressive de la quantité de données à traiter, et donc du temps de calcul nécessaire à l'intégration d'une nouvelle observation. Cette augmentation continue du temps de calcul rend difficile à tenir la contrainte temps-réel lors de missions prolongées, où le temps d'inférence —fait de produire une estimation issue du modèle probabiliste— devient trop important.

C'est dans cette optique qu'on été développés des mécanismes de réorganisation des variables pour conserver au maximum la caractéristique éparse du graphe [Kaess et al., 2008]. Plus tard, l'approche sera améliorée par la conception d'une nouvelle structure de données —un *Bayes-tree* [Kaess et al., 2012]— permettant d'effectuer directement l'inférence sans passer par la représentation matricielle et les opérations d'algèbre linéaire. Cette structure permet également de profiter des perspectives de parallélisation des calculs offertes par la multiplication des coeurs dans les processeurs modernes.

Ces travaux n'ont pas seulement abouti à une formidable théorie permettant la modélisation et la résolution optimale du problème de l'estimation d'état d'un système en temps réel. En effet, ces avancées théoriques sont accompagnées d'une bibliothèque logicielle qualitative, activement maintenue et richement documentée[4]. À tel point que le problème d'optimisation de l'estimation (*back-end*) est généralement considéré comme résolu, et cette approche —ou des

4. Georgia Tech Smoothing and Mapping Library https://gtsam.org/

approches équivalentes— est utilisée par une majorité des systèmes de SLAM actuels.

2.5 Où contribuer ?

Nous avons vu les différentes briques logicielles qui composent généralement un système de SLAM. Mais quelles sont les stratégies pour les combiner ? Dans ce domaine, chaque approche à sa recette, il serait donc difficile de toutes les décrire. Cependant, il existe certaines tendances. L'écrasante majorité des solutions utilisent une *back-end*, qu'il s'agisse d'un filtre Bayésien, ou d'un *factor-graph*. Nous l'avons vu, cette modélisation probabiliste permet d'utiliser divers capteurs. La majorité des approches actuelles capitalisent sur cette possibilité et construisent des systèmes multi-modaux. Ils comportent très largement une centrale inertielle, couplée avec une caméra ou un lidar. Parfois même les trois. L'odométrie visuelle-inertielle *visual-inertial odometry (VIO)* semble toutefois concentrer les nouvelles recherches. Ce phénomène est une conséquence directe à la fois de la bonne complémentarité de ces deux capteurs et de leur accessibilité.

Après avoir dressé un paysage du domaine de recherche sur le SLAM, il est temps de répondre à une question essentielle : où contribuer de manière pertinente, en gardant en tête notre contexte spécifique de l'après-mine ?

Il s'est rapidement avéré, en prenant la mesure de la qualité et de l'aboutissement —tant théorique que logiciel— des solutions *back-end*, qu'il n'était pas pertinent de se concentrer sur ce domaine. Autre facteur déterminant, ces solutions —bien qu'optimales théoriquement— sont tributaires des informations fournies par les estimateurs *front-end*. Or ces derniers, bien que fournissant naturellement des estimations, produisent bien plus difficilement la confiance associée. La production de ces matrices de covariances permettant l'élégante prise en compte des incertitudes lors de la phase d'optimisation est loin d'être fiable. En effet, bien que les erreurs d'amplitudes soient déjà problématiques, certaines approches comportent également des erreurs de directions. Dans les pires cas, l'optimisation renforcera l'erreur au lieu de la compenser.

Il serait alors possible de chercher à améliorer les estimateurs existants dans leurs capacités à estimer leur propre erreur. Considérons toutefois les contraintes spécifiques de l'après-mine que nous nous sommes fixés. Nous avons vu que les estimateurs existants, qu'ils soient basés sur des centrales inertielles, des caméras, des lidars ou les trois ne répondent pas à ces contraintes. Cela pour plusieurs raisons. Premièrement, il s'agit de problèmes complexes, dont la conception et le paramétrage sont basés sur des hypothèses et des informations disponibles sur l'environnement. Or notre contexte prohibe de telles pratiques, ce qui mécaniquement a écarté cette piste.

Nous avons préalablement écarté les algorithmes d'adaptation du modèle ou *post-traitement*, car n'étant pas strictement nécessaires à la robotique autonome et donc pas prioritaires. Nous avons également écarté les détections de boucles ou *loop-closure*, car s'appuyant sur les estimateurs existants et nécessitant un système de SLAM opérationnel, ce dont nous ne disposions pas. Quelles pistes reste-il ?

Au regard des observations précédentes, la contribution évidente se situe dans la conception d'un nouvel estimateur, en rupture avec les estimateurs de position existants. Nous avons vu que les approches classiques souffraient de divers maux : solutions complexes, hypothèses fortes sur

l'environnement et problèmes systémiques d'associations de données. C'est décidé, ce sont ces trois problèmes que nous allons tenter d'adresser à travers la conception d'un nouvel estimateur de position.

Deuxième partie

Contribution : positionnement

Table des matières

Chapitre 3
NAPS : Un système de positionnement nomade

3.1 Détermination de la nouvelle approche . . . 51
3.1.1 Vers un maillage de positionnement . . . 54
3.1.2 Vers une mesure en coopération active . . . 55
3.2 NAPS . . . 57
3.2.1 Réalisation du maillage temporel . . . 57
3.2.2 Détermination de la position globale . . . 58
3.3 Implémentation proposée . . . 60
3.3.1 Architecture de NAPS . . . 60
3.3.2 Architecture de validation . . . 62
3.3.3 Un développement difficile . . . 63
3.4 Expérimentations . . . 65
3.4.1 Protocoles expérimentaux . . . 65
3.5 Résultats . . . 67
3.5.1 Analyse qualitative . . . 67
3.5.2 Analyse quantitative . . . 70
3.6 Conclusion . . . 71

Chapitre 4
Un cas d'usage : la cartographie 3D

4.1 Vers une solution de SLAM complète . . . 75
4.2 Lidar, nuage, et recalage . . . 75
4.2.1 D'un ensemble de nuages vers un modèle . . . 75
4.2.2 Opération de recalage . . . 76
4.3 Architecture du système de cartographie . . . 77
4.4 Logique de recalage et d'amélioration des nuages . . . 79
4.4.1 Mécanisme d'association temporelle . . . 79
4.4.2 Filtrage en fonction de la vitesse . . . 79
4.4.3 Diminution du bruit statique . . . 79
4.4.4 Filtrage des éléments du système . . . 80
4.5 Expérimentations et Résultats . . . 80

4.5.1 Étude qualitative 80
4.5.2 Étude quantitative 84
4.6 Améliorations 84
4.6.1 ICP et raffinement Bayésien 84
4.6.2 Vers d'autres marqueurs 86
4.6.3 Contrôle robotisé du processus d'exploration 88
4.7 Conclusion 89

Chapitre 3

NAPS : Un système de positionnement nomade

3.1 Détermination de la nouvelle approche

La mesure de positions et l'établissement de cartes ne date pas d'hier. En effet, on estime que l'apparition de l'arpentage et de la géométrie coïncide avec l'invention de l'écriture, en Mésopotamie vers 3000 ans avant J-C. Des tablettes d'argiles sumériennes datées du XIV - XIIIe siècles avant notre ère représentant des parcelles foncières et leurs accès attestent de la pratique de la cartographie. L'une de ses tablettes du Penn Museum est présentée sur la figure 3.1. Mais comment procédaient-ils pour créer ces cartes ?

Équipés de simples piquets en bois servant de références et de cordes graduées pour les mesures de distances, ils reportaient ensuite leurs observations à l'échelle sur les tablettes, grâce à une règle. Les sources sont floues, mais ils semblerait qu'il a fallut attendre les Babyloniens pour voir émerger la notion d'angle. L'origine du découpage en 360°, parfois attribué à la définition de 1° comme la fraction d'angle correspondant à mouvement du ciel entre deux nuit consécutives. Une autre explication vient de leur système sexagésimal de numération, ils auraient alors divisé l'angle d'un triangle équilatéral en 60 unités. Il est vertigineux de constater que bien que la technique ait énormément évolué, les principes de base qu'utilisent toujours les géomètres aujourd'hui étaient déjà maîtrisés il y a 3000 ans.

D'ailleurs, comment sont réalisées les cartes de nos jours ? J'ai été surpris de constater que les terrains sont toujours légalement définis par rapport au réseau de bornes géodésiques. Une borne géodésique est un repère permanent, qui marque l'emplacement d'un point géodésique —point dont la position précise sur Terre est déterminée par triangulation. La figure 3.2 présente une telle matérialisation. Chacune de ces bornes est référencée par rapport à ses voisines, formant ainsi un réseau géodésique. Le Réseau Géodésique Français (RGF93) est composé de $80k$ bornes qui forment un maillage précis permettant par la suite de localiser les éléments par rapport à ces bornes. L'idée de réaliser un système de positionnement par maillage —référencement mutuel des points— est intéressante, toutefois une borne n'étant pas identifiable, il est toujours possible de réaliser des erreurs d'association. Or nous l'avons vu, de telles erreurs ont un impact fortement négatif sur la précision des estimations obtenues. D'autres systèmes sont-ils exempts de ce défaut ?



FIGURE 3.1 – Carte summérienne gravée dans une tablette d'argile, Penn Museum objet CBS13885.



FIGURE 3.2 – Borne du RGF au sommet du Mont Ventoux, Arbautjc

La création d'amers artificiels n'est pas une idée nouvelle, particulièrement dans le domaine de la navigation maritime. En effet, on retrace l'apparition des premiers phares à l'Antiquité. Initialement de simples bûchers placés sur des éperons rocheux ou des bâtiments, ces derniers ont connus de nombreuses améliorations techniques, mais il fallu attendre 1757 pour que l'ingénieur suédois Jonas Norberg entreprenne la construction d'un fanal équipé de miroirs tournants. Les miroirs permettent de limiter l'angle de diffusion de la lumière du phare, et leur rotation permet de changer la direction de cette diffusion. Cela crée ainsi une alternance pour les navires entre des périodes d'obscurité et de lumière. Le rythme —la fréquence— à laquelle cette alternance s'effectue permet d'identifier avec certitude le phare. Ce principe développé au XVIIIe siècle a depuis longtemps résolu le problème de mauvaises associations d'amers.

Dans le contexte de notre application de surveillance des mines, l'objectif de précision doit permettre de détecter des mouvements de roche dans les modèles reconstruits. Il est donc nécessaire d'avoir une très bonne précision afin de pouvoir détecter au plus tôt les mouvements internes. En effet, le temps de détection d'un mouvement est inversement proportionnel à la précision de mesure.

Nous avons donc dégagé une piste intéressante, qui consiste à étudier la faisabilité d'un système de positionnement coopératif par maillage, qui intégrerait un mécanisme d'identification des pairs. La construction d'un réseau de phares dans l'environnement à explorer s'éloigne toutefois quelque peu de notre problématique initiale d'exploration autonome…

3.1.1 Vers un maillage de positionnement

Se pose alors la question de la transposition de ces principes avec des systèmes cyber-physiques. Le premier de ces principes, le maillage ou réseau de positionnement est directement transposable en construisant un réseau de balises. C'est le cas de certains systèmes comme le robot SPOT [5] de Boston Dynamics, qui utilise un réseau de QRcodes pré déployé et positionné. L'observation d'un de ces repères géoréférencé lui permet alors de mettre à jour sa position et ainsi corriger la dérive de son système d'odométrie interne. C'est une transposition directe du système de bornes géodésiques, mais le système ainsi formé n'est pas autonome et nécessite une fois de plus un travail préalable, et une grande maîtrise de l'environnement. Afin d'éviter de pré-équiper l'environnement, il est possible d'utiliser des agents mobiles —des systèmes cyber-physiques— pour créer des bornes géodésiques temporaires. Ces bornes seront localisées par les bornes dont on connaît déjà la position, et serviront à localiser à leur tour d'autres bornes. Ce **maillage temporel** permet de suivre le principe du réseau de positionnement, tout en limitant fortement le nombre d'agents nécessaires à sa concrétisation. La figure 3.3 présente ce concept à travers différentes étapes exploitant 5 robots. Dans la première vignette, trois robots sont positionnés en A, B et C, points dont nous connaissons la position. Dans la seconde, deux autres robots sont positionnés en D et E, puis on calcule leurs positions (par triangulation par exemple, comme pour les bornes géodésiques). Les robots occupant précédemment les positions A et B se déplacent maintenant vers les points A' et B', dont la position est estimée depuis les points C, D, et E. Ce mécanisme permet d'implémenter le principe de réseau de positionnement avec un nombre restreint d'agents mobiles. Toutefois il est toujours possible ici de faire des erreurs d'association, particulièrement avec un flotte de robots homogènes.

5. https://www.bostondynamics.com/spot

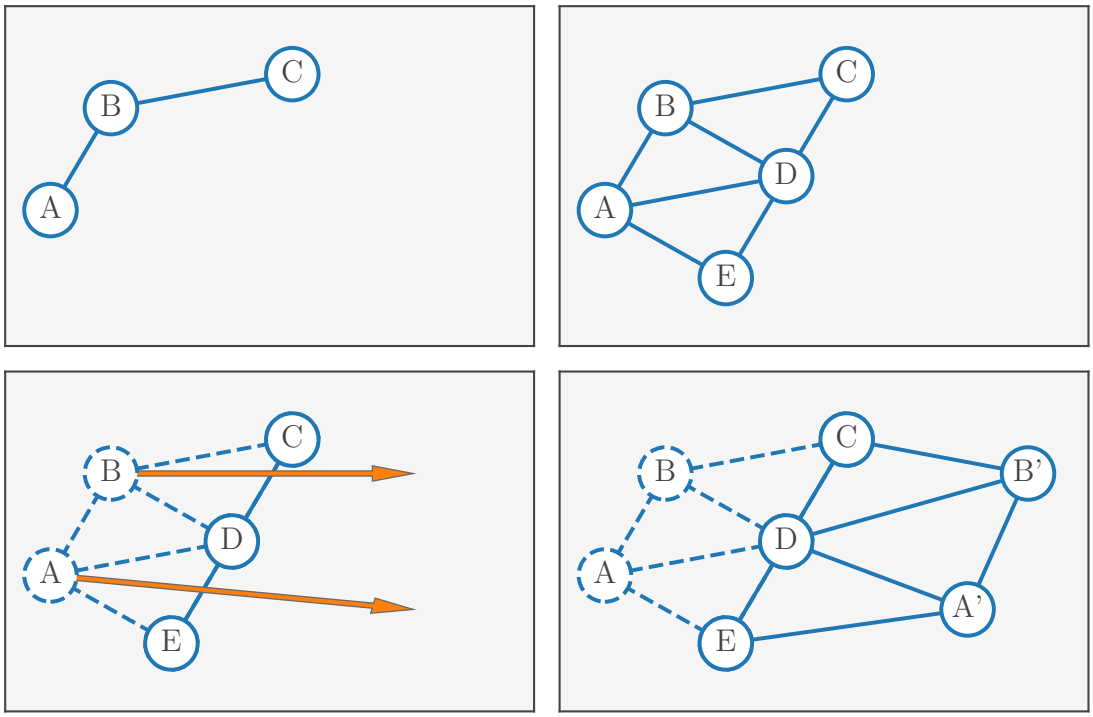

FIGURE 3.3 – Processus de maillage temporel, les flèches oranges représentant des mouvement d'éléments

3.1.2 Vers une mesure en coopération active

Le second principe inspiré des phares, permettant de prévenir totalement les erreurs d'association n'est possible qu'en raison du caractère actif des deux éléments lors d'une observation. En effet, le bateau qui observe le phare effectue une lecture active à la fois de l'information de direction et de l'information d'identité, elle même **volontairement transmise** par le phare. Ainsi, cette mesure est coopérative, dans le sens ou l'observateur lit une information envoyée par la référence —ici le clignotement du phare. Mais comment reproduire ce schéma avec des systèmes cyber-physiques ? La réponse naturelle lorsque l'on pense à la transmission est bien la communication sans fil. De très nombreuses approches de localisation appliquées à l'*internet of things (IoT)*, appelées *location of things (LoT)* [Wanderley et al., 2019] ont été étudiées. Cependant, la précision de ces solutions est loin d'être suffisante pour l'application que nous ciblons de robotique autonome. Étant basées sur des antennes, initialement non conçues pour mesurer des grandeurs physiques comme des distances où des angles, ces approches ne permettent ni un positionnement précis, ni un positionnement fiable des éléments. Ces dernières années, des solutions basées sur l'*ultra wideband (UWB)* [Mazhar et al., 2017] se sont imposées dans le domaine de la localisation intérieure. Basées sur trilatération à partir des estimations de distances généralement calculées en fonction du temps de trajet des ondes, cette méthode permet d'atteindre dans de bonnes conditions une précision de l'ordre de $10cm$. Toutefois, cette précision n'est toujours pas suffisante, particulièrement lorsque l'on prend en compte la propagation de l'erreur due au principe de maillage temporel.

Mais alors, quels capteurs —émetteur et récepteur— utiliser pour effectuer ces mesures coopératives où l'erreur d'association n'est pas possible ? Nous avons dans un premier temps imaginé un système composé de diodes éléctroluminescentes (LED), émettant des impulsions lumineuses à une fréquence donnée dans toutes les directions, et d'une caméra capable de distinguer ces clignotements. L'association serait alors impossible à rater. Toutefois en 3D, localiser une seule LED dans l'image ne permet pas d'en connaître la position, seulement de savoir qu'elle se situe sur la demi-droite partant de la focale de la caméra et passant par cette LED. Pour obtenir la position, il existe plusieurs solutions : la première consiste à effectuer la même mesure depuis un autre point de l'espace, donc à trianguler la position. Cette autre position peut également être l'autre optique d'une même caméra stéréo. Cette technique requiert toutefois un angle et une translation minimales entre les deux points de vue afin d'obtenir une estimation précise de la profondeur. Une autre solution consiste à suivre plusieurs LEDs dont on connaît le positionnement relatif. Ainsi, il est possible en résolvant le classique problème *Perspective-n-Points (PnP)* [Horaud et al., 1989], d'obtenir à la fois la position relative entre la caméra et l'ensemble de LEDs, mais également l'orientation de cette *matrice* de LEDs dans le repère de la caméra. Par la suite, nous ferons référence à cet ensemble rigide de LEDs sous la dénomination de **marqueur**. En effet, intégrer cet élément sur n'importe quel système permet de le marquer de manière unique, et d'en réaliser le suivi, et ce sans autre source lumineuse. Les versions les plus abouties de ces approches [Li et al., 2012] permettent, en utilisant une vingtaine de points, d'obtenir une estimation précise de l'orientation $\approx 0.5°$, mais proposent mécaniquement —c'est la donnée la moins contrainte du problème— une erreur en translation élevée, de l'ordre de 0.5% de la distance évaluée. Cette erreur peut paraître faible, mais elle correspond à $5cm$ d'erreur pour une mesure à $10m$, trop importante donc pour notre application.

Nous avons alors envisagé de doubler cette caméra d'un télémètre laser, permettant une acquisition précise de la distance entre les robots. Ainsi, une fois l'orientation entre deux éléments déterminée de manière précise grâce au triplet caméra-LEDs-PnP, il serait possible d'orienter le télémètre laser du robot observateur —qui comporte caméra et télémètre— dans la direction du robot observé —qui comporte les LEDs— afin de produire une estimation précise de la translation entre les deux robots. Cette solution comporte en revanche des limitations purement techniques, liées notamment au faible cône de diffusion du télémètre, qui doit alors être monté sur une tourelle capable de viser précisément une direction, et ce sur un angle le plus grand possible, idéalement 360°. Le développement d'un tel système nécessite toutefois un travail spécialisé d'ingénierie, pour lequel nous n'avons ni les ressources, ni les compétences.

Lors de nos recherches d'alternative, nous avons envisagé l'utilisation des systèmes de suivi pour la réalité virtuelle. En effet, ces systèmes sont bons marchés, et offrent une précision submillimétrique. Nous disposions de modèles issues de la collaboration entre *HTC* et *Valve*[6]. Leur principe est assez similaire à celui du triplet caméra-LEDs-PnP. Le rôle des caméras et des LEDs y est inversé. La caméra est remplacée par une station laser qui illumine la pièce selon deux axes, et les LEDs par des récepteurs capables de décoder les informations diffusées dans la pièce par le balayage laser. La transmission et la récupération de ces informations permet au marqueur[7] de résoudre le problème PnP, générant ainsi une estimation de la pose (translation et rotation) relative entre la station laser et le marqueur. Cette estimation est réalisée avec une fréquence relativement élevée de 250Hz qui permet un suivi continu des marqueurs. Chaque

6. Site du système de suivi https://partner.steamgames.com/vrlicensing
7. Modèle de marqueur choisi https://www.vive.com/fr/accessory/tracker3/

station laser couvre un espace 3D de forme pyramidale délimité par une portée maximale (~5m) et deux ouvertures angulaires verticale (110°) et horizontale (150°). La précision de l'estimation de position fournie est homogène à l'intérieur de cet espace. Une seconde base laser peut être ajoutée afin d'améliorer l'estimation de translation, corrigeant partiellement le problème évoqué précédemment. Nos premières évaluations de performances nous ont conduit à pousser plus loin la possibilité d'utiliser ce système comme instrument de mesure locale dans notre maillage de positionnement global.



FIGURE 3.4 – Photographie des éléments du système de suivi local

3.2 NAPS

Nous avons déterminé dans le chapitre 2 qu'il était nécessaire de concevoir un système de positionnement d'un type nouveau pour s'affranchir de la dépendance à l'environnement. Nous avons également identifié que la combinaison de mesures locales —effectuées en coopération active— pour former un maillage temporel permettrait d'obtenir un système de positionnement global indépendant de l'environnement. Nous avons également sélectionné par commodité un système commercial capable de fournir une transformation entre deux éléments, en coopération active et donc non sujet aux erreurs d'association. Il s'agit maintenant de déterminer comment, à partir de ces transformations locales, générer ce maillage temporel de positionnement.

3.2.1 Réalisation du maillage temporel

La figure 3.3 présentée précédemment donne une bonne idée de la procédure de maillage temporel générale, mais ne prend pas en compte la spécificité des capteurs employés. La section 6.1 détaillera ces modèles ultérieurement, mais pour l'instant nous nous contenterons de définir le

cône de diffusion des stations lasers —que nous appellerons **phares**, car ils jouent exactement le même rôle. Ce cône est défini de la même manière que le cône de vision d'une caméra, avec deux angles α et β représentant l'ouverture et une portée maximale p. Le principe général de maillage temporel est appliqué dans la figure 3.5 aux spécificités du système de positionnement local. Ce processus est décomposé en deux types de phases distinctes. Durant le premier ($phases_{0,1}$), les phares sont fixes, et fournissent le suivi des marqueurs qui se déplacent librement dans la zone de diffusion. À la fin de ces phases, un des marqueurs (le $marqueur_2$ dans notre exemple) s'immobilise à la frontière de la zone de diffusion, proche de la direction dans lequel le système souhaite se déplacer. Sa pose est enregistrée, il sert maintenant de **référence** et les phares peuvent se déplacer. Ce déplacement est le deuxième type de phase. La seule contrainte sur leur nouvelle position est que chaque phare doit avoir une ligne de vue directe avec le marqueur **référence**. Une fois les phares placés, leur pose est calculée, les marqueurs sont à nouveau suivis et le marqueur référence peut à nouveau bouger. L'enchaînement des phases de déplacement et de suivi permet de couvrir tout l'espace de la mission et de fournir un positionnement global, c'est-à-dire exprimé par rapport à la référence initiale. Il est important de noter que quand un marqueur sort de la zone de diffusion (c'est le cas du $marqueur_1$ quand les phares se déplacent dans la $phase_1$) il n'est évidemment plus possible de calculer sa pose. Toutefois, lorsque ce dernier entre à nouveau dans la zone de diffusion d'un phare (comme le $marqueur_1$ dans la $phase_2$), il est a nouveau suivi par le système.

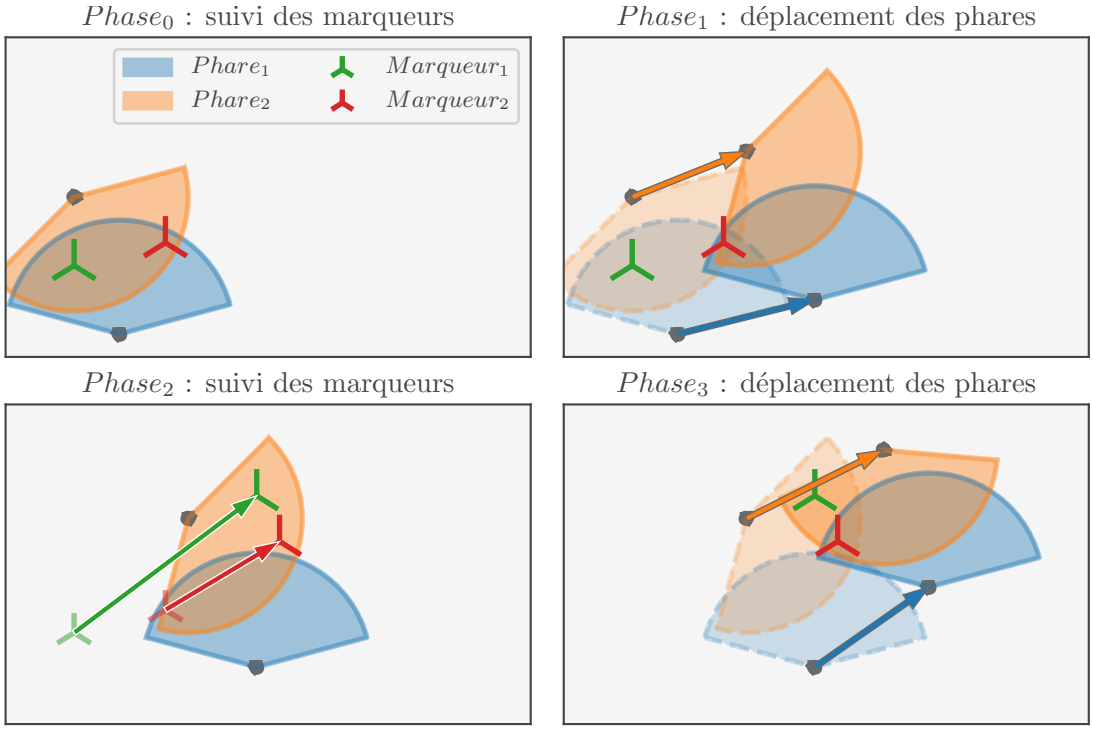

FIGURE 3.5 – Processus de construction du maillage temporel

3.2.2 Détermination de la position globale

Nous avons présenté le processus qui nous permet de déplacer le système de positionnement sans préciser la méthode de calcul qui permet de produire et de conserver une estimation globale

de positionnement. Globale ici est entendue au sens où les poses calculées sont exprimées dans le repère initial de l'expérience —qui peut par exemple être la pose initiale d'un phare—, au lieu d'être exprimées localement. Le système de positionnement commercial que nous utilisons fournit en effet un ensemble de transformations exprimées dans le repère de l'un des phares. La figure 3.6 présente ces informations locales. Nous nommerons cet ensemble de transformations un **arbre de transformations**. Nous notons ${}^{A}T_{B}$ la matrice de transformation (voir section 2.1.1) qui contient la position et l'orientation de B, exprimée dans le repère A. Autrement dit c'est la transformation qui décrit le déplacement entre la pose A et la pose B. Toutes les poses sont exprimées dans le repère local par commodité, et celui-ci correspond à l'un des phares. Ainsi, la transformation pour ce phare, mettons ${}^{L}T_{P_0}$, est égale à la matrice identité.

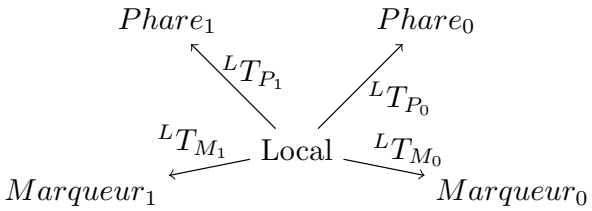

FIGURE 3.6 – Poses des éléments du système exprimées dans un repère local

Afin de pouvoir exprimer l'ensemble des poses dans le référentiel global, il faut déterminer la transformation ${}^{G}T_{L}$ contenant la position et l'orientation du repère local exprimé dans le repère global. Dans l'état initial du système, avant la première phase de déplacement des phares, les repères global et local ne font qu'un. Afin de traduire cette superposition, ${}^{G}T_{L}$ est initialisée avec la matrice identité 4×4 notée I_d. Avant de basculer vers la première phase de déplacement, un des marqueurs —le $Marqueur_i$ (M_i) par exemple— est immobilisé et sa transformation dans le repère global ${}^{G}T_{M_i} = {}^{G}T_{Ref}$ est enregistrée :

$$ {}^{G}T_{Ref} = \underbrace{{}^{G}T_{L}}_{I_d} \cdot {}^{L}T_{M_i} $$

Cette étape d'immobilisation et d'enregistrement est appelée le **verrouillage**. une fois le verrouillage du $Marqueur_1$ effectué, les phares se déplacement, et une fois en position fournissent un nouvel arbre de transformations locales. La figure 3.7 illustre les informations alors à notre disposition après ce déplacement.

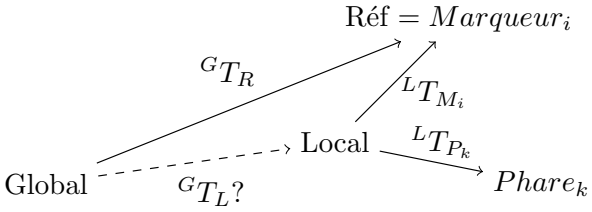

FIGURE 3.7 – Arbre de transformations après déplacement des phares

Connaissant l'identité de la référence et sachant quelle n'a pas bougé, il est alors possible de déterminer la nouvelle matrice ${}^{G}T_{L}$ dans une opération que nous nommons **déverrouillage** :

$$ {}^{G}T_{L} = {}^{G}T_{Ref} \cdot {}^{L}T_{M_i}^{-1} $$

L'arbre de transformations obtenu après l'opération de déverrouillage est présenté dans la figure 3.8. Le marqueur qui servait de référence est désormais libre de se déplacer. Obtenir la pose globale d'un élément, par exemple du $Phare_k$ revient alors à une simple multiplication de matrices :

$$^{G}T_{P_k} = ^{G}T_{L} \cdot ^{L}T_{P_k}$$

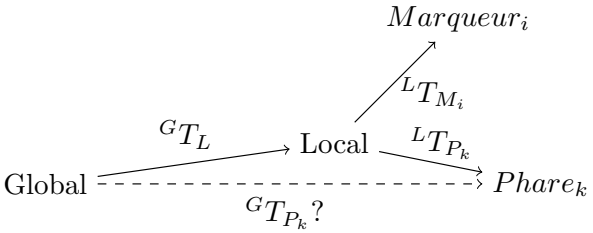

FIGURE 3.8 – Arbre de transformations après déverrouillage

C'est la répétition des étapes de **verrouillage** et **déverrouillage**, respectivement avant et après le déplacement des phares qui permet de déterminer la pose globale des éléments du système. Il est important de préciser que lors de l'étape de déverrouillage, où l'on calcule une nouvelle estimation de la transformation $^{G}T_{L}$, l'erreur faite sera définitivement intégrée. Cela conduit, de manière analogue aux estimateurs plus classiques à une dérive de l'estimation de position au fil de la mission. Toutefois, contrairement à ces approches, l'ajout d'erreur ne se fait que lors des déplacement des phares, et non plusieurs fois par seconde. Cela permet si la mission comporte une phase statique —par exemple un forage minier— de n'accumuler aucune erreur sur toute cette période. De plus, les éléments étant statiques lors de la phase de déverrouillage, il est possible de se baser sur de nombreuses mesures pour minimiser l'effet du bruit de mesure.

3.3 Implémentation proposée

La réalisation d'un prototype de système de positionnement, construit sur les bases théoriques présentées précédemment, n'a pas été chose aisée. En effet, il a fallu réaliser un nombre conséquent de tâches d'ingénierie, allant de la soudure au développement de bibliothèques complexes optimisées avec des contraintes temps réel. Nous n'aborderons pas ici l'ensemble des problématiques rencontrées. Nous nous concentrerons en effet sur la présentation de l'architecture logicielle à laquelle nous avons abouti.

3.3.1 Architecture de NAPS

Un système cyber-physique est composé nous l'avons vu d'une collection de pièces matérielles et logicielles. Avant même de commencer le développement d'un tel système, il faut déterminer comment connecter le matériel et le logiciel d'une part, mais également les différentes briques logicielles d'autre part. Face à cette problématique, une pratique commune consiste à adopter un intergiciel, plus connu sous son nom anglais de *middleware*. L'utilisation d'un tel outil permet de simplifier la décomposition des tâches, tout en standardisant les échanges, permettant une

intégration plus aisée de nouveau matériel. En robotique, c'est *Robot Operating System (ROS)*[8] qui s'est imposé au fil des années. Contrairement à ce que son nom indique, ce n'est pas un système d'exploitation, mais un intergiciel doublé d'une bibliothèque gigantesque d'utilitaires liés à la robotique.

Nous allons aborder trois concepts fondateurs de ROS :

Noeud maître Effectue la synchronisation entre les noeuds.

Noeud Les noeuds sont des processus classiques, qui se signalent au noeud maître. Il est possible de les coder en C++ ou en Python.

Topic Ce sont les canaux de communication de ROS. Ils permettent aux noeuds de s'échanger simplement des informations en écrivant où en écoutant sur des canaux spécifiques.

Si nous avons choisi d'utiliser ROS, c'est également grâce à la richesse des développements open source qui l'accompagnent. En effet, en plus de proposer les algorithmes classiques employés en robotique, cet écosystème dispose de nombreux pilotes de matériels. Ces derniers sont soit mis à disposition directement par les fabriquants de capteurs et d'actionneurs, soit par les membres de la communauté.

Trois éléments principaux composent la solution logicielle proposée :

Pilote Malheureusement aucun pilote ROS n'est disponible pour le système de suivi. C'est donc le premier élément que nous avons réalisé. Il encapsule le logiciel de *Valve* nommé *SteamVR*, et permet de configurer le système correctement. Il est également capable d'effectuer un redémarrage de *SteamVR* tout en supprimant les bases de données de ce dernier pour forcer une recalibration du matériel. Ces efforts supplémentaires ont été nécessaires car le pilote fourni est propriétaire et ne permet pas de contrôler pleinement le système physique. Nous reviendrons ultérieurement sur cette problématique.

Coeur C'est ce noeud qui gère tout le processus. Il s'assure d'effectuer toutes les opérations nécessaires lorsque le système souhaite changer de phase —lors d'un verrouillage ou d'un déverrouillage. Il enregistre donc les références, et les combine aux informations locales pour générer une nouvelle estimation de la pose du repère local. Il peut demander au pilote une recalibration du système.

IO C'est un noeud de commodité. La détection des touches d'un clavier permet de demander au coeur de déclencher les différents événements de contrôle en fonction des besoins de l'expérimentation. Il n'est pas nécessaire que cette télécommande soit exécutée sur la même machine que le coeur. Ce logiciel de commande à distance, comporte une autre fonctionnalité essentielle, il joue un son de réussite où d'échec en fonction des retours que lui fait le coeur de NAPS. Cela peut paraître superflu, mais c'est fort pratique.

La figure 3.9 décrit l'architecture de NAPS. On y retrouve les trois éléments présentés précédemment, ainsi qu'un arbre de transformations. Cet arbre est en réalité une brique de la bibliothèque standard de ROS. Il permet de gérer un arbre des transformations, et met ainsi à disposition de toutes les composantes du système les estimations de poses qui lui sont fournies. Ainsi, les informations calculées par notre système sont mises à disposition de manière standard.

8. Site de ROS https://www.ros.org/

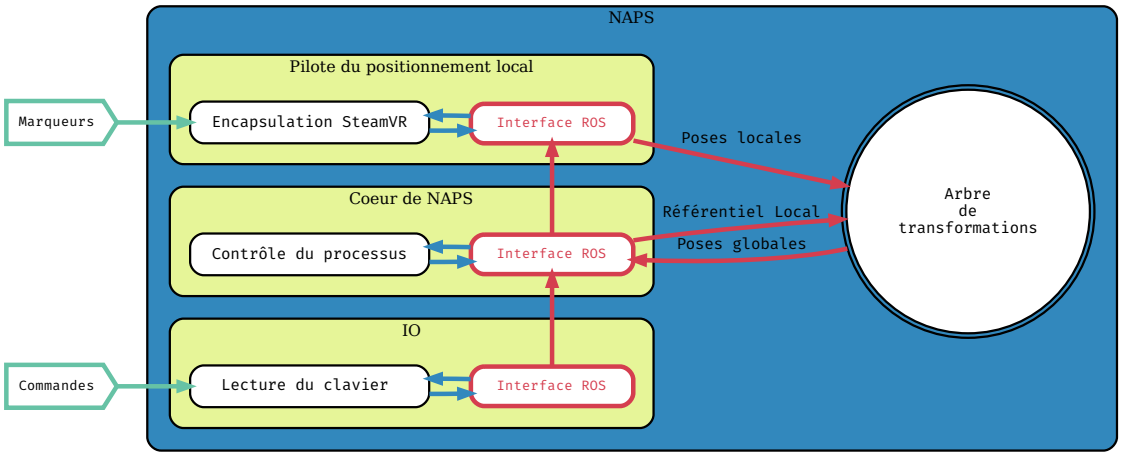

FIGURE 3.9 – Architecture générale de la solution logicielle proposée

Il est par conséquent facile par la suite d'exploiter les transformations produites, par exemple en les affichant avec l'outil dédié de ROS : *rviz*.

Afin de limiter la dépendance à ROS, chacun des éléments est décomposé en deux parties : l'une comprenant la logique, et l'autre servant d'interface entre cette logique et les systèmes de communication de ROS. En plus d'apporter de la clarté dans le code, cela permet d'utiliser ces briques en dehors de ROS.

C'est toujours dans l'objectif de limiter la dépendance —mais au système physique cette fois-ci— que la décomposition en trois éléments à été choisie. En effet, pour peu qu'un autre capteur soit livré avec un pilote ROS, il sera directement utilisable dans NAPS. Enfin, le noeud télécommande permet d'envoyer des événements de contrôle exactement de la même manière que le ferait un robot. Ainsi, lorsque le système sera complètement embarqué, l'élément IO sera naturellement remplacé par un noeud décisionnel robotique, et sa connexion à NAPS sera totalement transparente.

3.3.2 Architecture de validation

Cette courte section présente l'architecture qui a servi à réaliser la validation du prototype proposé. La figure 3.10 permet de comprendre comment les données ont été acquises. On y retrouve l'architecture de NAPS présentée précédemment, ainsi que des nouveaux éléments, représentant les outils nécessaires à la production d'un jeu de donnée comportant une vérité terrain. Plus de détails serons donnés sur le matériel ayant permis de mesurer cette vérité terrain, nous nous concentrons ici sur la partie logicielle. Cette dernière comporte un client *Virtual-Reality Peripheral Network (VRPN)*[9] qui traduit le protocole VRPN diffusé par le système matériel en messages ROS qui sont ensuite communiqués aux noeuds ROS qui le souhaitent. Nous ne faisons

9. Dépôt git du groupe VRPN https://github.com/vrpn/vrpn/wiki

qu'employer ce traducteur, il est disponible dans la bibliothèque ROS [10].

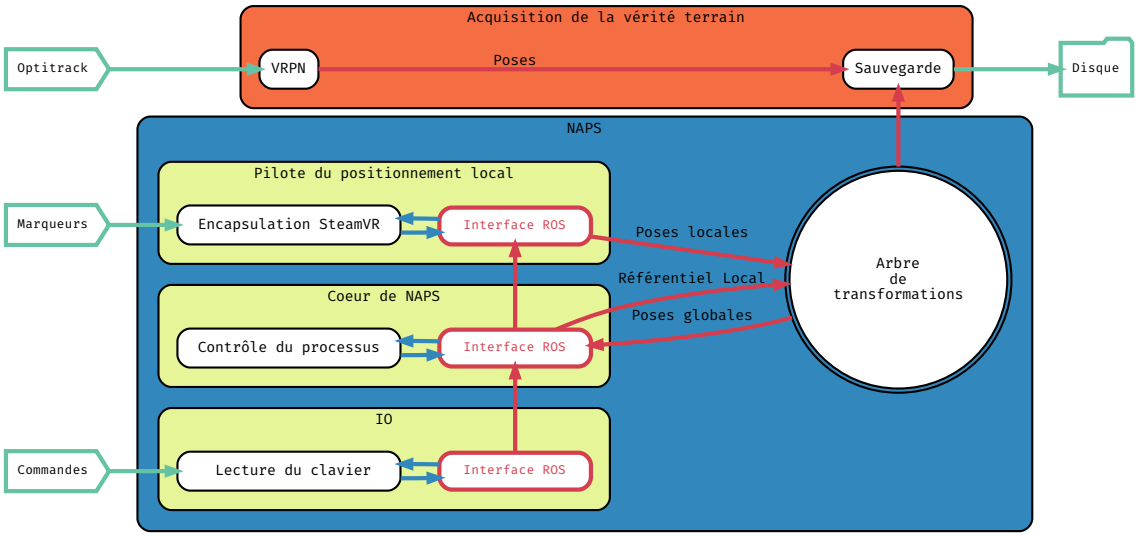

FIGURE 3.10 – Ajout du groupe d'acquisition de la vérité terrain

Nous avons ensuite crée un noeud ROS de **Sauvegarde** qui écoute les diffusions du traducteur d'un côté —pour obtenir la trajectoire vérité terrain— et demande de l'autre côté à l'arbre de transformation les poses formant la trajectoire produite par NAPS. Ces deux acquisitions étant simultanées, nous procédons à l'horodatage des données puis nous les écrivons sur le disque. Cette procédure permettra d'évaluer précisément notre système de positionnement, mais nous y reviendrons bientôt.

3.3.3 Un développement difficile

Cette section est une parenthèse me permettant d'évoquer la difficulté globale rencontrée autant lors de la phase initiale d'évaluation des approches et de choix de direction que lors de la phase d'élaboration de notre prototype.

Premièrement, la recherche en robotique demande de pouvoir disposer de divers capteurs et actionneurs afin d'évaluer les différentes approches de la littérature et déterminer quelle voie suivre. L'idéal serait d'avoir un ensemble de capteurs et de robots prêts à l'emploi, accompagné des connaissances nécessaires à leur mise en œuvre. En l'absence d'un tel écosystème favorable, chaque nouvelle évaluation d'une piste nous a demandé la prise en main complète des capteurs impliqués, en partant de zéro et ce sans aide. Bien que très enrichissantes, ces tâches s'éloignent du travail de fond de la thèse et grignotent toujours plus le temps déjà limité disponible. De plus, la conception d'un système robotique est un challenge qui demande de nombreuses compétences : la mécanique, l'électronique, l'algorithmique, et bien d'autres. Les nombreux soucis matériels ainsi que les soucis liés comme la mise à jour inopinée des pilotes… demandent d'apporter un

10. Client VRPN ROS http://wiki.ros.org/vrpn_client_ros

soin constant au système. La conception et la maintenance de systèmes robotiques nécessite un ingénieur spécialisé, employé à temps plein. Durant cette thèse, nous ne disposions pas de ce soutien, il a donc été nécessaire que je réalise, en mobilisant mes compétences limitées dans ces vastes domaines, l'ensemble de ces tâches.

De plus, la lourdeur des démarches administratives requises pour l'acquisition de matériel à représenté un réel frein dans la thèse, autant dans la phase initiale qui demandait l'évaluation rapide de différentes techniques que dans la phase de développement de notre solution. Ces freins se sont incarnés à la fois dans la quantité importante de travail administratif à fournir, que dans la lenteur du processus de commandes. Ce phénomène a été grandement renforcé par la pandémie, où dans le cas le plus extrême, nous avons attendu presque un an pour recevoir du matériel nécessaire à l'avancée de notre recherche. Il est bien évidemment toujours possible d'explorer d'autres pistes pour minimiser le temps perdu, mais l'environnement dans lequel j'ai évolué est largement sous-optimal pour répondre au type de problématiques soulevées par le sujet de thèse.

Une autre problématique à laquelle nous avons été confrontés est la nécessité d'**optimiser** le logiciel produit. En effet, à contrario de la simulation où le temps est manipulable, les expériences dans le monde réel nécessite le respect de cette contrainte temporelle. un soin particulier a donc été porté aux algorithmes utilisés et ce à tous les niveaux. Le mécanisme principal permettant l'optimisation du code consiste à **vectoriser** les calculs pour tirer pleinement parti des capacités des machines actuelles. J'ai pour cela suivi un guide complet [11], comportant de nombreux exemples pratiques. Nous avons abordé d'autre approches comme la compilation juste-à-temps, ou l'utilisation du GPU, mais la vectorisation des calculs est restée la solution de prédilection car plus portable et nécessitant moins d'adaptations du code. Ici aussi, ces problématiques ont été très formatrices, et j'ai eu l'occasion d'en apprendre d'avantage sur les méthodes de calcul et sur les représentations mémoire. Toutefois, ces travaux ont également consommé une quantité non négligeable de temps.

Un autre écueil concerne les complications induites par l'utilisation de matériel et surtout de logiciel propriétaire. En effet, le pilote (*SteamVR*) que nous avons utilisé —car il n'y avait pas d'alternative— n'est pas documenté, et pas spécialement conçu pour la récupération des informations dont nous avons besoin (ces informations sont normalement récupérées par des jeux via des moteurs, à haut niveau). Nous avons donc du déployer un trésor d'inventivité pour pouvoir d'une part récupérer ces informations, et d'autre part contrôler le système pour le modeler conformément à nos besoins. Dans un premier temps, il a fallu procéder par essai et erreur pour configurer le système pour qu'il fonctionne uniquement avec les marqueurs (sans le casque de réalité virtuelle) puis pour récupérer les informations de positionnement et les identifiants des différents éléments. Un bon exemple représentant le problème de contrôle du système est la procédure que nous utilisons pour forcer la recalibration. Nous mettons tout simplement fin au processus du driver, puis nous supprimons les bases de données de ce dernier, afin de s'assurer qu'il n'utilise pas les anciennes données qui ne correspondent plus à la réalité. Cette procédure est loin d'être optimale. En effet, cette fonction de recalibration existe nécessairement dans le pilote propriétaire, et dans un système ouvert, un appel de fonction aurait suffit à régler le problème… Cet aspect, contrairement aux autres difficultés évoquées précédemment n'a pas été particulièrement formateur, si ce n'est dans la conclusion tirée de cette expérience : **ne jamais utiliser de systèmes fermés** ! Une alternative libre [12] existe, mais jusqu'à de très récents déve-

11. Python to Numpy https://www.labri.fr/perso/nrougier/from-python-to-numpy/
12. Libsurvive https://github.com/cntools/libsurvive

loppements, la qualité des données fournies était loin de celle de l'implémentation propriétaire. Nous avons donc développé un pilote alternatif se basant sur cette version ouverte, en prévision du jour où leurs estimations s'affineront.

Ces critiques peuvent parfois paraître acerbes, mais mon intention est plutôt de l'ordre du retour d'expérience constructif, muni du recul acquis au long de cet exercice qu'est la thèse. Ce sont ces difficultés qui, bien que chronophages, m'ont permis d'acquérir de nombreuses connaissances et compétences, parfois poussées. De plus, ces analyses m'ont permis d'améliorer certains de ces aspects. Revenons sur la nécessité d'un écosystème favorable composé de matériels et de connaissances pour la bonne conduite d'une recherche en robotique. C'est dans ce cadre que j'ai crée un groupe GitLab pour l'équipe Simbiot, permettant de concentrer et de pérenniser les expériences acquises. Toujours dans la même optique, j'ai effectué un effort de mise à disposition et de documentation complète des pilotes, des algorithmes et des recherches que j'ai eu l'occasion d'effectuer.

3.4 Expérimentations

La particularité de NAPS, à savoir l'utilisation de capteurs autres que les caméras et les IMU, a empêché l'utilisation de jeux de données communément utilisés par la communauté. Par conséquent, nous avons conçu et collecté nos propres jeux de données, en gardant à l'esprit que l'acquisition précise de la vérité terrain est un facteur déterminant pour la validation de l'approche. Dans ce contexte, nous avons utilisé un système de capture de mouvement composé de 8 **Optitrack Prime 17w** caméras infrarouges formant un système de capture de mouvement —utilisé principalement dans le cinéma et l'animation 3D— fournissant un suivi 6DOF à 250 Hz dans un volume de capture de $5 \times 4 \times 3$ mètres avec une précision inférieure au millimètre pour l'estimation de la pose (d'après les données de calibration du système de capture de mouvement). Une autre contrainte a été le nombre limité de dispositifs à notre disposition (2 phares et 3 marqueurs). Il est utile de mentionner que le système de capture de mouvement illumine toute la zone avec une lumière infrarouge. Cela peut perturber le système de suivi **VIVE** qui est basé également sur la communication infrarouge entre les appareils. Cependant, bien que cette configuration puisse désavantager quelque peu le système NAPS en réduisant sa précision, des tests préliminaires ont montré que de telles perturbations n'étaient pas assez importantes pour invalider les résultats obtenus lors de ces expériences.

3.4.1 Protocoles expérimentaux

Pour démontrer la validité de notre approche, deux protocoles expérimentaux distincts ont été conçus. Tous deux utilisent notre processus de déplacement en deux phases, décrit précédemment, alternant entre le suivi du ou des explorateurs et le repositionnement des phares. De plus, ils sont tous deux conçus en fonction de la zone limitée d'acquisition de la vérité terrain (Optitrack) à notre disposition (couvrant $\approx 20m^2$ alors qu'un phare couvre $\approx 30m^2$). En raison de cette limitation, presque tous les phares ont été localisées en dehors de la zone couverte par le système de capture de vérité terrain, nous empêchant d'obtenir cette information pour tous les appareils. Cela explique pourquoi nous n'évaluons qu'une seule trajectoire (qui pourrait être celle de l'explorateur). En effet, la comparaison des seules estimations de la position de l'explorateur est suffisante pour valider la précision de notre système, car toute erreur sur la position

des phares est directement répercutée sur la trajectoire de l'explorateur.

Comme notre implémentation de NAPS utilise *Robot Operating System (ROS)* [8], les estimations de pose sont calculées et publiées directement sur les *topics* —les canaux de communication de ROS— pour tous les phares et tous les explorateurs à un rythme de 250 Hz. En outre, un autre nœud ROS écrit dans des fichiers la vérité terrain et les estimations de pose immédiatement après leur publication. Afin de limiter le bruit de mesure, après chaque déplacement des phares, le calcul de leur nouvelle position se base sur de multiples observations. Cette intégration temporelle dure 1s, et compile 250 mesures pour obtenir une estimation plus fiable.

1. Mouvements aléatoires (*random moves*) : L'objectif du premier protocole est de couvrir les phases de transition d'un processus d'exploration, à savoir le repositionnement — translations et rotations— du système de positionnement. Dans cette première phase, l'explorateur se déplace librement dans la zone formée par les phares afin de remplir sa mission, puis il se pose n'importe où dans cette zone. Dans la deuxième phase, tous les phares sont réarrangés de manière aléatoire dans la même zone. Il est important de noter que tous les phares sont déplacés, ce qui ajoute effectivement des erreurs d'estimation au système.

   Dans le contexte de cette expérience, l'explorateur est utilisé comme référence et toutes les balises sont déplacées pour former une nouvelle zone de suivi. Cependant, l'utilisation de l'explorateur comme référence n'est qu'un choix pratique mais il n'induit pas de perte de généralité de l'approche puisque tous les éléments du système sont suivis avec la même précision.
   Les résultats de cette expérience sont étiquetés **RMi**. Pour chaque expérience, la concaténation brute (sans post-traitement) des étapes consécutives (phases 1 et 2) sans aucun post-traitement fournit les trajectoires horodatées d'un élément (l'explorateur) correspondant respectivement aux estimations produites par NAPS et à la vérité terrain.

2. Mouvements dirigés (*guided moves*) : Ce deuxième protocole est conçu pour reproduire un processus complet d'exploration guidée, c'est-à-dire le déplacement du système dans une direction spécifique. Le principe est le même, sauf qu'au lieu de se déplacer de manière aléatoire, les balises sont déplacées vers une zone voisine, dans une direction donnée. Afin de préserver la précision du système lorsque les balises se déplacent vers une autre zone, l'explorateur doit atterrir près de la frontière de la zone actuelle de suivi qui est commune à la zone visée. Ensuite, la deuxième phase —repositionnement des phares— peut commencer.

   Ce processus itératif est répété plusieurs fois afin d'évaluer l'accumulation d'erreurs sur une longueur de chemin réaliste (plusieurs dizaines de mètres) nécessitant plusieurs zones successives de suivi. Ces expériences sont nommées **GMi** par la suite.

   Les jeux de données produits et leur caractéristiques sont présentés dans la table 3.1. Les jeux sont composés de 4 expérimentations de mouvements aléatoires (RM) et de 4 expérimentations de mouvements guidés (GM). De plus, un test à grande vitesse (*High speed test* HST) est ajouté, dans lequel la robustesse de NAPS aux mouvements rapides de l'explorateur est évaluée. Les différences entre le nombre de poses de la vérité terrain

et le nombre de poses de NAPS proviennent de fréquences d'acquisition différentes, inhérentes aux différents systèmes. Le nombre de déplacements dans la table correspond aux nombre de fois où les phares ont été repositionnés lors de l'expérience.

TABLE 3.1 – Jeux de données collectés durant les expérimentations

	Longueur (m)	NAPS Déplacements	Vérité terrain poses	NAPS poses
RM1	40	2	45k	22k
RM2	50	3	46k	28k
RM3	53	2	27k	17k
RM4	52	2	27k	17k
GM1	51	1	26k	16k
GM2	74	4	65k	32k
GM3	40	2	33k	8k
GM4	176	9	125k	50k
HST	117	2	22k	10k

3.5 Résultats

Afin de pouvoir comparer nos résultats avec l'état de l'art de l'odométrie (visuelle-inertielle), nous devons utiliser les mêmes métriques. Ce que nous devons quantifier est l'erreur —la différence— entre la trajectoire réelle et la trajectoire estimée par NAPS. Nous avons pour cela utilisé la bibliothèque logicielle open Toolbox for trajectory evaluation [Zhang and Scaramuzza, 2018]. Cette bibliothèque, fournie par l'université de Zurich et l'ETH Zurich [13], permet à la fois de calculer les erreurs quadratiques moyennes (RMSE) le long des trajectoires complètes, ainsi que de produire des analyses de type KITTI [Geiger et al., 2012]. Ces analyses consistent à découper les trajectoires en segments de taille régulière, à aligner les segments correspondants de l'estimation et de la vérité terrain, puis à calculer l'erreur d'estimation sur le dernier état —la dernière pose— de ce segment.

3.5.1 Analyse qualitative

Pour proposer un premier aperçu de la précision, les trajectoires superposées mesurées par le système de capture de vérité terrain et NAPS sont données dans les figures 3.11 et 3.12, respectivement pour une expérience de déplacement aléatoire et pour une expérience de déplacement guidé. Comme les trajectoires sont initialement exprimées dans des systèmes de coordonnées différents (Optitrack et NAPS), la toolbox d'alignement (type sim3) a été utilisée pour obtenir les chevauchements. La position d'arrivée de l'élément suivi pouvant être différente de celle de départ et le sens de parcours n'ayant pas d'influence sur le positionnement, il n'est pas utile de distinguer ces points sur la figure. Comme on peut le constater, la trajectoire estimée par NAPS est fidèle à la trajectoire vérité terrain mesurée par le coûteux système de capture de mouvements.

13. Robotics and Perception Group http://rpg.ifi.uzh.ch

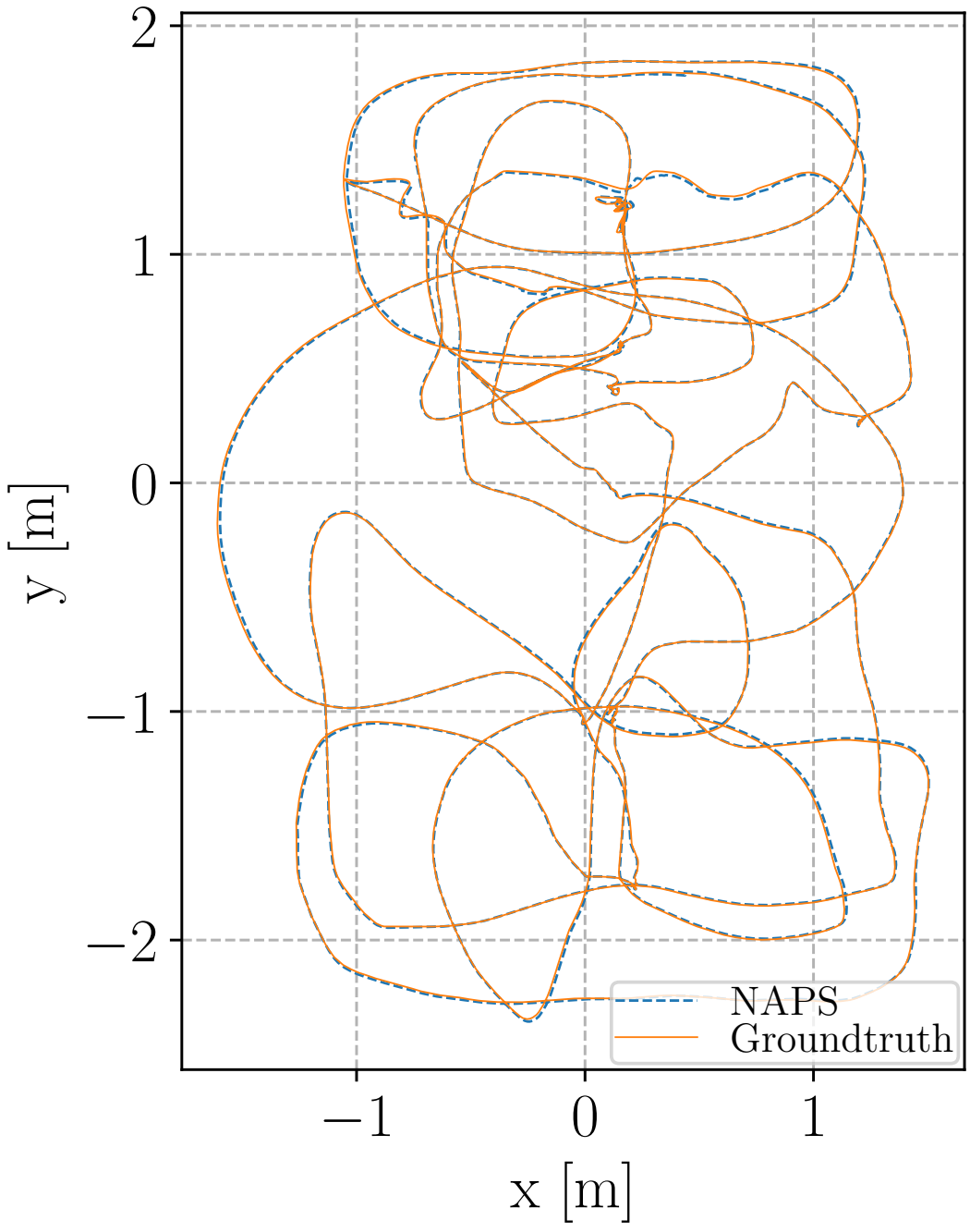

FIGURE 3.11 – Vue du dessus de la trajectoire GM2

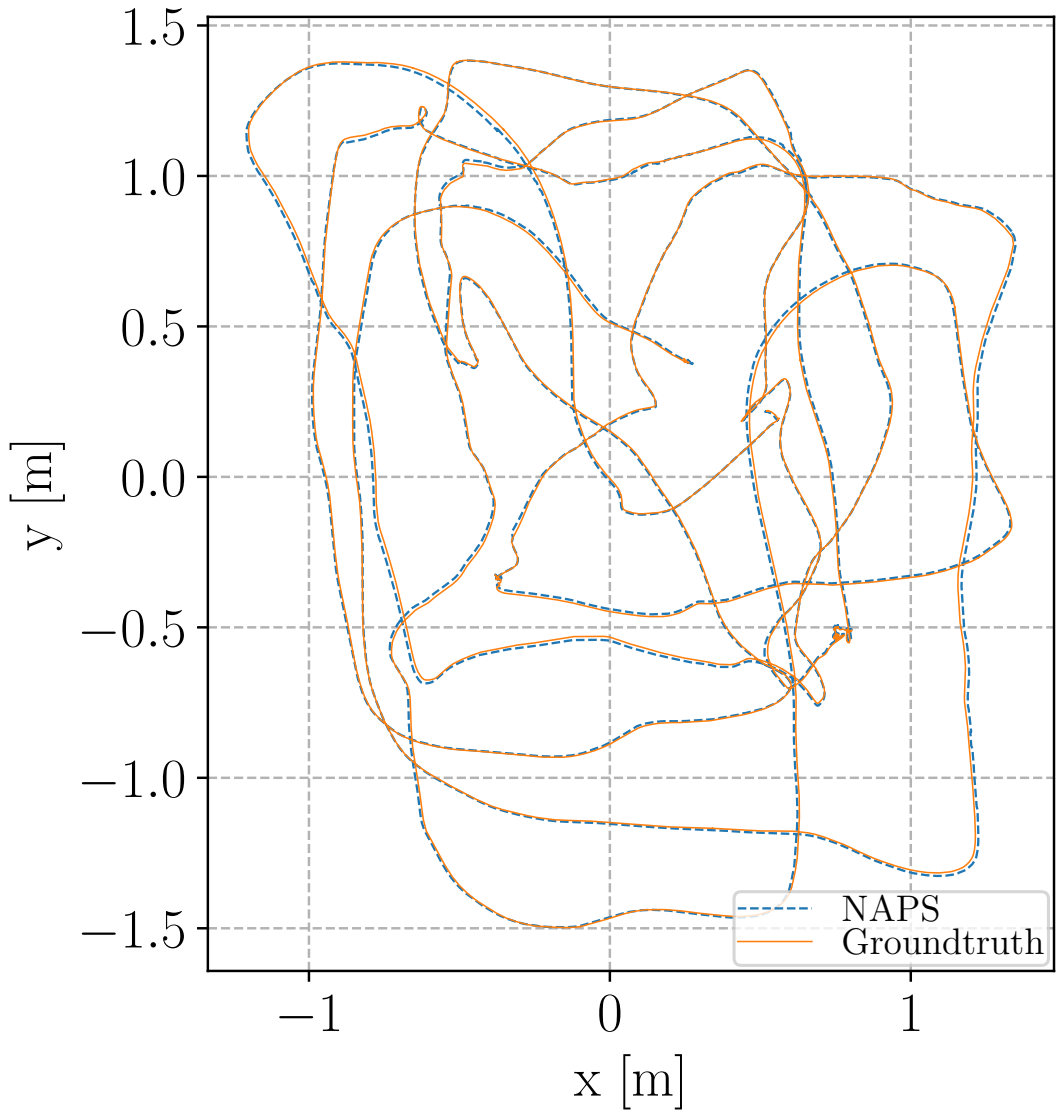

FIGURE 3.12 – Vue du dessus de la trajectoire RM2

Afin d'avoir une idée plus fine de la précision de notre système, les erreurs de translation et d'orientation des estimations de la pose NAPS dans l'expérience GM2 sont représentées sur les figures 3.13 et 3.14, pour toutes les dimensions ou axes de rotation. Les autres expériences obtiennent des tracés similaires, mais dans l'expérience GM2, un pic d'erreur de translation est visible à l'étape 0 pour une valeur de x. Toutefois, cela correspond à une perte de ligne de vue temporaire entre l'élément suivi et les phares au cours du test manuel. Cependant, ce problème peut être évité en utilisant des algorithmes assurant la ligne de vue, que nous développerons par la suite dans la section 6. Toutes les autres mesures sont cohérentes et restent dans un intervalle d'erreur [-40 :30] mm. Il est important de constater ici que la dérive de notre système est limitée, l'augmentation de l'erreur en fonction de la distance parcourue est bien inférieure aux autres systèmes de positionnement.

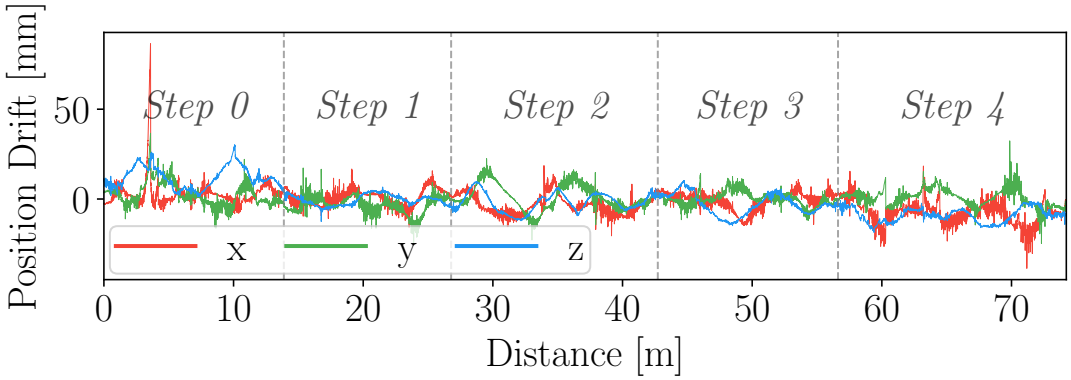

FIGURE 3.13 – Erreur de translation

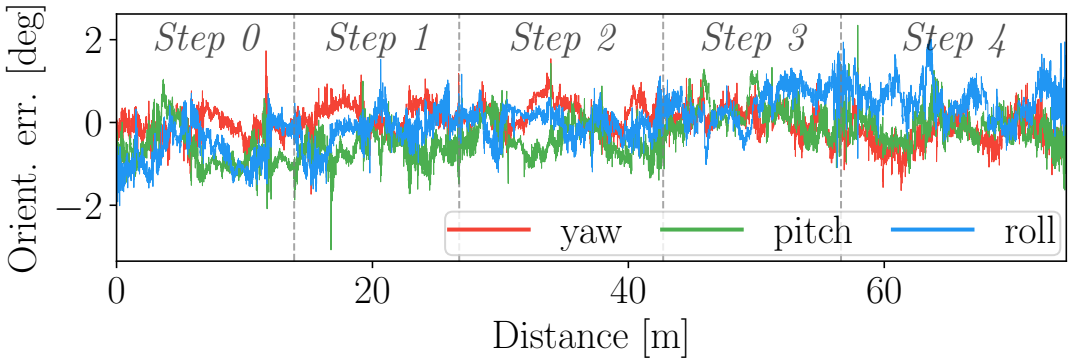

FIGURE 3.14 – Erreur de rotation

3.5.2 Analyse quantitative

La métrique de l'erreur quadratique moyenne (*Root Mean Square Error ou RMSE*) est souvent utilisée pour l'analyse des erreurs de trajectoire. Elle offre l'avantage de résumer les erreurs sur l'ensemble de la trajectoire avec une seule valeur. Dans le tableau 3.2 sont présentées les RMSE obtenues lors de nos expériences. Comme précédemment mentionné, une comparaison

directe de nos résultats avec ceux obtenus par d'autres algorithmes est assez délicate car les jeux de données sont différents. Cependant, les résultats présentés dans plusieurs benchmarks d'odométrie [Schubert et al., 2018, Delmerico and Scaramuzza, 2018] suggèrent un ordre de grandeur plus élevé de la métrique RMSE. Par exemple, les meilleurs résultats rapportés dans [Delmerico and Scaramuzza, 2018] sont de $0,03\,m$ pour un ensemble de données spécifique —*MH_02_easy* $73.5m$, EuRoC Dataset [Burri et al., 2016]— alors que c'est la valeur RMSE maximale observée pour NAPS, avec la trajectoire la plus longue $(176\,m)$.

TABLE 3.2 – RMSE de translations et d'orientation de l'estimation fournie par NAPS

	RM1	RM2	RM3	RM4	GM1	GM2	GM3	GM4	HST
Trans. (m)	0.011	0.010	0.016	0.015	0.011	0.011	0.010	0.031	0.014
Ori. (deg)	1.288	0.936	1.090	0.894	0.973	0.939	1.492	1.669	1.824

Une autre approche courante pour l'analyse de trajectoire est l'analyse dite *KITTI* [Zhang and Scaramuzza, 2018]. Son principe est de diviser la trajectoire entière en sections régulières à longueur fixe. Chaque couple de sections composés des sections de la trajectoire estimée et de la vérité terrain correspondante est aligné, puis l'erreur d'estimation de l'état final —la différence entre la dernière pose estimée et la dernière pose vérité terrain— est calculé pour chaque couple de sections le long de la trajectoire complète. Ce processus est répété en choisissant différentes longueurs de découpe. Cette technique a été utilisée pour produire les figures 3.15 et 3.16, dans lesquelles les distributions d'erreurs relatives en translation et en lacet sont représentées pour les différentes longueurs de découpe. Concernant les angles, seul le lacet est considéré car c'est la seule rotation dont il n'est pas possible de corriger la dérive à partir de la force de gravité mesurée par une centrale inertielle. Cette analyse a été effectuée à partir de l'ensemble des trajectoires obtenues dans toutes nos expérimentations. On peut voir que les erreurs médianes de NAPS sont de $29\,mm$ et de $0,5\,°$ pour une longueur de section de $35\,m$. À titre de comparaison, les meilleurs résultats rapportés dans la littérature [Delmerico and Scaramuzza, 2018] sont obtenus par l'approche *VINS-MONO* [Qin et al., 2017], avec une erreur de translation médiane de $150\,mm$ pour la même longueur de chemin. Bien que les jeux de données ne sont pas les mêmes, la diversité, la complexité et la généralité de nos expériences les rendent représentatifs de cas d'utilisation généraux.

Ainsi, nous pouvons conclure que, compte tenu de la dépendance limitée de notre prototype NAPS à l'égard de l'environnement, de la robustesse des capteurs employés et son taux de mise à jour de $250Hz$ permettant l'intégration temporelle —et donc la minimisation des erreurs dues au bruit de mesure—, des résultats similaires devraient être obtenus dans la plupart des contextes réels.

3.6 Conclusion

Nous avons conçu et présenté un système de positionnement nomade et précis (NAPS). Son originalité réside dans le couplage entre techniques employées par les géomètres, et la levée du problème d'association de données en s'inspirant du modèle des phares maritimes. L'emploi d'un ensemble d'agents mobiles actifs permet de générer un maillage de positionnement dans l'environnement, qui s'affranchi de toute dépendance à l'environnement grâce à l'identification des

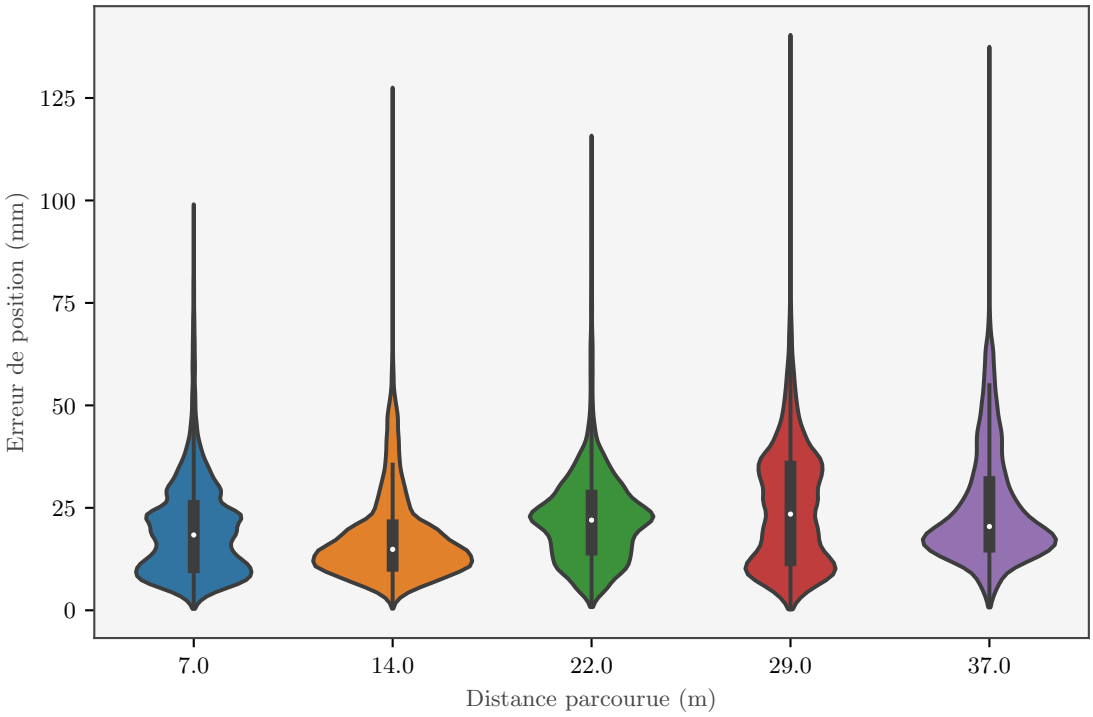

FIGURE 3.15 – Distribution des erreurs en translation par longueur de segment

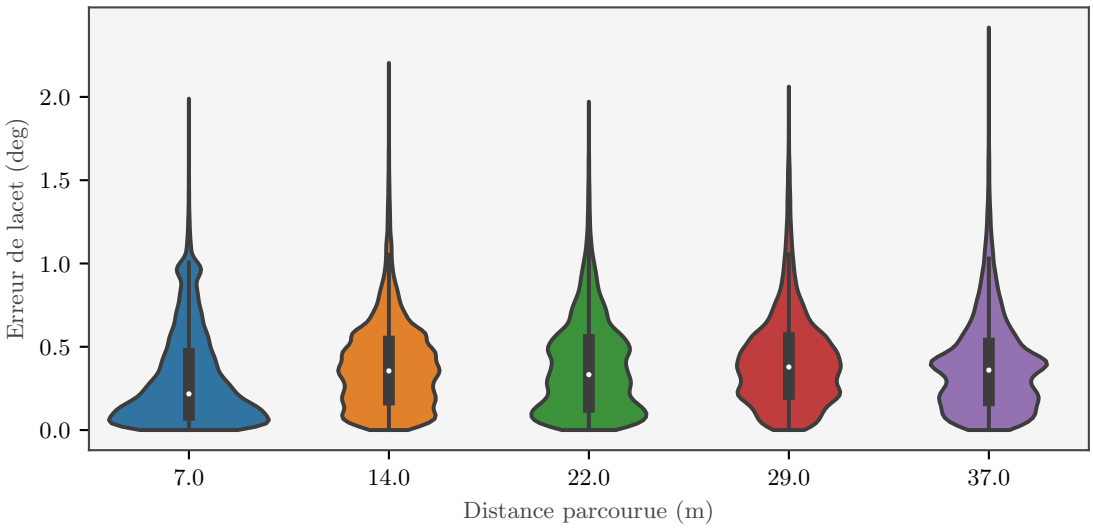

FIGURE 3.16 – Distribution des erreurs en rotation par longueur de segment

pairs.

L'intérêt et la validité du système ont été démontrés de manière pratique par une série d'expériences, montrant la très bonne précision de l'algorithme de positionnement. En effet, les erreurs de positionnement de NAPS restent de l'ordre du centimètre sur des trajets assez longs (<45 mm pour 175 m), ce qui représente une erreur bien moindre que celle induite par les solutions existantes. La qualité des estimations obtenues est d'autant plus impressionnante que notre système n'a pas recours aux classiques et lourds systèmes *back-end*, tous employés par les solutions auxquelles nous nous comparons.

Bien que les résultats de NAPS soient très prometteurs, plusieurs pistes de travail peuvent être identifiées. En particulier, nos expériences ont mis en évidence que la façon dont les balises sont déplacées peut avoir un impact sur la précision globale. Ainsi, nous étudierons dans le chapitre 6 différentes stratégies de déplacement des balises afin de préserver la précision tout en garantissant la contrainte de visibilité.

L'amélioration du prototype matériel est nécessaire afin de pérenniser la solution proposée. En effet, le matériel (HTC VIVE) que nous utilisons à l'heure actuelle comporte deux limitations majeures : sa portée est limitée, et son pilote propriétaire —nous empêchant un contrôle total du matériel par le logiciel. La première contrainte a été soulevée par des industriels interrogés lors de la phase de transfert de technologie entamée lors de la dernière année de thèse. L'augmentation de la portée permettrait de multiplier les cas d'applications possibles de NAPS. Cette limitation est purement matérielle, la suite logicielle produite étant totalement indépendante du capteur qui fournit les informations de positionnement locales. Une discussion est en cours avec des fabriquants de systèmes métrologiques, susceptibles de développer des capteurs spécifiques, ce toujours dans le cadre de la valorisation industrielle des développements effectués dans le cadre de la thèse.

Nous avons donc présenté dans ce chapitre notre contribution au problème de positionnement, à travers des idées nouvelles conduisant à la réalisation et à l'analyse d'un système de positionnement précis et fonctionnel. Nous verrons dans le prochain chapitre l'utilisation de ce système couplé à un lidar, réalisant ainsi un système complet de SLAM.

Chapitre 4

Un cas d'usage : la cartographie 3D

4.1 Vers une solution de SLAM complète

Nous avons précédemment démontré la pertinence du système de positionnement proposé. Notre système multi-agents est capable de se positionner précisément, et ce dans tous les environnements sans nécessiter d'adaptations particulières. Toutefois, il reste aveugle en l'état, et n'a pas de moyens de percevoir ses alentours.

Comme nous l'avons vu dans la présentation de notre contexte particulier dans la section 1.1 du chapitre d'introduction, les caméras ne sont pas une option. En effet, les mines sont potentiellement privées de toute lumière extérieure et ne sont pas nécessairement équipées en éclairages. Le choix du capteur permettant de percevoir l'environnement s'est donc naturellement porté vers le lidar.

4.2 Lidar, nuage, et recalage

Nous avons abordé le fonctionnement du lidar dans la section 2.3.3 de l'état de l'art. Rappelons simplement que dans l'état actuel des choses, il s'agit d'un télémètre laser rotatif permettant d'effectuer la mesure de distances à des angles connus.

4.2.1 D'un ensemble de nuages vers un modèle

Bien qu'initialement exprimés en coordonnées polaires par rapport au centre optique du lidar, ces coordonnées sont généralement converties en coordonnées cartésiennes pour obtenir des points $[x, y, z]$. Certains systèmes d'acquisition fournissent des informations supplémentaires comme l'intensité du signal laser réfléchi comme c'est le cas pour le laser que nous utilisons (Velodyne VLP16). Un nuage est donc une collection de points 3D, exprimés dans le repère du lidar. Notre objectif consiste à exprimer ces points dans le repère global. Ainsi, la simple concaténation successive des nuages de points exprimés dans le repère global formera un modèle de l'environnement.

4.2.2 Opération de recalage

Le recalage d'un nuage consiste à exprimer ses points dans le repère global. Pour cela, nous allons utiliser l'estimation de pose produite par NAPS. Pour cela, il faut dans un premier temps équiper le lidar d'un marqueur pour pouvoir le suivre.



FIGURE 4.1 – Équipement du lidar avec un marqueur

La figure 4.1 représente l'association réalisée entre le lidar *Velodyne VLP16* et le marqueur [14]. Le choix du point de fixation est fait pour maximiser la visibilité du marqueur tout en évitant toute obstruction des lasers du lidar.

Cette association nous permet d'obtenir le suivi du marqueur. Or, nous avons besoin de la pose du centre optique du lidar pour effectuer le recalage des points. La figure 4.2 illustre l'arbre de transformations permettant de déterminer la pose du lidar. Ici, toutes les transformations sont estimées par NAPS, à l'exception de la transformation ${}^{M}T_{Lidar}$ entre le marqueur et le lidar qui est fixe. Cette transformation est donc mesurée avec les instruments à notre disposition —un pied à coulisse— avec une précision limitée. Cette erreur humaine de mesure sera malheureusement répercutée sur la qualité du recalage produit. Il serait possible de corriger cette transformation si nous disposions d'un environnement contrôlé aux modèles 3D précisément connus. Toutefois, une telle calibration demande de nombreux efforts, alors que la réalisation d'un joint de fixation extrêmement précis est un processus largement maîtrisé dans l'industrie. Cette limitation, bien que dégradant les résultats obtenus, ne représente pas une problématique de recherche, nous avons donc choisi de ne pas nous y atteler, en acceptant la réduction de la précision du recalage induit. En effet, si les résultats obtenus sont satisfaisants, nous savons que l'intégration mécanique précise de ces deux équipements permettra d'améliorer la qualité du modèle reconstruit.

Le calcul de la pose du lidar, la transformation ${}^{G}T_{Lidar}$, revient à une simple chaîne de multiplication de matrices :

$$ {}^{G}T_{Lidar} = {}^{G}T_{L} \cdot {}^{L}T_{M} \cdot {}^{M}T_{Lidar} $$

Maintenant que nous savons déterminer la pose du lidar dans l'environnement, intéressons

14. L'image du marqueur est issue du modèle 3D réalisé par Qtit, et la photographie du lidar est tirée du kit médiatique proposé sur le site officiel du constructeur Velodyne

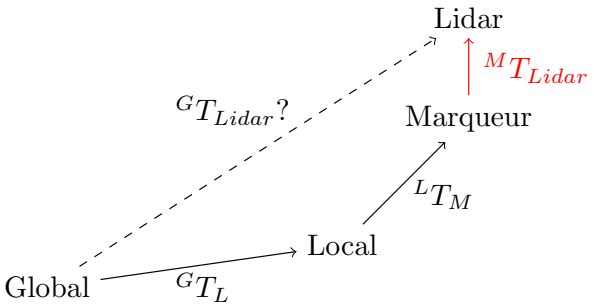

FIGURE 4.2 – Arbre de transformations permettant de déterminer la pose du lidar

nous à l'opération de recalage à proprement parler. Cette opération permet, à partir de la pose du lidar lors de la capture d'un nuage de points P, de produire P', le même nuage, mais exprimé dans le repère global. La pose du lidar est une matrice de transformation homogène. L'application de cette matrice à chacun des points permet ainsi d'exprimer les points dans le repère global. Chacun des points est également transformé en coordonnées homogènes par l'ajout d'une quatrième dimension de valeur 1, tel que $p = \begin{bmatrix} x & y & z & 1 \end{bmatrix}$. L'opération de recalage est alors la suivante :

$$P' = \left\{ {}^{G}T_{Lidar} \cdot p \quad | \, \forall p \in P \right\}$$

Appliquer cette matrice sur chacun des points individuellement est toutefois une opération très coûteuse, d'autant plus que le lidar que nous utilisons produit 300k points par seconde. Vectoriser le calcul via la bibliothèque *Numpy* permet d'obtenir un gain phénoménal en temps de calcul. Il suffit pour cela de former une matrice $P = 4 \times n$ (avec n le nombre de points) et d'appliquer la transformation à cette matrice nouvellement créée : $P' = {}^{G}T_{Lidar} \cdot P$. Cette optimisation permet de profiter au maximum des impressionnantes capacités de calcul vectoriel de nos machines modernes, et par la même occasion de réaliser cette opération en temps réel, dans une enveloppe de puissance de calcul limitée.

4.3 Architecture du système de cartographie

Nous avons abordé les détails matériels et théoriques nécessaires à la compréhension de notre système de cartographie, abordons maintenant les détails logiciels. La figure 4.3 présente l'architecture de NAPS, maintenant dotée du module nécessaire à la cartographie. Ce module est composé de quatre éléments :

Pilote du lidar Cet élément, est fourni [15] par le constructeur *Velodyne* du lidar que nous utilisons pour nos expériences de validation. C'est un pilote ROS, qui transforme les messages envoyés par le lidar —sur le câble Ethernet qui permet sa connexion au système— en messages ROS diffusés sur un canal de communication dédié. Il est important de préciser que les messages ROS transmis sont standards, nous permettant ainsi de substituer ce matériel de manière transparente par un autre lidar dont le pilote ROS est disponible.

Interface ROS Cette catégorie regroupe tous les mécanismes de traduction et de connexion ROS nécessaires au couplage de la logique du module de cartographie en pur Python et

15. Dépôt git du pilote Velodyne https://github.com/ros-drivers/velodyne

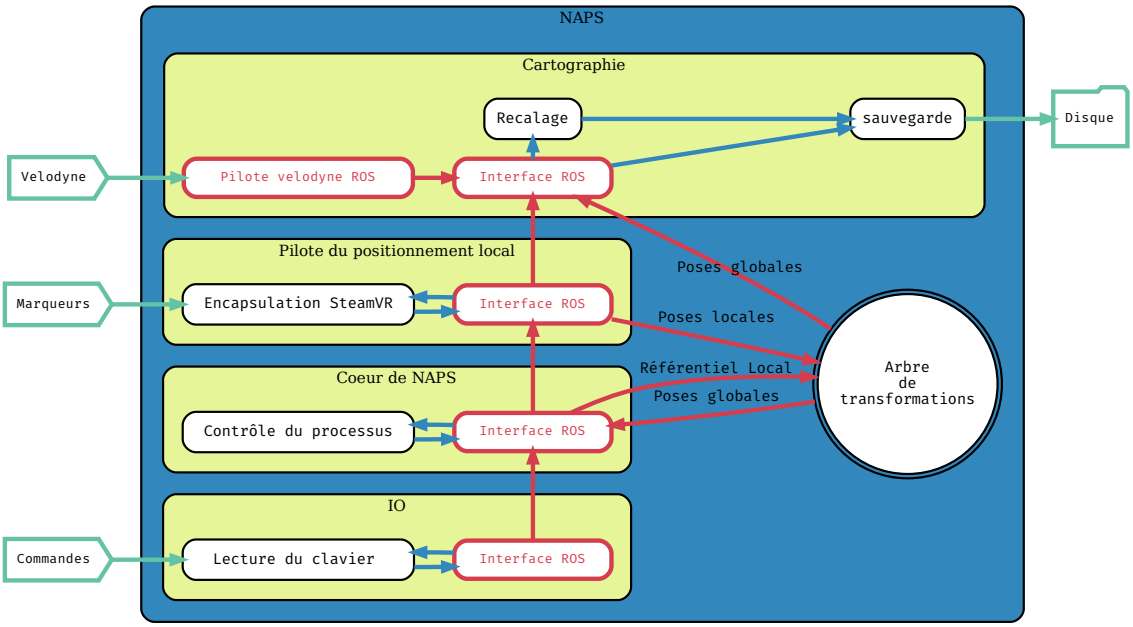

FIGURE 4.3 – Interfaçage du module de cartographie dans NAPS

le reste des systèmes. Cela permet d'une part d'utiliser cette logique en dehors de ROS, ou éventuellement de changer d'intergiciel de communication.

Logique de recalage C'est le coeur du travail effectué par le module de cartographie. Il récupère via l'interface les informations issues des pilotes de positionnement et du lidar et les combine selon une logique qui sera détaillée dans la section 4.4.

Utilitaire de sauvegarde Cet outil prend en entrées les nuages recalés par la partie logique, ainsi que l'arbre de poses fourni par l'interface ROS, effectue une compression et stocke chaque nuage dans un fichier indépendant, horodaté. Chaque fichier contenant un nuage contient aussi l'arbre des poses pour tous les éléments du système, permettant un accès ultérieur à ces précieuses informations. Deux modes sont proposés, sauvegarde à la volée ou lorsque explicitement demandé, par exemple par l'utilisateur.

Afin de pérenniser la solution proposée, nous avons choisi d'adapter les développements précédents pour les rendre compatibles avec ROS2. En effet, ROS2 n'était encore qu'une ébauche lorsque nous avons commencé nos développements, mais il est désormais disponible. Ce travail a été facilité par le découplage systématique opéré entre la logique de chacun des processus, et l'interface ROS associée. Ce choix de migration était nécessaire pour assurer la réutilisabilité du travail fourni, et ce pour les années à venir. Bien que ROS1 ne soit pas amené à disparaître demain, les nouveaux développements de la communauté et des pilotes constructeurs vont rapidement se tourner vers ROS2. Il sera donc plus facile d'intégrer la solution logicielle proposée dans de nouveaux projets.

4.4 Logique de recalage et d'amélioration des nuages

Cette section a pour objectif de présenter les mécanismes de synchronisation, de filtrage et d'augmentation, qui permettent d'obtenir un recalage plus précis de chaque nuage, conduisant mécaniquement à l'obtention d'un modèle de meilleure qualité.

4.4.1 Mécanisme d'association temporelle

Le nœud de cartographie NAPS est le module de cartographie. C'est ici que nous traitons les nuages de points. Pour synchroniser les nuages issus du lidar (10Hz) et les données de suivi (250Hz), nous maintenons un tampon avec les données de suivi, puis nous trouvons l'échantillon le mieux adapté dans ce tampon pour un nuage de points reçu.

Lorsqu'un nuage de points est reçu, nous sélectionnons la transformation correspondante dans le tampon et l'appliquons à chaque point du nuage de points pour le référencer dans le repère global, c'est le recalage du nuage. Les données des transformations de tous les dispositifs sont également intégrées au nuage de points, afin de permettre le filtrage hors ligne des points. Ensuite, le nuage de points peut être sauvegardé ou stocké en mémoire, pour être enregistré ultérieurement dans un fichier. Chaque nuage de points est sauvegardé dans son propre fichier, qui est nommé avec la date et l'heure précise.

4.4.2 Filtrage en fonction de la vitesse

Lors de la récupération de la position du marqueur, nous récupérons également les vitesses instantanées produites par la centrale inertielle du marqueur. Ces données nous servent à inhiber l'enregistrement des nuages en cas de vitesse trop élevée. En effet, le scan du lidar dont nous disposions est un processus itératif. Bien qu'importante, la vitesse de rotation —et donc d'acquisition— du lidar est limitée. Ainsi, si le robot portant le lidar se déplace trop rapidement, le début et la fin du scan ne correspondront pas, menant à une déformation du nuage récolté. Inhiber l'enregistrement d'un tel nuage déformé permet donc de limiter l'impact de ce phénomène sur le modèle final.

4.4.3 Diminution du bruit statique

À l'inverse du mécanisme d'inhibition présenté précédemment, nous proposons un système de diminution du bruit lorsque le robot ne bouge pas. Concrètement, lors de la réception d'une information de positionnement, si la pose du marqueur du lidar n'a pas changé —on fait la différence entre la dernière pose reçue et l'actuelle, et on la compare à un seuil— on regroupe ces échantillons au sein d'un ensemble. Si la prochaine pose reçue n'a pas changé, on l'ajoute également à l'ensemble. Et ainsi de suite jusqu'à réception d'une pose correspondant à un mouvement.

Lors de la réception d'un nuage de points dont l'horodatage correspond à un tel groupe, une moyenne est calculée sur l'ensemble des poses de ce groupe. Ce mécanisme permet d'atténuer le bruit et de produire par conséquent un recalage plus précis du nuage de points dans le repère global.

4.4.4 Filtrage des éléments du système

Le filtrage hors-ligne des perturbations de mesure induites par les agents du système est aisé car chaque fichier contenant un nuage contient aussi l'ensemble des poses des éléments du système. Ainsi, si une géométrie est définie pour chaque élément —cela peut être un modèle 3D du robot ou tout simplement une distance définissant une sphère dans laquelle est inclus le robot— il suffit de supprimer tous les points à l'intérieur de ce volume, qui sera positionné grâce aux informations du système. Toutefois, il n'est pas possible de supprimer toutes les perturbations. Bien qu'il soit possible d'enlever les points correspondants aux éléments du système, il n'est pas possible de régénérer les points qui auraient dus êtres mesurés par le laser s'ils n'avaient pas été cachés par l'agent.

4.5 Expérimentations et Résultats

Nous avons mené une série d'expérimentations consistant à scanner le bâtiment du LORIA. Nous avons donc produit plusieurs jeux de données, composés de nuages points horodatés et géoréférencés. Seule une sélection de trois environnements avec des caractéristiques spécifiques listées dans le tableau 4.1 sera présentée dans ce chapitre.

TABLE 4.1 – Jeux de données produits

Nom	Enveloppe couverte	# Nuages	# Points	Figure(s)
Boucle	$15m \times 15m \approx 225m^2$	893	1.20e+07	4.4
Bureaux	$40m \times 32m \approx 1300m^2$	4434	6.39e+07	4.5
Couloir	$92m \times 32m \approx 3000m^2$	6114	1.14e+08	4.6, 4.7

4.5.1 Étude qualitative

Nous allons présenter les spécificités des trois jeux de données ainsi que les modèles 3D reconstruits à partir de ces jeux.

Le premier modèle présenté est issu du jeu **boucle**. Il correspond à une boucle faite dans un petit corridor étriqué, encombré de divers obstacles comme des étagères. Il est important de préciser qu'un seul phare a été mis en oeuvre, diminuant ainsi grandement la précision du système de positionnement. Le modèle reconstruit est présenté sur la figure 4.4. On constate l'importante erreur d'orientation accumulée lors du parcours de cette boucle, qui a nécessité de nombreux repositionnements du phare. Il est important de préciser que le carré orange au plafond est une particularité architecturale et non un défaut de reconstruction.

Le second modèle correspond au jeu **bureaux**, qui couvre une plus grand surface, mais surtout une plus grande variété d'environnements. En effet, cette expérimentation a débuté dans les larges espaces du couloir, de la salle de réunion du LORIA, puis s'est poursuivie dans le hall du bâtiment C pour finir dans de petits bureaux, reliées entre eux par un étroit corridor. Le résultat obtenu est présenté par la figure 4.5. Le résultat obtenu est globalement fidèle à la géométrie des lieux, bien que l'on puisse constater une fois plus des déformations dues à l'erreur d'orientation. La partie comprenant le corridor et les bureaux est en bas à droite de l'image.



FIGURE 4.4 – Vue du dessus du pire cas d'erreur de lacet

C'est, semblerait-il les environnements les plus difficiles à cartographier pour notre système.

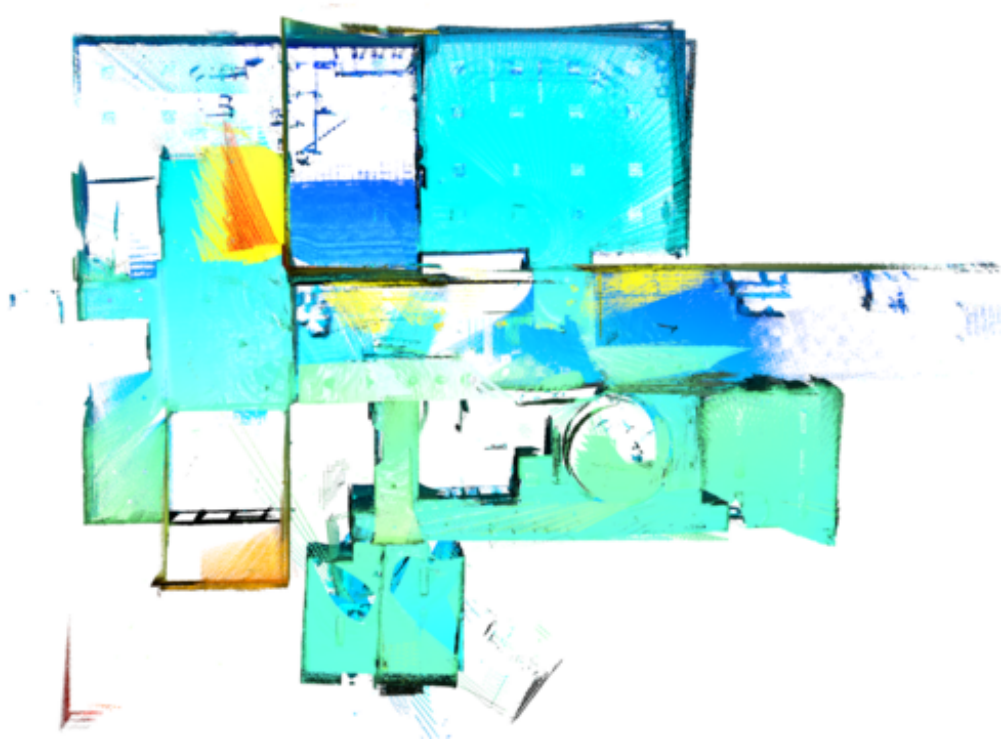

FIGURE 4.5 – Vue du dessus du modèle reconstruit, jeu *Bureaux*

Enfin, la plus longue expérimentation réalisée, correspondant au scan de l'intégralité du **couloir** du bâtiment C du LORIA. Nous avons également visité l'atrium, qui est la grande structure visible sur les figures 4.6 et 4.7. Il est important de constater le caractère rectiligne du couloir reconstruit, fidèle à la géométrie initiale du bâtiment. Dans cette expérience, le système a évolué dans un espace relativement dégagé, typique des mines de calcaires initialement visées. La précision du modèle reconstruit, visible également sur la figure 4.7 montre qu'il s'agit de l'environnement de prédilection de notre système, en tous cas dans le cadre de son emploi manuel. Il est en effet plus facile de positionner correctement les phares et de faire évoluer le couple marqueur/lidar dans un environnement libre. Ici, l'enchaînement moins tortueux des déplacements du système ne produit que peu d'erreurs d'orientations.

L'analyse qualitative des modèles obtenus permet de valider l'approche de génération de modèles bruts, qui malgré leur défaut d'orientation représentent des modèles exploitables. Les modèles denses reconstruits permettent en effet de s'immerger dans l'environnement pour un humain, ou d'appliquer des algorithmes comme la recherche de chemins pour un robot. Nous avons également identifié les environnements larges et ouverts comme terrains de prédilection pour notre système de reconstruction. Malgré les bons résultats obtenus dans de telles conditions —semblables aux environnements initialement visés, les mines— il convient d'adresser les problèmes d'orientation identifiés.

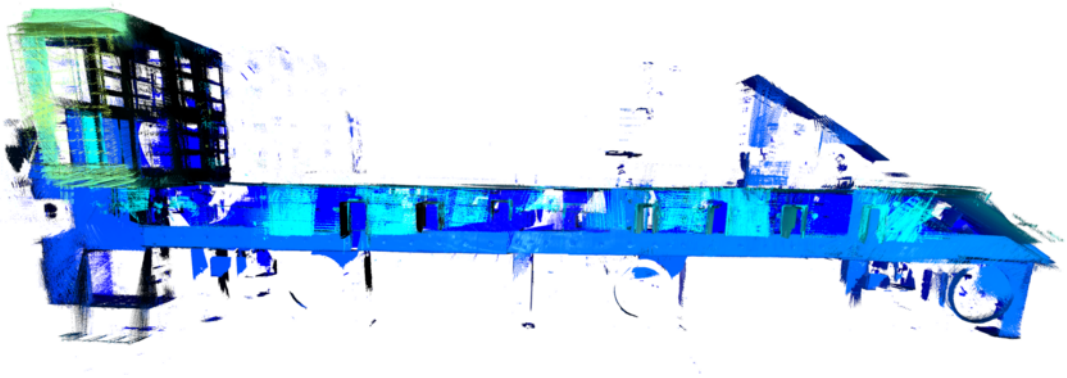

FIGURE 4.6 – Vue du dessus du modèle du couloir du bâtiment C du LORIA

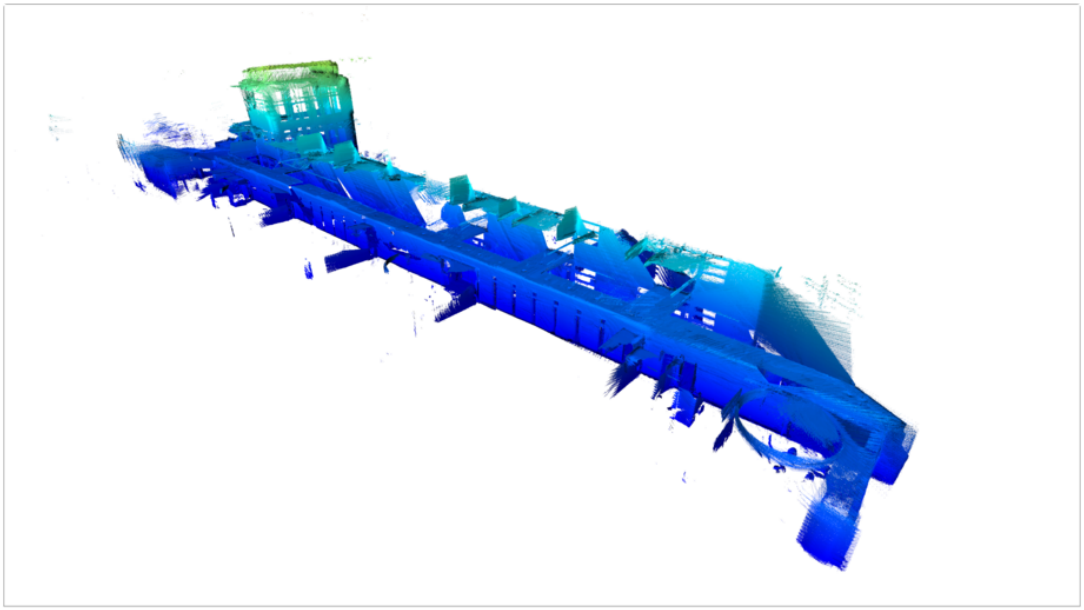

FIGURE 4.7 – Vue de biais du modèle du couloir du bâtiment C du LORIA

4.5.2 Étude quantitative

La production d'une analyse quantitative est difficile pour un modèle 3D reconstruit. En effet, l'acquisition de la vérité terrain —un autre modèle 3D de l'environnement— demande l'intervention de géomètres experts, employant du matériel haut de gamme (laser très précis de l'ordre du mm contre $3cm$ pour notre lidar). Malheureusement, une telle intervention s'est avérée impossible à mettre en oeuvre dans le contexte spécifique auquel nous à confronté la pandémie mondiale. De plus, même si nous disposions de tels modèles, les métriques à mettre en oeuvre pour évaluer la similitude entre deux nuages ne sont pas absolues. Afin d'évaluer quantitativement la reconstruction obtenue, il aurait fallu appliquer les mêmes métriques d'erreur sur les résultats produits par d'autres algorithmes de SLAM. Systèmes dont nous ne disposons pas.

Concernant la translation, l'ordre de grandeur de son erreur étant inférieur à celui de l'erreur du capteur, il n'est pas la source principale des déformations observées sur le modèle reconstruit.

Attardons nous toutefois sur les erreurs angulaires, qui sont facilement identifiables, notamment dans la figure 4.4, qui représente le cas extrême constaté lors de l'ensemble de nos expérimentations. Ici, l'erreur de lacet est très importante $\approx 10°$. Cela s'explique par l'accumulation des erreurs lors du passage dans cette boucle exiguë, qui a nécessité de nombreux repositionnements de l'unique phare utilisé. De plus, la précision de l'estimation d'orientation calculée dépend fortement du maintien de la visibilité entre le phare et le marqueur du lidar, contrainte qu'il est difficile de garantir lorsqu'il faut déplacer manuellement les éléments du système dans un endroit étriqué. Ce maintien de la visibilité fera l'objet de dévelloppements futurs, présentés dans le chapitre 6. Nous tenions à présenter le pire cas constaté, toutefois, la majorité des erreurs d'estimations de lacet sont bien plus raisonnables. En effet, même les erreurs visibles comme sur le coin en haut à droite du modèle présenté sur la figure 4.5, ne représentent que quelques degrés (dans cet exemple $\approx 2.85°$). L'effet de distorsion constaté est dû à la distance entre les points et le centre du lidar lors du scan, qui amplifie l'effet de l'erreur de la rotation estimée.

4.6 Améliorations

Nous avons vu précédemment que le principal axe d'amélioration se situait du côté de l'estimation de l'orientation. Plusieurs axes d'amélioration sont présentés et feront l'objet de futures recherches et développement, notamment dans le cadre du transfert technologique en cours. Nous n'aborderons ici pas le problème déjà évoqué dans la section 4.2.2 d'erreur dues à l'imprécision dans le mécanisme de fixation du marqueur au lidar. Il s'agit d'un problème qui, bien qu'il explique en partie les erreurs constatées, relève de l'ordre de l'ingénierie mécanique —parfaitement maîtrisée par l'industrie— plutôt que de la recherche en robotique.

4.6.1 ICP et raffinement Bayésien

Les erreurs en rotations sont plus faciles à corriger par l'ICP, car les nuages disposent de mesures lointaines, ou le moindre recalage influe beaucoup sur l'estimation d'orientation. Nous nous sommes donc interrogés sur la possibilité d'utiliser un raffinement hors-ligne pour améliorer la reconstruction obtenue. Ce processus de raffinement est long et ne peut pas être appliqué en

temps-réel, ce qui exclut son utilisation pendant la mission avec NAPS.

L'idée consiste à utiliser l'estimation de transformation entre deux nuages fournie par NAPS comme transformation initiale pour l'algorithme d'ICP (*point-to-plane*) qui atténue fortement les imprécisions angulaires. Nous utilisons également un *Robust-kernel* configuré à partir d'un seul paramètre, l'écart type σ du capteur utilisé. Dans notre cas, l'écart type du lidar est $\sigma = 0.05m$.

De plus, un coûteux mécanisme de détection de boucle —par comparaison, pour chaque nuage, avec un échantillon des autres nuages du jeu de données— permet de réduire les erreurs lorsque l'on revient sur ses pas.

Les estimations de poses et les détections de boucles sont ensuite confiées à un *factor-graph* qui optimisera —lissera— la trajectoire du lidar à partir de la précision de chaque recalage. L'implémentation Python de ce mécanisme de raffinement repose entièrement sur la récente librairie Open3D déjà évoquée.

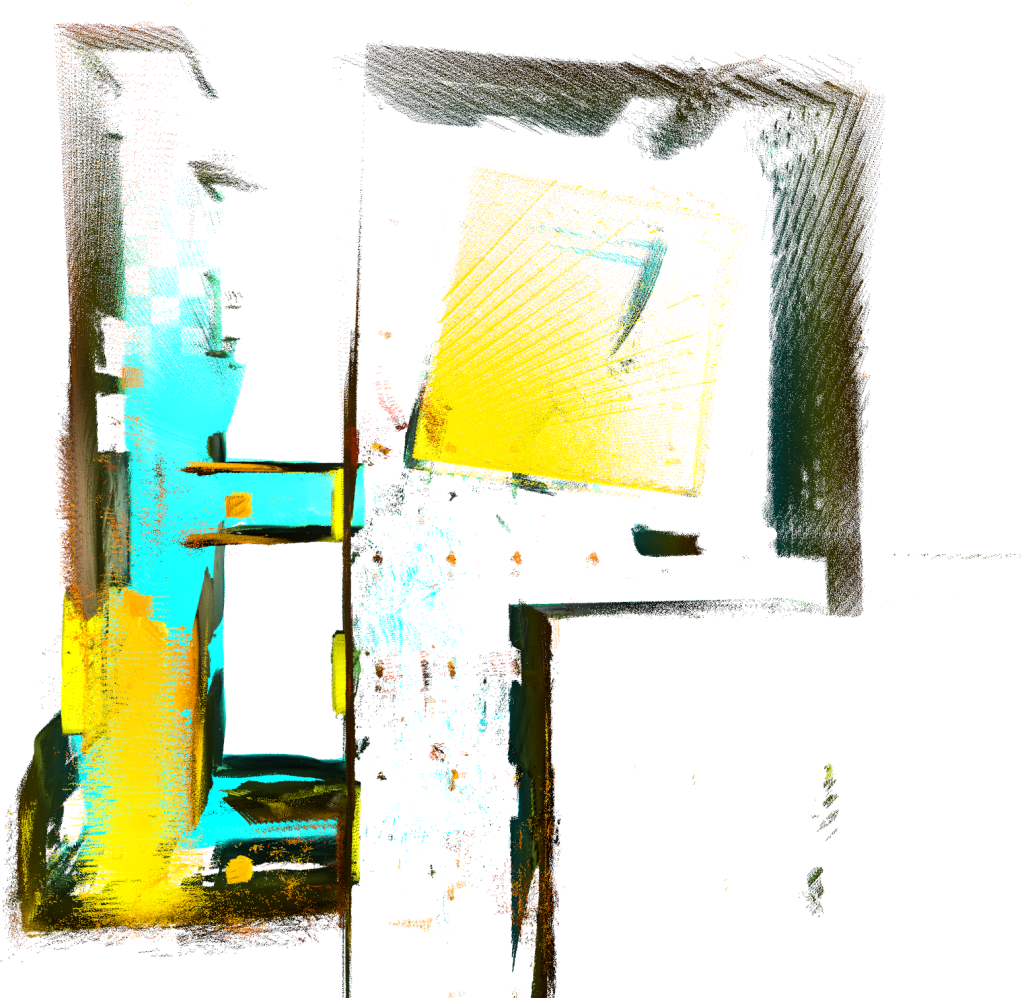

FIGURE 4.8 – Modèle obtenu par ICP et Factor-graph, à partir des poses fournies par NAPS

Les résultats obtenus pour le jeu *Couloir* sont présentées dans la figure 4.8. On peut voir que

la forte erreur angulaire a été corrigée. Afin de valider l'implication de notre système de positionnement dans la qualité des résultats finaux, nous avons soumis les nuages de points vierges —avant recalage par notre système— au même processus de recalage hors-ligne. Le *résultat* obtenu est présenté dans la figure 4.9. Il semblerait que la configuration de l'expérimentation ne présente pas un recouvrement suffisant, et que l'absence de transformation initiale précise conduise l'ICP à converger vers des minimaux locaux, conduisant inévitablement à l'échec du système.

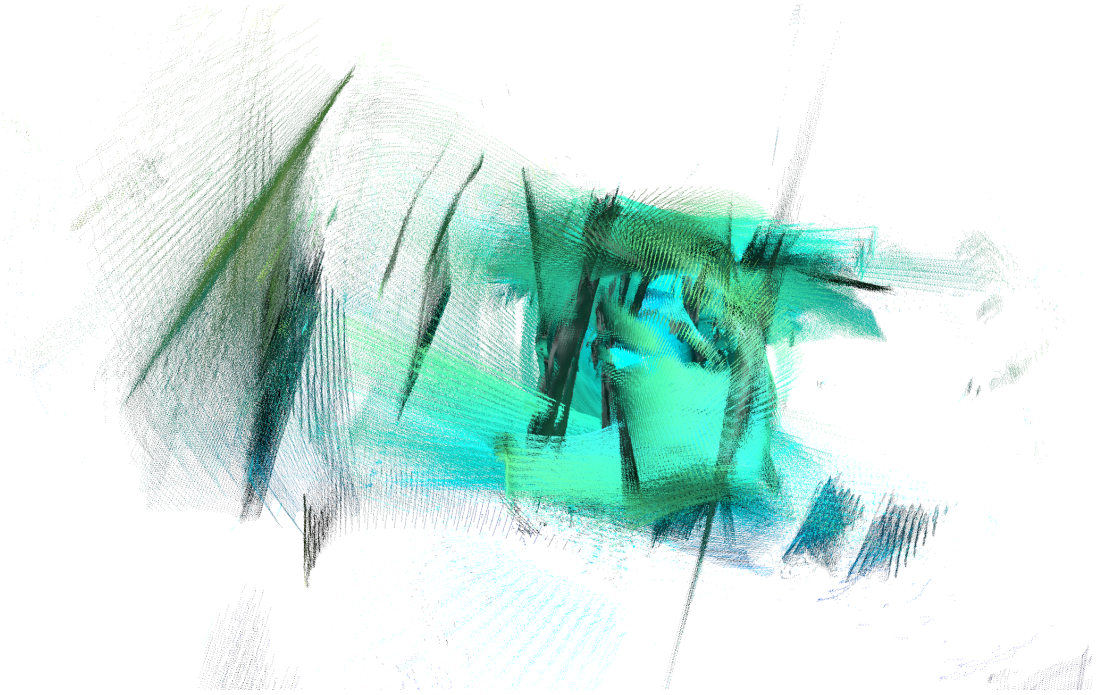

FIGURE 4.9 – Modèle obtenu par ICP et Factor-graph, sans les poses fournies par NAPS

Nous avons donc mis en oeuvre avec succès un système de raffinement hors-ligne, permettant de gommer les défauts en orientation du modèle brut reconstruit par NAPS. Toutefois, ce processus lourd en calculs est très loin de pouvoir être exécuté en temps réel. Il convient donc d'explorer d'autres pistes permettant de minimiser l'erreur angulaire dans le processus de capture.

4.6.2 Vers d'autres marqueurs

L'erreur d'estimation de l'orientation de NAPS est directement héritée de l'erreur faite par le capteur fournissant les estimations locales. Or cette erreur est elle même issue de l'erreur sur l'estimation d'orientation faite par l'algorithme *Perpective-n-Points (PnP)* présenté dans la section 3.1.2. Cet algorithme, se base sur la position d'un ensemble de points dans l'espace image pour déterminer l'orientation du solide composé de ces points —ici notre marqueur. L'erreur faite lors de la mesure de la position des points —correspondants aux récepteurs lasers du marqueur— se répercute alors naturellement sur l'estimation d'orientation. Seulement, cette erreur ne se répercute pas avec la même intensité en fonction de l'éloignement entre les points et le centre du marqueur. En effet, plus le point est éloigné du centre, moins son imprécision impacte le calcul de

l'orientation. Afin d'illustrer ces propos, considérons l'exemple présenté sur la vignette de gauche de la figure 4.10. Dans cet exemple réduit à 2 dimensions pour la visualisation, deux solides A et B sont composés de deux récepteurs —points— à leurs extrémités. A' et B' sont les positions estimées, respectivement éloignées de la position réelle d'une même distance : $A'A = B'B = \epsilon$. Les angles $\widehat{AOA'}$ et $\widehat{BOB'}$ représentent l'erreur d'orientation associées à chacun des solides. Il est alors facile de montrer que la longueur supérieure du solide B permet de réduire l'erreur en rotation car pour un même ϵ, $\widehat{AOA'} > \widehat{BOB'}$.

La seconde vignette de la figure 4.10 présente l'évolution de l'erreur de lacet maximale en fonction du diamètre du marqueur. Cette courbe est calculée en prenant pour base le pire cas constaté pour l'estimation du lacet : 10°, sachant que le marqueur que nous utilisons à un diamètre $d = 10cm$. Un simple calcul trigonométrique permet par la suite de déterminer, pour divers diamètres d'objets, l'erreur maximale théorique.

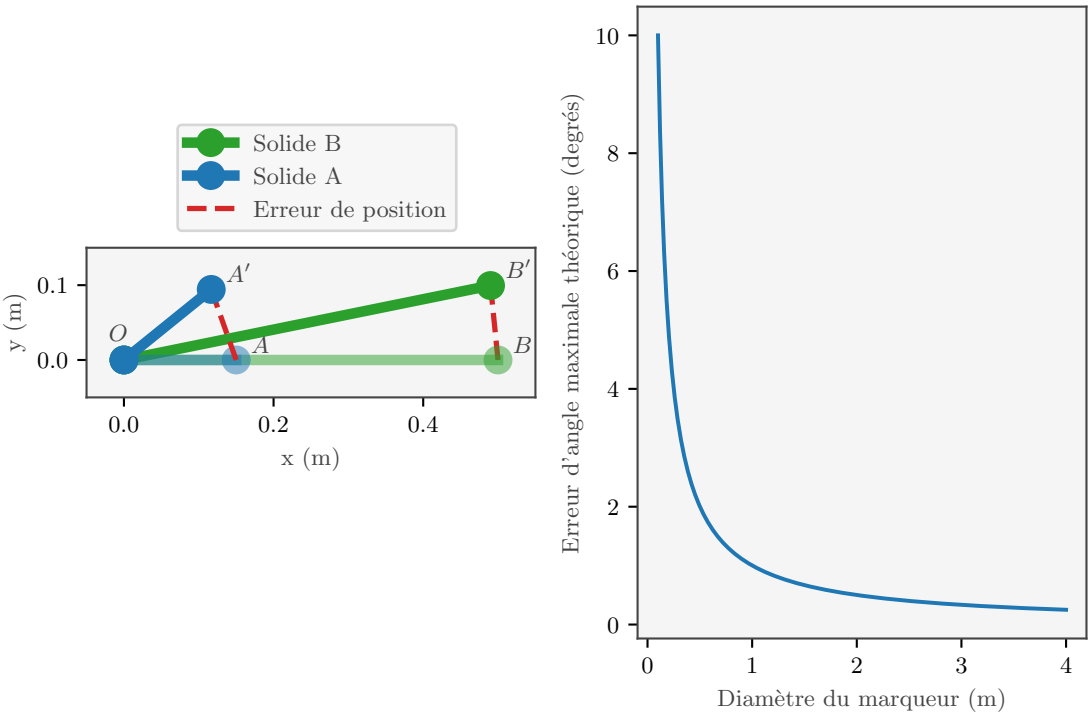

FIGURE 4.10 – Diminution de l'erreur avec un plus grand capteur

Nous l'avons vu, l'augmentation de la taille des marqueurs permettrait d'augmenter drastiquement la précision angulaire du système de positionnement, corrigeant ainsi à la source les problèmes de déformations observés sur les modèles 3D reconstruits. Mais comment procéder pour réaliser un tel marqueur ?

Le site partenaire [16] de *Steam* (propriété de *valve* qui fournit le logiciel *SteamVR*) propose des outils et les plans de l'électronique nécessaire à la mise en œuvre de marqueurs personnalisés. Couplés à une matrice de convertisseurs de signaux lumineux [17], il est possible de fabriquer des

16. Kit de développement propriétaire https://partner.steamgames.com/vrlicensing#DevKit
17. Photo-décodeur https://triadsemi.com/product/ts4231/

marqueurs de n'importe quelle taille et de n'importe quelle forme. Un marqueur personnalisé serait idéal, car en plus de réduire l'erreur angulaire de part sa taille, il pourrait mieux s'adapter aux robots afin de réduire les *angles morts*. Toutefois, la conception et la mise en œuvre d'une telle solution demande des compétence et des ressources en électronique dont nous ne disposons pas.

Il a alors fallu se reporter sur une autre solution. Plutôt que de construire à partir de zéro un marqueur plus grand, nous pouvons en combiner plusieurs, positionnés et fixés ensemble de manière avantageuse. Ces n marqueurs formeraient ensemble un seul solide, et nous pouvons connaître leurs positions relatives. La figure 4.11 représente l'arbre de transformations d'un tel système. Les transformations en rouges sont les transformations fixes, définie par la géométrie de la pièce mécanique —ou du robot— qui relie à la fois les différents marqueurs et le lidar. Le choix de combiner les marqueurs au sein d'un **meta-marqueur** intermédiaire permet d'interchanger de manière transparente le capteur —ici le lidar. Chaque marqueur incorporé dans le meta-marqueur estime sa propre position, conduisant le système à générer n estimations de la pose du meta-marqueur. Plusieurs approches sont possibles pour fusionner ces poses —qui correspondent bien au même objet physique. La plus simple, qui ne demande pas d'heuristiques supplémentaires, consiste à utiliser la moyenne des poses fournies comme **meta-estimation** de la pose du meta-marqueur. Il est également possible d'utiliser des approches basées sur le consensus comme *RANSAC* présenté précédemment pour effectuer cette fusion. Une fois cette pose déterminée, on procède de la même manière qu'avec un seul marqueur pour déterminer la pose du lidar, ou de tout autre capteur. Bien qu'attrayante, cette solution nécessite encore une fois une mécanique de précision pour être mise en œuvre, sinon quoi la pose estimée du lidar ne correspondra pas à la réalité. Nous n'avons pas encore eu l'occasion de tester un tel prototype, mais c'est à l'ordre du jour.

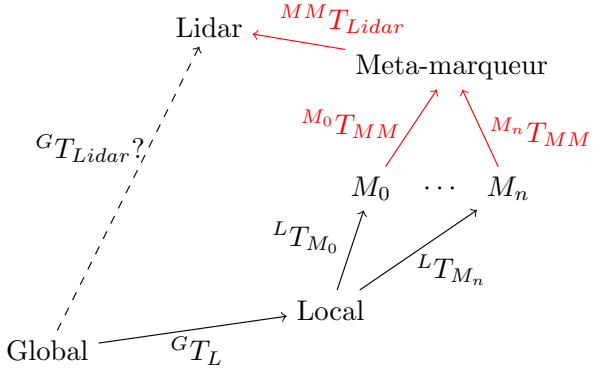

FIGURE 4.11 – Calcul de la pose avec un *meta-marqueur*

4.6.3 Contrôle robotisé du processus d'exploration

Enfin, un autre axe d'amélioration se situe dans l'automatisation complète du processus, et particulièrement dans le calcul du placement des éléments. Le placement des phares a une incidence sur la performance du système. En effet, un phare est placé à une position dite sous-optimale s'il existe une autre position qui permettrait de couvrir plus d'espace. Couvrir plus d'espace permet de diminuer le nombre de déplacements nécessaires pour mener à bien une

mission donnée. Or, c'est uniquement lors du déplacement des phares que de l'incertitude est ajoutée aux estimations de poses des éléments du systèmes. De plus, assurer la visibilité constante entre le phare et les marqueurs n'est pas évident pour un humain, particulièrement dans des endroits exigus et encombrés comme dans le cas de l'expérimentation *boucle*. Garantir la visibilité constante tout en calculant les positions optimales des phares est donc un axe majeur d'amélioration de la solution proposée, en plus d'être nécessaire à son autonomisation complète. Les travaux effectués en ce sens sont présentés dans le chapitre 6.

4.7 Conclusion

Nous avons présenté dans ce chapitre un cas concret d'utilisation du système de positionnement développé durant la thèse. L'approche est impressionnante de part sa sobriété en calculs et permet de traiter en temps réel les $300k$ points mesurés par le lidar toutes les secondes. Cette simplicité n'implique toutefois pas de compromis sur les modèles obtenus, exploitables sans traitements supplémentaires malgré la faiblesse du système concernant l'estimation d'orientation. Défaut qui sera sans nul doute rapidement corrigé, notamment par l'emploi d'un *méta-marqueur*, doté d'un algorithme de fusion et assemblé mécaniquement de manière fine, par de potentiels partenaires industriels.

Le système proposé se démarque principalement par sa simplicité, et le peu de paramétrages et de travaux préliminaires nécessaires à son emploi par un utilisateur lambda dans un nouvel environnement. Nous avons toutefois constaté que les environnements exigus rendent difficile l'emploi manuel d'un tel système, et qu'il ne donne son plein potentiel que dans les espaces plus ouverts comme dans l'expérimentation *couloir*.

Bien que produisant initialement des modèles bruts, il reste tout à fait possible d'intégrer le système de positionnement et de capture dans un processus plus large comprenant par exemple un traitement hors-ligne plus fin des modèles obtenus. Cette intégration judicieuse du modèle brut reconstruit par NAPS dans un processus de raffinement s'est avérée corriger complètement les défauts d'orientation du nuage brut. Il est cependant important de rappeler que ce processus de raffinement n'a de sens que si une exploration autonome en temps réel est rendue possible par un système de cartographie moins fin comme celui proposé dans ce chapitre.

Troisième partie

Contribution : planification

Table des matières

Chapitre 5
Génération de trajectoire

5.1 Lecture de la carte . . . 95
5.2 Création de zones restreintes . . . 96
5.3 Géométrisation des zones restreintes . . . 97
5.4 Calcul du diagramme de Voronoï . . . 100
5.5 Filtrage des frontières . . . 103
5.6 Structuration des segments . . . 103
5.7 Organisation en allées . . . 107
5.8 Plus court chemin . . . 109
5.9 Construction de la solution . . . 113
5.10 Ré-échantillonage des points . . . 114
5.11 Lissage de la trajectoire . . . 116
5.12 Approche discrète . . . 121
5.12.1 Données du problème pour le calcul de la trajectoire du robot . . . 121
5.12.2 Processus de calcul du chemin . . . 121
5.13 Constat et évolutions . . . 123

Chapitre 6
Calcul des placements des Phares

6.1 Modélisation du problème local . . . 128
6.2 Cas idéal . . . 131
6.3 Sentiers adaptés à l'environnement . . . 132
6.4 Réduction de l'espace de solutions . . . 133
6.5 Vers une heuristique d'orientation des phares . . . 136
6.6 Un modèle des occlusions de l'environnement . . . 138
6.7 Fonction d'évaluation de solution locale . . . 139
6.8 Optimisation sur les sentiers . . . 142
6.9 Construction d'une solution globale . . . 142
6.10 Prendre en compte plus de solutions . . . 145
6.11 Une meilleure solution locale . . . 148
6.12 Échantillonage multi-échelle . . . 148

6.13 Une meilleure solution globale ? 151
6.14 Approche alternative discrète 153
6.14.1 Données du problème 153
6.14.2 Résultats attendus 153
6.14.3 Processus itératif de calcul 153
6.15 Constat et évolutions 155
1 Contexte 159
2 Contributions 159
3 Perspectives 161
3.1 Amélioration des propositions 161
3.2 Étendre le champ d'application 161

Chapitre 5

Génération de trajectoire

On a présenté précédemment un système de suivi et de cartographie précis mais nécessitant un opérateur humain pour choisir et réaliser les objectifs (déplacements) de la mission. Afin de rendre notre système autonome, il faut dans un premier temps déterminer la prochaine position/objectif à atteindre et ensuite déterminer une trajectoire correspondante. La première partie ne relève pas de NAPS car elle est directement dépendante de la nature de la mission. Une fois l'objectif fourni par le gestionnaire de mission, la seconde partie relative à la trajectoire peut être intégrée au système NAPS afin de le rendre totalement autonome.

La génération de trajectoire nécessite un point de départ, une destination et une représentation de l'environnement. Cette dernière peut être complète ou partielle, connue a priori ou calculée dynamiquement, selon le contexte de la mission.

Dans le domaine de l'exploration d'environnements, il existe des algorithmes efficaces de génération de trajectoire tel que Rapidly-Exploring Random Tree (RRT) [LaValle, 1998]. Cependant, ils ne tiennent pas compte des contraintes spécifiques de la cartographie et ne permettent pas de garantir un placement au centre de l'espace libre maximisant les informations pouvant être collectées en suivant la trajectoire.

Les algorithmes de calcul de trajectoire que nous présentons dans ce chapitre sont opérationnels quelles que soit l'étendue et la forme de l'environnement. Il est important de noter que dans une mission d'exploration, l'environnement connu augmente au fur et à mesure du déroulement de la mission et cette information est directement prise en compte pour calculer les trajectoires suivantes.

Dans ce chapitre, par souci de clarté et afin de pouvoir mettre en évidence les capacités des algorithmes présentés, nous utilisons des cartes d'environnement quasiment complètes.

5.1 Lecture de la carte

Le choix d'une simple image comme point d'entrée n'est pas innocent, ce format a l'avantage d'être courant pour stocker des cartes, il sera ainsi aisé pour un utilisateur de fournir un plan existant. Dans l'autre cas de figure, où les données d'entrées sont produites directement à partir des informations acquises, c'est également une image qui est générée en direct, la carte présentée

dans la figure 5.1 en est un bon exemple. Ici, seul l'intérieur de la maison a été exploré, l'extérieur restant à découvrir. Les algorithmes décrits dans cette partie fonctionnent de manière transparente sur une carte complète ou partielle.

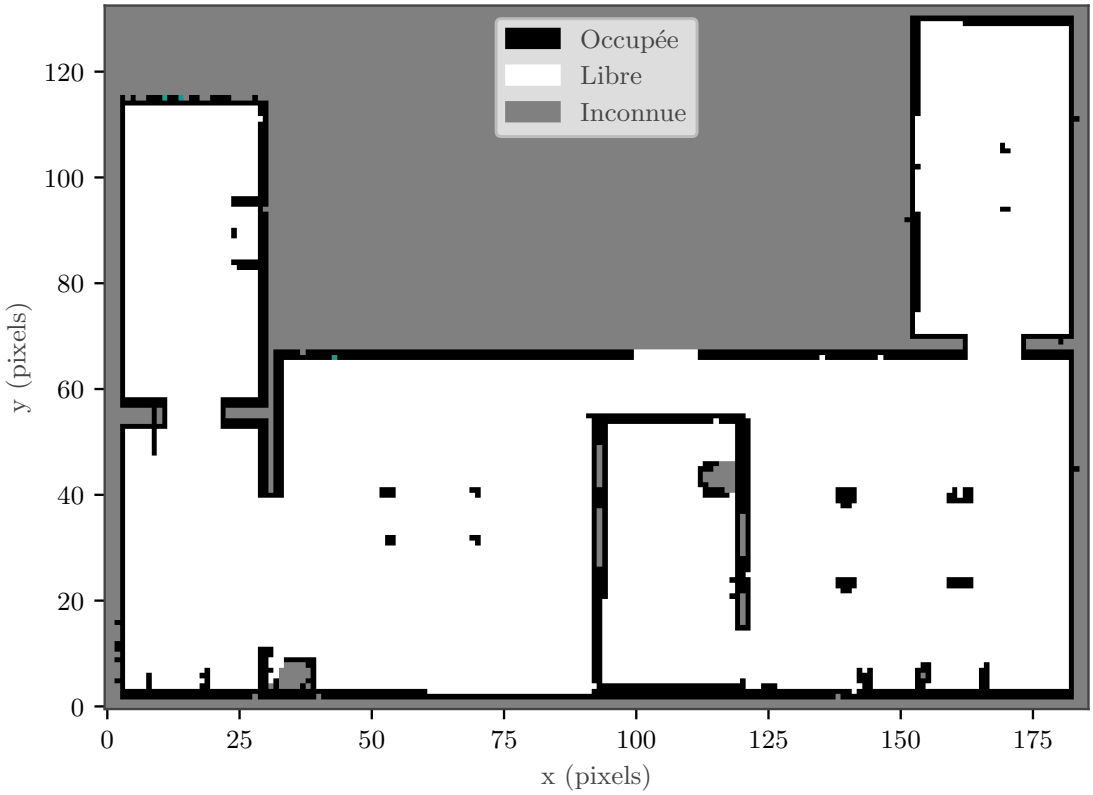

FIGURE 5.1 – Exemple de carte sous forme d'image représentant une maison

Cette image servira de point de départ dans le processus de planification décrit dans ce chapitre. Dans notre cas, nous interpréterons chacune des valeurs des pixels comme étant la probabilité de rencontrer un obstacle dans ce pixel. Nous convertissons dans un premier temps cette image en niveaux de gris, nous permettant avec un simple seuil de catégoriser chaque pixel :

Libre probabilité quasi-nulle que la case contienne un obstacle.

Occupée Forte probabilité de rencontrer un obstacle.

Inconnue Les données de la carte ne permettent pas d'établir avec suffisamment de confiance la nature du pixel.

5.2 Création de zones restreintes

Une des données d'entrée du problème, mais également une contrainte importante de notre problème de planification, est la distance de sécurité minimale à respecter à tout instant lors de la mission. Cette distance permet d'éviter toute collision avec les obstacles.

Afin de garantir de manière simple et robuste cette contrainte de distance minimale d —dite de sécurité— tolérée entre un agent et un obstacle, nous allons artificiellement former des **zones restreintes** autour des obstacles, tel que les contours extérieurs de ces zones soient à une distance équivalente à la *distance de sécurité* des obstacles. La figure 5.2 présente le résultat de

cette opération de traitement d'image, effectué avec la bibliothèque *scikit-image* [der Walt et al., 2014].

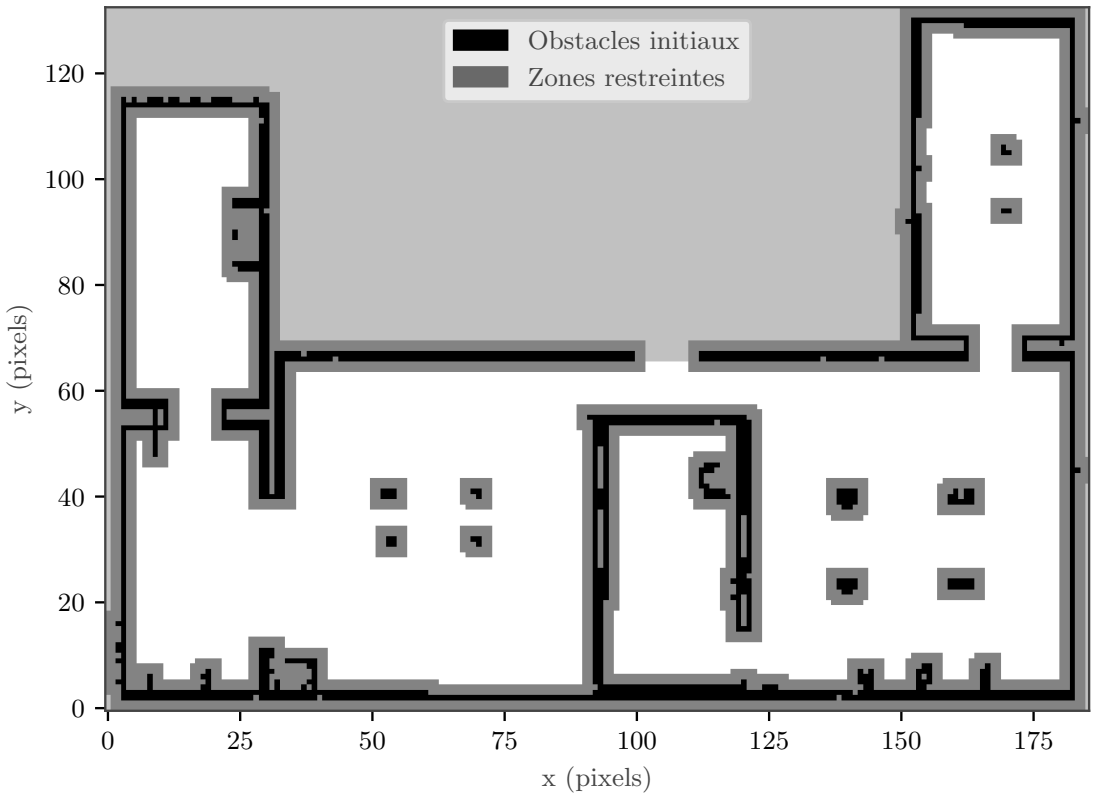

FIGURE 5.2 – La même carte, une fois les obstacles dilatés

Ainsi, chaque case libre —blanche dans notre cas— sera accessible sans risques de collision avec un obstacle. L'intérêt principal de cette méthode réside dans sa simplicité. Effectivement, une fois cette opération réalisée, il ne sera plus nécessaire de vérifier les contraintes relatives aux obstacles et aux caractéristiques de l'agent, nous permettant de sauvegarder un précieux temps de calcul ultérieurement.

5.3 Géométrisation des zones restreintes

Nous avons maintenant des obstacles dilatés, représentant plus fidèlement l'espace interdit d'accès pour nos agents. Nous avons maintenant besoin de délimiter les contours de ces zones restreintes, afin de pouvoir les manipuler plus aisément par la suite.

Afin de répondre à cette problématique, nous allons employer l'algorithme dit *marching squares* qui est un cas spécifique de l'algorithme *marching cubes* [Lorensen and Cline, 1987]. Une étape préliminaire de seuillage est nécessaire, permettant de distinguer plus nettement les obstacles. L'algorithme, présenté dans la figure 5.3, consiste à se placer entre 4 pixels, et à considérer leurs valeurs pour générer une ligne. Ces valeurs sont figurées par les points blancs —considérés libres— et noirs —considérés comme obstacles— aux extrémités des cas possibles. Le parcours de l'image permet de construire un à un les segments colorés représentés dans le cadrant *Cas*

individuels. Ce même cadrant lie également ces segments aux points qui ont permis de les déterminer, en colorant le carré formé par ces points de la même couleur que le segment. Enfin, les contours obtenus par concaténation des segments précédents sont présentés avec une couleur unique dans le dernier quadrant. L'implémentation de *marching squares* que nous utilisons est celle proposée par *Scikit* [der Walt et al., 2014].

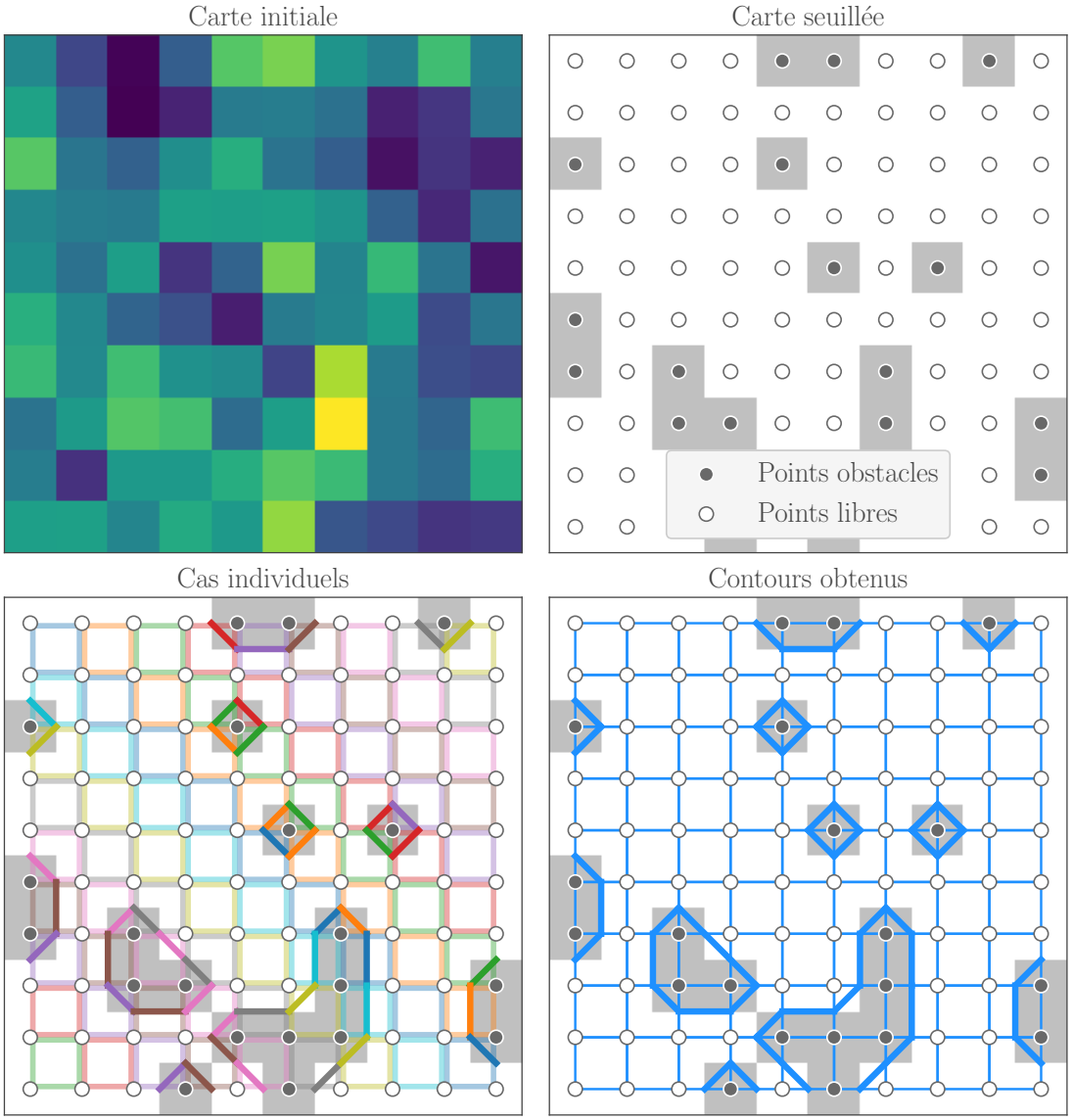

FIGURE 5.3 – Procédure de détermination des contours.

Comme le montre la figure 5.4, les contours obtenus —lignes colorées sur la figure— correspondent bien à ceux des zones restreintes —zones gris souris— préalablement déterminées. Cette étape peut paraître anodine, mais nous venons en réalité de changer de nature de données. En effet, nous disposions auparavant uniquement d'images, nous avons maintenant des données géométriques sous la forme d'un ensemble de contours c_k associés chacun à un obstacle k tel que c_k est une liste de couples de coordonnées (x, y). Voici en guise d'exemple les données du

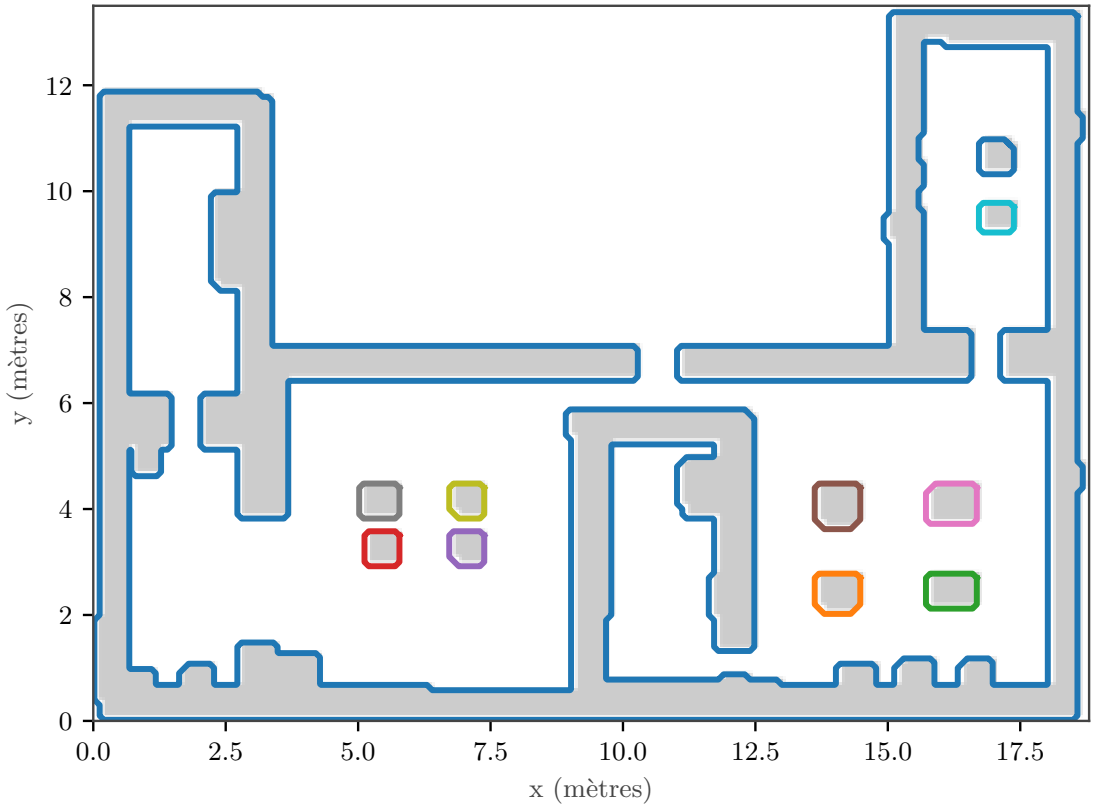

FIGURE 5.4 – Contours obtenus avec l'agorithme marching squares

premier contour détecté —le plus grand sur la figure 5.4— :

$$c_0 = \left[\begin{pmatrix} 133., & 185.8 \end{pmatrix}, \begin{pmatrix} 132., & 185.8 \end{pmatrix}, \ldots, \begin{pmatrix} 16., & 6.8 \end{pmatrix}, \begin{pmatrix} 15., & 6.8 \end{pmatrix} \right]$$

Une dernière étape est nécessaire pour s'affranchir totalement de l'image de départ, il faut intégrer aux contours récemment calculés l'échelle de la carte. Cela permet de passer d'un repère indexé sur les pixels de l'image à un repère du plan $\mathbb{R}^2$. Dans notre cas, chaque pixel représente $10cm$ donc $1/10m$.

$$Echelle = \text{distance en mètres d'un coté du pixel}$$

Le passage en coordonnées cartésiennes consiste alors en une simple multiplication de matrice par le réel $Echelle$:

$$c_k = c_k \times Echelle$$

On obtient alors une liste de coordonnées cartésiennes correspondant à la discrétisation des contours des zones restreintes, discrétisation qui servira de point de départ pour la prochaine étape de planification.

$$c_0 = \left[\begin{pmatrix} 13.3, & 18.58 \end{pmatrix}, \begin{pmatrix} 13.2, & 18.58 \end{pmatrix}, \ldots, \begin{pmatrix} 1.6, & 0.68 \end{pmatrix}, \begin{pmatrix} 1.5, & 0.68 \end{pmatrix} \right]$$

5.4 Calcul du diagramme de Voronoï

Un diagramme de Voronoï est un pavage du plan, réalisé à partir d'un ensemble de points discrets, reflétant l'influence —la proximité— de chacun de ces points sur ce plan. Ce pavage compte de multiples applications, allant de la modélisation de micro structures en Physique et en Chimie jusqu'à la diminution du bruit des images issues de télescopes en Astrophysique, tout en étant utilisé dans l'aménagement du territoire et encore bien d'autres applications concrètes.

La décomposition de Voronoï —opération qui permet d'obtenir un pavage— a été généralisée en dimension n par Georgy Fedoseevich Voronoï en 1908, nous permettant de réaliser ultérieurement ces calculs en dimension 3. Toutefois, les données initiales utilisées par notre planification étant extraites à partir d'une carte à deux dimensions, on se contentera ici de se placer dans le plan $\mathbb{R}^2$.

Soit S l'ensemble fini de points du plan dont nous voulons étudier l'influence ; Une région R_p de Voronoï —ou cellule de Voronoï—, associée à un point p de S est l'ensemble des points qui sont plus proches de p que de n'importe quel autre point de S.

$$R_p = \{x \in X \mid \forall q \in S,\ \|x - p\| \leq \|x - q\|\}$$

Notre utilisation du pavage sera un peu différente de l'étude de l'influence des points. En effet, nous allons utiliser l'ensemble des points formant les contours des obstacles précédemment déterminés comme ensemble S. On peut alors se poser la question de l'intérêt d'étudier l'impact de ces points sur l'espace.

Le diagramme résultant de cette décomposition est présenté sur la figure 5.5. Les contours des zones restreintes sont représentés par les lignes, et les zones d'influence —ou régions— de Voronoï sont représentées par les polygones colorés. Comme on peut le voir sur la partie zoomée, chaque région d'influence R_p correspond a son point associé p.

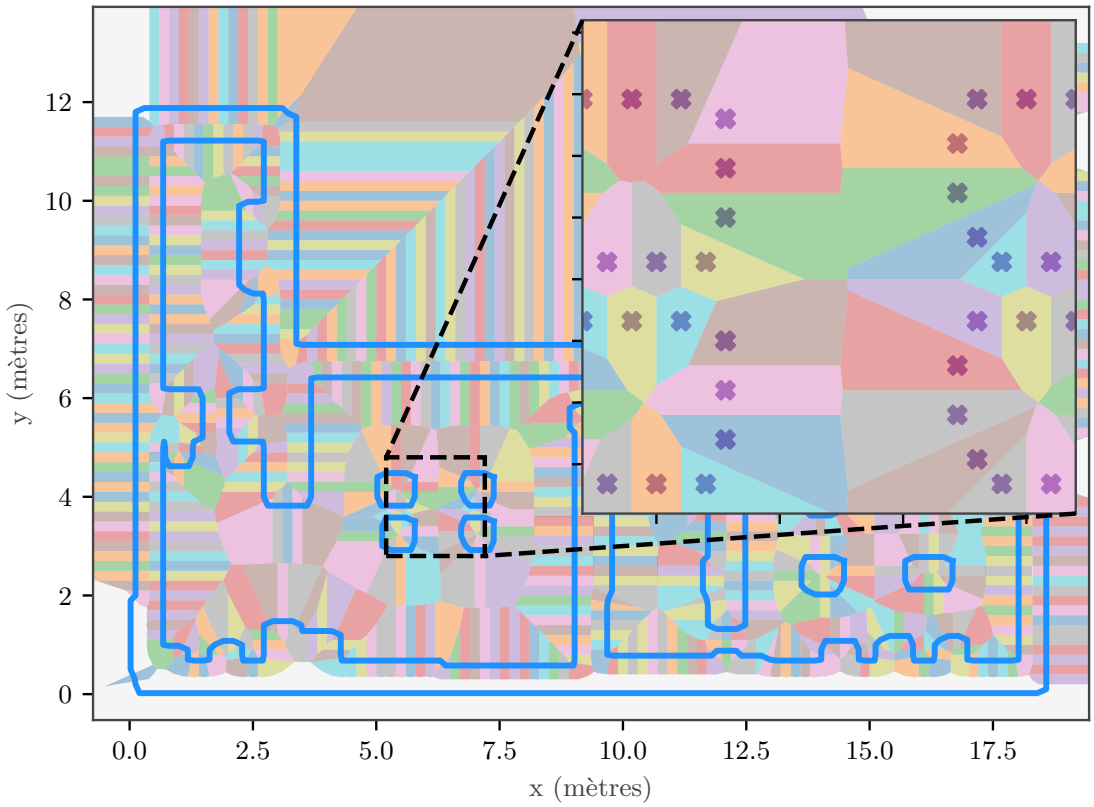

FIGURE 5.5 – Diagramme de Voronoï brut obtenu, avec zoom

Une application possible serait de connaître facilement, en fonction de la région dans laquelle un agent se trouve, quel est l'obstacle le plus proche. Mais ce n'est pas cette propriété qui nous intéresse ici. Par construction, la frontière entre deux régions R_a et R_b se situe sur la médiatrice des points a et b. Considérons donc les frontières comme des briques, des segments qui une fois adroitement assemblés, formeront des trajectoires dont chacun des points sera équidistant aux deux obstacles les plus proches de ce point. Ces briques sont représentées sur la figure 5.6, par les segments colorés. On constate que, si certains enchaînements de segments forment des trajectoires viables, de nombreux segments conduisent à des obstacles, voire les traversent, ce qui n'est évidemment pas un comportement désirable. Nous aborderons le processus de sélection des segments dans la section 5.5. Intéressons nous pour l'instant à la pertinence des segments valides.

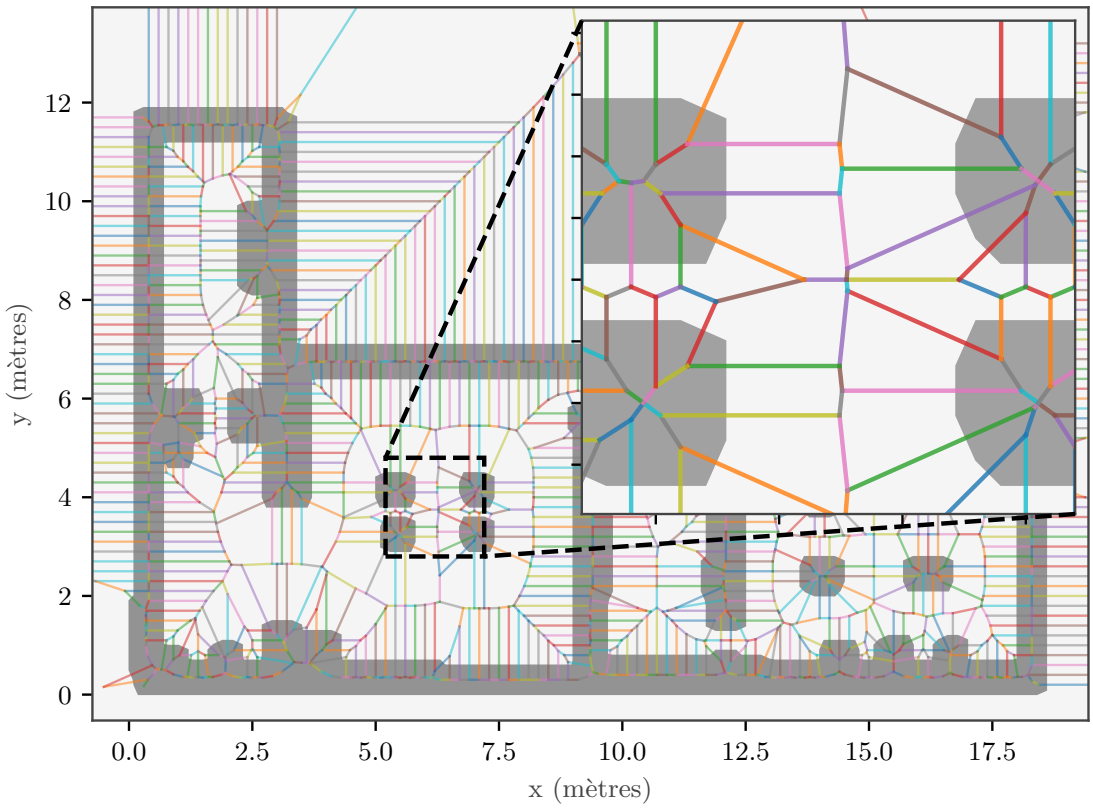

FIGURE 5.6 – Segments frontières des régions d'influences

Cette propriété de la trajectoire, d'être définie systématiquement au milieu de l'espace vide, a plusieurs conséquences. Première raison, historique, de l'emploi de diagrammes de Voronoï en robotique pour la génération des trajectoires : un segment équidistant aux deux obstacles les plus proches est également par définition un segment le plus éloigné de ces deux obstacles, qui permet toutefois de passer entre. C'est donc le meilleur candidat pour éviter de malencontreusement rencontrer l'un de ces deux obstacles.

Bien que cet évitement *naturel* des obstacles est un propriété qui nous intéresse, ce n'est pas la seule. Effectivement, se trouver au milieu de l'espace vide —libre d'obstacles— permet également de maximiser la couverture qu'a l'agent de la zone, qu'il s'agisse du champ de vision d'un opérateur humain, d'une caméra ou encore d'un Lidar.

5.5 Filtrage des frontières

L'ensemble des segments frontières obtenu précédemment comporte à la fois des segments valides —qui ne passent pas par des obstacles— et des segments invalides. Ces derniers sont dus à la discrétisation des obstacles opérée lors de l'utilisation des points constituants les contours pour générer le diagramme de Voronoï.

L'objectif de cette section est de présenter le travail de filtrage effectué. La nature des opérations nécessaires nous ont encouragé à étudier les capacités de diverses *bibliothèques* d'analyses spatiales :

Scikit-geometry Qui est un *wrapper* Python de la très complète librairie c++ CGAL [Fogel and Teillaud, 2015] . Le frein principal au choix de cette option réside dans la légèreté de sa documentation Python. Ainsi, dans le cadre de la réalisation d'une preuve de concept, il paraissait judicieux d'opter pour une autre solution.

Sympy-geometry Sympy est une bibliothèque orientée vers le calcul symbolique, et notre application est purement numérique, nous avons donc rapidement écarté cette option.

Shapely [**Gillies and Others,**] Toutes les opérations de *Shapely* utilisent une implémentation c++ à travers la bibliothèque Ouverte *GEOS*[18]. Nous avons choisi *Shapely* pour l'accessibilité des opérations principales dont nous avions besoin. Toutefois, quelques fonctionnalités comme la gestion de faisceau —demi-droite, ray en anglais— ne sont pas couvertes, nous forçant à contourner certains problèmes.

La logique de sélection des segments valides repose sur des opérations d'ensembles géométriques. Nous avons donc créé l'ensemble Z des polygones non réguliers à partir des contours c_k précédemment déterminés dans la section 5.3. De ce fait, chaque polygone $z \in Z$ représente géométriquement une zone restreinte. Pour filtrer les segments, il suffit donc de tester s'il existe une intersection avec n'importe lequel des polygones, et le considérer invalide le cas échéant. Soit S l'ensemble des segments et V l'ensemble des segments dit valides avec $V \subset S$, ainsi :

$$V = \{s \in S \mid \forall\ z \in Z,\ s \cap z = \varnothing\}$$

Plus concrètement, nous utiliserons les opérations ensemblistes offertes par la bibliothèque d'analyse spatiale sélectionnée. Il est important de préciser que le calcul de l'intersection de deux objets géométriques étant coûteuse, nous éviterons de l'employer autant que possible. Pour vérifier la validité d'un segment, il est donc plus avisé de tester l'existence d'une intersection plutôt que de calculer l'intersection et de vérifier quelle est nulle.

La figure 5.7 présente le résultat du filtrage obtenu en affichant uniquement l'ensemble V des segments valides. Nous avons maintenant un ensemble de briques de construction, qui reste cependant difficilement exploitable en l'état. En effet, il reste nécessaire de structurer intelligemment ces données pour pouvoir en tirer le maximum.

5.6 Structuration des segments

L'objet de cette section est dans un premier temps de présenter et d'expliquer au mieux les choix structurels effectués, puis, dans un second temps, d'en détailler la réalisation.

18. Geometry Engine - Open Source : https://trac.osgeo.org/geos/

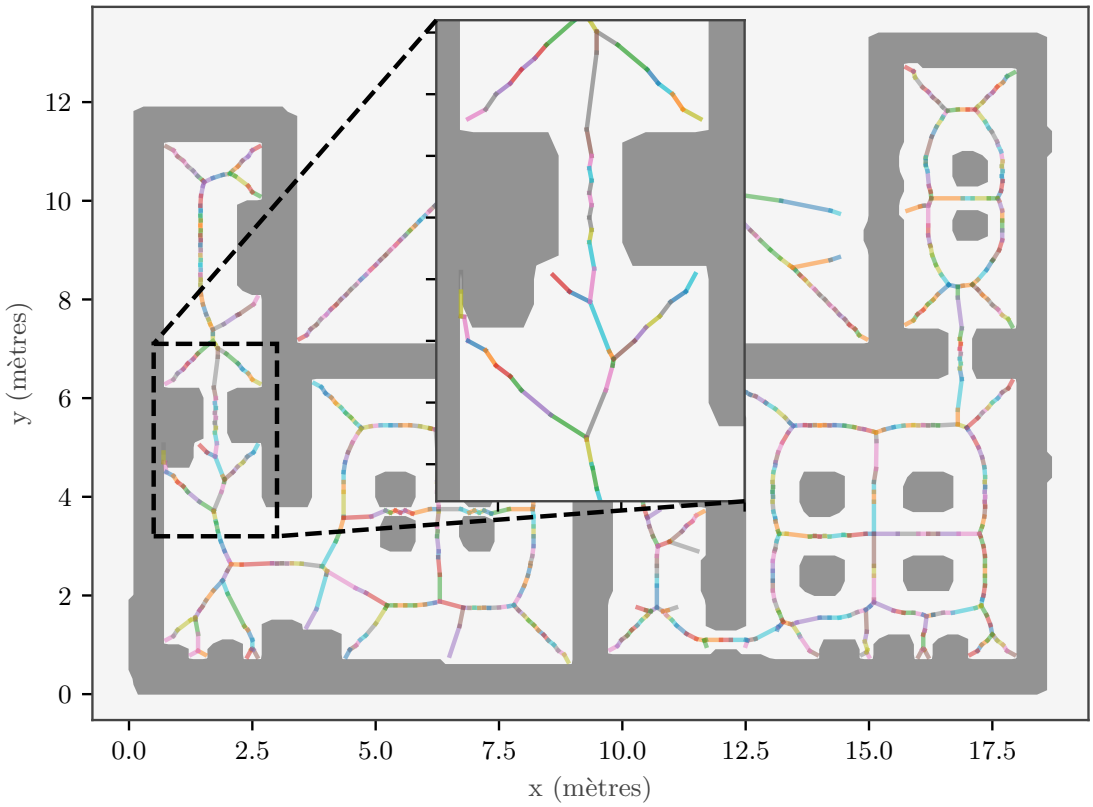

FIGURE 5.7 – Ensemble des segments frontières valides, avec zoom

Nous allons donc commencer par un constat. Nous avons précédemment généré un ensemble de points P correspondants aux extrémités des segments V, qui mis bouts-à-bouts formeraient un ensembles de trajectoires potentielles, vérifiant les propriétés de sûreté via le respect des zones restreintes, et de couverture via l'équidistance aux zones restreintes —les segments sont toujours au centre de l'espace libre—. Réfléchissons à ce que nous voulons faire de ces données. Une des contraintes de départ est d'être capable de générer une trajectoire à partir d'un point d'origine et d'une destination.

Afin de répondre à cet impératif, il faut dans un premier temps savoir quels sont les segments empruntables depuis le point de départ, puis après avoir parcouru un segment et être arrivé à son autre extrémité, quels sont les nouveaux segments empruntables et leur extrémité opposée associée, et ainsi de suite. C'est ainsi la notion de connexion entre les points, ou encore d'adjacence que nous abordons ici. Or, une structure de données excelle dans la représentation des relations de connexion entre éléments : le graphe. Un graphe est composé d'un ensemble de sommets, qui dans notre cas est l'ensemble P précédemment déterminé, ainsi que d'un ensemble d'arêtes, ici nos segments V. Le graphe est également muni d'une fonction d'incidence ϕ, associant à chaque arrête une paire de sommets correspondants aux extrémités de l'arrête.

Nous voulons pouvoir traverser les segments dans un sens où dans l'autre, et comme on peut le voir sur la figure 5.7, il peut y avoir des boucles, nous utiliserons alors un multigraphe non orienté avec boucles, défini par le triplet :

$$G = (P,\ V,\ \phi)$$

Dans ce graphe, chaque arête représente une relation d'adjacence entre ses extrémités. Ainsi, pour chaque arête a, b ses sommets extrêmes a et b sont adjacents, ce que nous noterons $a \sim b$. L'*ordre* d'un graphe $|P|$ est son nombre de sommets, et la *taille* d'un graphe $|V|$ est son nombre d'arêtes. Il existe deux manières d'implémenter un graphe en informatique, une matrice d'adjacence ou une liste d'adjacence. La première est particulièrement adaptée à un graphe de grande taille mais d'ordre faible, et inversement, la seconde est la plus efficace pour représenter un graphe d'ordre élevé mais de taille réduite. Dans notre situation, chaque sommet ayant un degré faible —c'est-à-dire un petit nombre de sommets adjacents—, nous aurons plus de sommets que d'arêtes. En effet, par construction via décomposition de Voronoï, le degré de chaque noeud est borné : $0 < \ deg(p) \leq \ n$ avec n raisonnablement petit : il n'est pas possible de garantir cette propriété —il peut y avoir des cas très particuliers où $n \approx 10^2$ — mais en pratique il est extrêmement rare d'avoir $n \geq 3$. Ainsi, le nombre d'arêtes (et donc la taille) du graphe sera limité. L'option liste d'adjacence est donc la meilleure voie.

La liste d'adjacence d'un multigraphe non orienté avec boucles est simplement la liste des points adjacents —où voisins— de chaque sommet. Ainsi, la liste d'adjacence G_a du sommet a est définie de la manière suivante :

$$G_a = \{p \in P \mid p \ \sim \ a\}$$

Nous savons maintenant comment organiser nos données, mais nous n'avons pas évoqué un point important et trop souvent absent des ouvrages et cours sur l'algorithmique : le peuplement des structures. En effet, il existe différentes manières de *remplir* notre liste d'adjacence, comme

c'est généralement le cas pour n'importe quelle structure de données. Deux approches sont présentées dans le code source 5.1. La première, consiste à parcourir tous les points de l'ensemble P, et pour chacun, trouver à quel segment il appartient —ce qui demande de parcourir tout l'ensemble S —pour enfin pouvoir ajouter une relation d'adjacence à p et à son voisin q. Le coût de cette opération est exorbitant : $\mathcal{O}(PV)$. La deuxième approche se contente quant à elle de parcourir une seule fois l'ensemble V, ce qui donne une complexité en $\mathcal{O}(V)$. Nous avons bien évidemment choisi la seconde approche. Fermons maintenant cette parenthèse sur le peuplement des structures de données pour revenir à nos problèmes de trajectoires.

```
# Parcours des points
for p in P:
    for v in V:
        if p in v:
            q = autre_extrémité(p, v)
            G[p] += [q]
            G[q] += [p]

# Parcours des segments
for v in V:
    p = v[0]
    q = v[1]
    G[p] += [q]
    G[q] += [p]
```

Code source 5.1 – Peuplements possibles de notre graphe

La figure 5.8 illustre le graphe obtenu. Le zoom effectué sur le coin en bas à droite de l'image permet de visualiser différents aspects du résultat obtenu. Les sommets du graphe sont figurés par les points bleus et les arêtes par les segments colorés. Le graphe nous a également permis de calculer le degré de chaque sommet, permettant de les catégoriser et d'obtenir l'ensemble C des carrefours :

$$C = \{p \in P \mid deg(G_p) \ > \ 2 \ \}$$

Il est également aisé de retrouver l'ensemble B des Culs-de-sacs (La lettre B est une référence à *Bag end*, demeure des *Sacquet*) :

$$B = \{p \in P \mid deg(G_p) \ = \ 1 \ \}$$

Nous avons donc structuré sous forme de graphe nos segments autrefois désorganisés, et pouvons maintenant calculer un chemin de parcours de l'environnement respectant nos contraintes. Cependant, cette structure comporte encore des inconvénients, problème auquel nous allons remédier dans la section suivante.

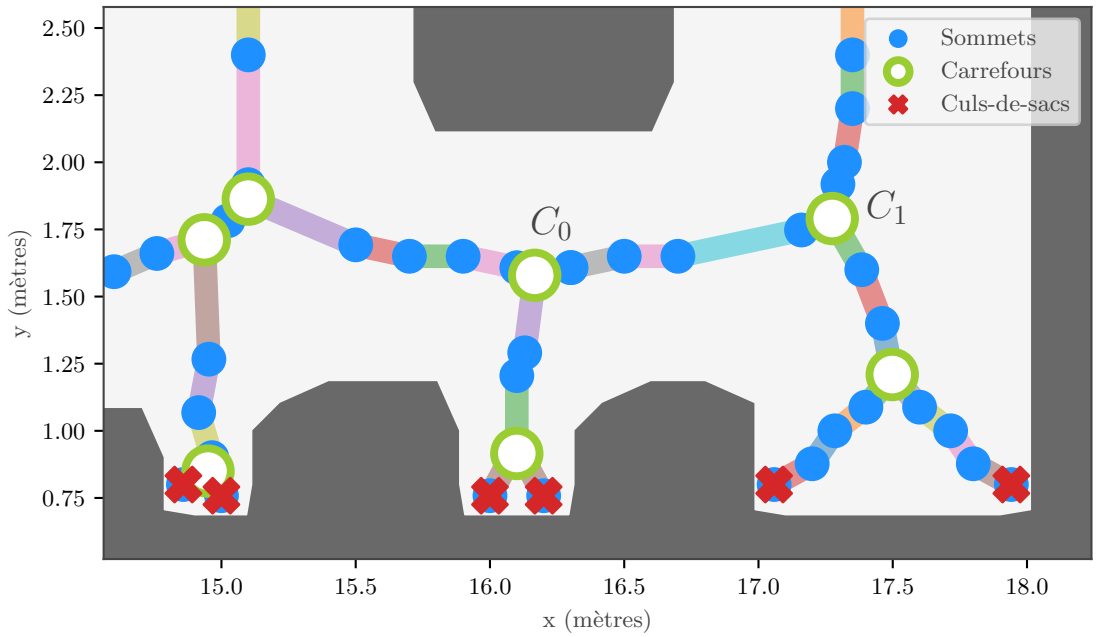

FIGURE 5.8 – Graphe brut obtenu, centré sur un coin de la carte initiale.

5.7 Organisation en allées

Le vocabulaire choisi pour représenter les objets que nous manipulons est emprunté à l'odonymie française —l'étude des noms propres désignant une voie de communication—. En effet, notre application consiste à déterminer des voies de communication entre des zones accessibles de l'environnement, alors pourquoi ne pas utiliser l'existant et s'inspirer de nos noms de rues. Le graphe G obtenu à la section précédente nous permet déjà de naviguer dans l'environnement, mais pas de manière optimisée. En effet, lorsque l'on emprunte un passage entre deux carrefours, admettons entre C_0 et C_1 de la figure 5.8, il n'y a plus de réelle décision à prendre avant d'arriver à C_1. Imaginez un instant votre GPS vous intimer toutes les minutes `Continuez tout droit` alors que vous êtes sur l'autoroute. Ou encore le tramway énoncer régulièrement `Nous ne sommes pas à un arrêt`. Vous l'aurez compris, on souhaite n'avoir à prendre des décisions que lorsque cela s'avère strictement nécessaire.

Une solution simplificatrice possible serait de construire un nouveau graphe, où les sommets seraient les carrefours C et les arêtes la concaténation des segments joignant les carrefours. Dorénavant, nous appellerons cette suite de segments une **allée** et noterons A l'ensemble des allées. L'approche paraît séduisante, mais en ne considérant que les carrefours comme sommets, nous perdons la capacité de nous rendre aux extrémités des impasses appelés culs-de-sacs. Il faut donc intégrer ces culs-de-sacs aux sommets, en créant un nouvel ensemble E appelé extrémités, défini de la manière suivante :

$$E = C \cup B$$

Cette fois-ci, la fonction d'incidence ϕ_2 associera à chaque arête, en plus d'une paire de sommets, un identifiant d'allée, permettant de retrouver ultérieurement les propriétés géométriques de l'allée correspondante. Notre nouveau graphe `raffiné` GR sera donc défini par le triplet suivant :

$$GR = (E,\ A,\ \phi_2)$$

Le ratio entre l'ordre $|E|$ et la taille $|A|$ du graphe va évoluer, mais pas suffisamment pour considérer l'utilisation d'une matrice d'adjacence. La liste d'adjacence reste la meilleure solution d'après les métriques disponibles. Pour prendre en compte l'évolution de ϕ_2, nous allons devoir stocker plus d'informations dans chaque liste d'adjacence. Ainsi, la liste d'adjacence GR_a du sommet a sera définie par :

$$GR_a = \{(p,\ id_{ap}) \in P \mid p\ \sim\ a\}$$

Vient maintenant la phase de peuplement de GR, notre graphe nouvellement défini. L'enjeu de cette étape est double : il faut d'un coté déterminer le graphe, et de l'autre construire les allées une à une. Pour réaliser cette étape délicate, nous allons nous appuyer sur le graphe G déjà disponible. Il s'agit alors ici de parcourir G, en étiquetant les points E avec les allées dont ils sont membres tout en créant les allées au fur et à mesure de l'exploration du graphe.

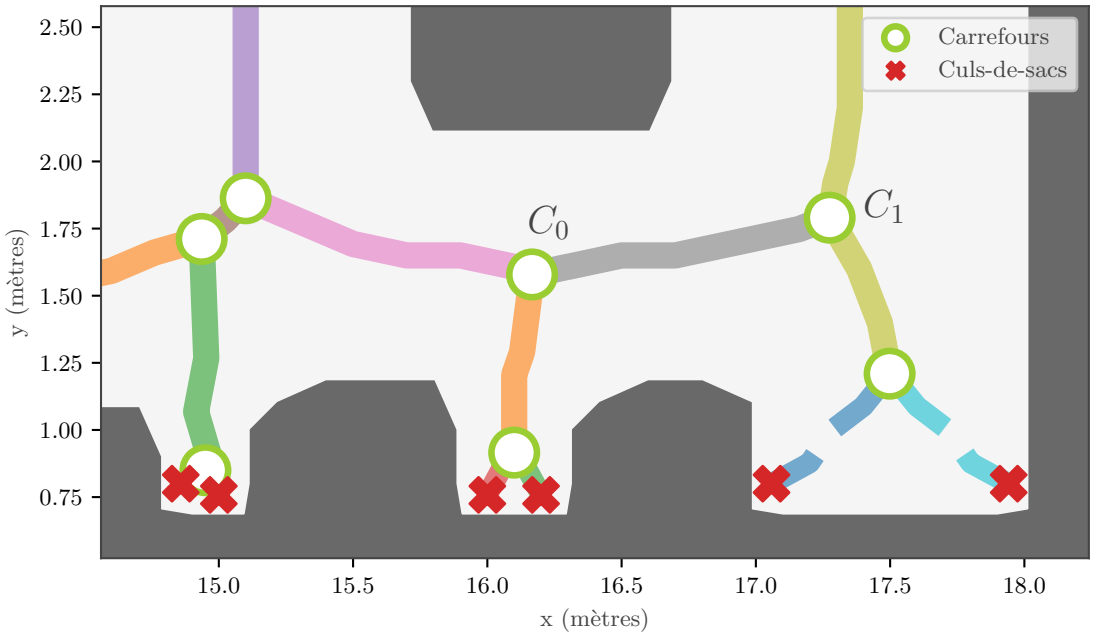

FIGURE 5.9 – Graphe d'allées obtenu à la suite de l'opération d'étiquetage

Le résultat de cette opération complexe d'étiquetage, le graphe GR et l'ensemble A des allées et de leur définition géométrique nous permet de générer la figure 5.9 sur laquelle on peut constater que les géométries des allées correspondent bien à la concaténation de leurs segments, et qu'elles permettent bien de relier deux extrémités. Les impasses sont représentées en pointillés pour illustrer notre capacité à les détecter facilement —en un seul test avec cette nouvelle structure—. Nous avons maintenant tous les éléments pour trouver efficacement un chemin dans l'environnement, dans le respect des contraintes que nous nous sommes imposées au début de ce chapitre.

5.8 Plus court chemin

La recherche du plus court chemin dans un graphe est un problème largement abordé et il existe de nombreux algorithmes permettant de trouver une solution optimale —au sens de la minimisation du coût—. Ces algorithmes sont souvent adaptés à un type spécifique de graphe et ne peuvent s'appliquer à d'autres types. Il est donc primordial de bien identifier de quel type de graphe l'on dispose. Nous avions évoqué la nature **multigraphe non orienté avec boucles** de notre premier graphe G. Lors de la conception de notre second graphe GR, nous lui avons adjoint un identifiant d'allée. bien que cet identifiant ne puisse être utilisé en l'état comme un poids, il permet de récupérer les données géométriques de l'allée concernée, et donc sa longueur. Or la longueur est précisément une estimation proportionnelle du coût réel que représente la traversée d'une allée. Nous avons donc maintenant un **multigraphe à pondération non-négative, non-orienté avec boucles**.

Comme évoqué précédemment, il existe de nombreux algorithmes de recherche du plus court chemin. D'après la taxinomie présentée en [Madkour et al., 2017], notre cas spécifique est dit statique *simple source*. Statique car la carte n'évolue pas pendant le calcul de chemin, et *simple source* par opposition à *toutes les paires*. En effet, seuls les chemins depuis la position actuelle de l'agent principal nous intéressent. Nous utiliserons l'acronyme $SSSP$ (*Single Source Shortest-Path*) pour définir notre problème dans la suite de cette section. La table 5.1 présente les caractéristiques des algorithmes considérés comme solutions potentielles à notre problème. Ces candidats produisent tous une solution optimale au problème posé, ils peuvent toutefois trouver des solutions différentes dans le cas où il existe plusieurs solutions optimales. Il s'agit donc maintenant de trouver un compromis entre efficience —complexité asymptotique— et simplicité de son implémentation. Définissons deux variables permettant d'exprimer plus clairement la complexité des algorithmes :

$$n = ordre(GR) = |E| \qquad \text{nombre d'extrémités dans l'exemple} = 114$$
$$m = taille(GR) = |A| \qquad \text{nombre d'allées dans l'exemple} = 122$$

Table 5.1 – Caractéristiques d'algorithmes simple-source statiques de plus court chemin

Nom	Auteurs	Complexité
Dijkstra	E. W. Dijkrstra [Dijkstra, 1959]	$\mathcal{O}(n^2)$
Fibonacci Heaps in $SSSP$	Fredman and Tarjan [Fredman and Tarjan, 1987]	$\mathcal{O}(n \log n + m)$
Thorup	Mikkel Thorup [Thorup, 1999]	$\mathcal{O}(m)$

L'option la plus simple, mais également la moins performante est celle de Dijkstra [Dijkstra, 1959]. Elle consiste en un parcours en largeur du graphe —dit de force-brute—, durant lequel on calcule la distance minimale entre l'extrémité actuelle et la source, en sauvegardant l'extrémité d'où l'on vient —précédente— dans le chemin minimal ainsi formé entre la source et l'extrémité actuelle. On pourrait tout-à-fait utiliser cette option en vue de la taille réduite de nos données (comparée à la taille que prendrait le même algorithme sur une image brute). Toutefois, cela limiterait fortement les capacités de passage à l'échelle de notre solution de planification, ce qui n'est pas souhaitable.

La seconde solution, proposée par Fredman et Tarjan [Fredman and Tarjan, 1987], consiste à utiliser un tas de Fibonacci —une implémentation optimisée de tas-minimum— afin d'améliorer la solution de Dijkstra en parcourant les solutions les plus prometteuses en premier. Un tas-minimum est une structure de données permettant de stocker des couples $(clé, priorité)$ permettant de récupérer efficacement la *clé* dont la priorité est minimale. Dans notre cas, cela revient à stocker des couples $(extrémité, d_{min}(source, extrémité))$. Ainsi, plutôt que de parcourir un nouveau sommet —une extrémité— arbitrairement parmi les sommets qui restent à parcourir, on choisit en premier celui dont la priorité —ici la distance entre l'extrémité et la source— est la plus faible. Cela permet d'éviter de parcourir les chemins plus coûteux —ici plus longs— dans le graphe, permettant selon l'instance de s'épargner de nombreux calculs. Cette proposition semble être un bon compromis entre efficience et simplicité.

Enfin, d'autres solutions plus complexes, comme celle proposée par Thorup [Thorup, 1999], reposent elles aussi sur le stockage des candidats à visiter dans une structure spécifique : un ensemble hiérarchisé par une arborescence stockant des paires de sommets. Cette approche est très intimement liée au modèle mémoire RAM —Random Acces Memory— et se démarque autant par son efficacité exceptionnelle ($\mathcal{O}(m)$) que par sa complexité de mise en oeuvre. C'est donc une option qui, bien que prometteuse, ne convient pas au cadre déjà fort limité en temps et en énergie du doctorat. Néanmoins, dans le cadre de l'exploitation industrielle des solutions présentées dans ce chapitre, explorer une telle solution pourrait être tout à fait bénéfique.

On l'aura compris, la solution construite sur l'utilisation de tas de Fibonacci est donc la solution retenue ici. Bien que optimale en théorie, cette structure de données ne dispose pas à l'heure actuelle d'une implémentation Python performante [19]. Ce qui n'est guère surprenant si l'on prend en compte les considérations de leurs inventeurs [Fredman et al., 1986] :

> Fibonacci heaps have two drawbacks : They are complicated to program, and they are not as efficient in practice as theorically less efficient forms of heaps, since in their simplest version they require storage and manipulation of four pointers per node, compared to the two or three pointers needed for other structures.
> — Fredman, Sedgewick, Sleator and Tarjan

> Les tas de Fibonacci ont deux inconvénients : Ils sont compliqués à programmer, et ils ne sont pas aussi efficaces dans la pratique que les formes de tas théoriquement moins efficaces, puisque dans leur version la plus simple, ils requièrent le stockage et la manipulation de quatre pointeurs par noeud, comparés aux deux ou trois pointeurs nécessaires pour les autres structures.
> — Fredman, Sedgewick, Sleator et Tarjan - *traduit de l'anglais*

Nous nous sommes donc rabattus sur un tas binaire —également une structure de type tas-minimum— dont une implémentation est intégrée à la bibliothèque standard python [20], dont les performances sont bien supérieures en pratique au tas de Fibonacci.

Une implémentation de l'algorithme choisi –Dijkstra avec tas-binaire– est donnée par le code 5.2. Trois structures de données sont utilisées :

dist Un tableau qui contient pour chaque sommet, la distance minimale entre ce sommet et la source.

19. Tas Fibonacci Python https://github.com/danielborowski/fibonacci-heap-python
20. Tas binaire Python https://docs.python.org/3/library/heapq.html

piste Un dictionnaire Python contenant, pour chaque sommet traversé pendant le parcours, un couple (*sommet*, *allée*). Ce dictionnaire permet de reconstruire a posteriori le chemin parcouru pour arriver jusqu'à un sommet en particulier depuis la source. On peut le voir comme la mémoire des chemins suivis, où encore comme la piste de cailloux astucieusement disposée le long du chemin par le *Petit Poucet*, protagoniste principal du conte homonyme de Charles Perrault.

tas Tas-binaire permettant de récupérer de manière efficace le meilleur sommet-candidat à parcourir. C'est donc cette structure qui va diriger la direction de l'exploration des solutions.

La première étape consiste à initialiser correctement ces trois structures. Chaque case du tableau *dist* de distances doit être initialisé à $+\infty$ —puisque l'objectif est de minimiser les distances— à l'exception de la case correspondant au sommet *source*, point de départ de l'algorithme, qui elle sera initialisée à 0. Le dictionnaire *prev* quant à lui sera initialisé à $\varnothing$. En effet, n'ayant pas encore commencé l'exploration des chemins, il n'y a pas de chemins à mémoriser. Enfin, le *tas*, qui va jouer le rôle de timonier lors de cette exploration des solutions doit être initialisé avec le seul sommet accessible pour l'instant : le sommet *source*.

Ensuite vient la phase d'exploration du graphe GR obtenu à l'issue de la section 5.7. Cette phase consiste à —tant qu'il reste des sommets-candidats à explorer— sélectionner le candidat le plus prometteur u, l'explorer et intégrer les informations nouvellement acquises dans nos trois structures. La sélection se fait de manière efficace par extraction du candidat dont la distance est minimale dans le *tas*. L'exploration consiste par la suite à, pour chaque voisin v du noeud sélectionné u, calculer la distance d à laquelle il se trouve de la *source* en passant par u, en utilisant pour cela la longueur de l'allée a qui va de u à v.

$$d = dist[u] + longueur(a)$$

Il s'agit maintenant de déterminer si l'exploration de chaque voisin v est pertinente, c'est-à-dire si la distance nouvellement calculée d est inférieure à la distance précédemment enregistrée pour le sommet v, stockée dans $dist[v]$. Le cas échéant, il convient d'intégrer ces nouvelles données, qui sont :

— Il est intéressant de visiter v ultérieurement, on ajoute donc v au *tas* avec d en guise de priorité.
— On sauvegarde la nouvelle distance : $dist[v] = d$
— On ajoute u dans la piste comme précédent de v, en prenant soin d'ajouter l'allée a empruntée pour passer de l'un à l'autre.

Après exécution de l'algorithme précédent sur nos données, nous disposons de deux structures de données consciencieusement remplies. La première, le tableau *dist* des distances de chacun des sommets à la source n'est pas nécessaire pour trouver le chemin optimal, mais ces informations restent précieuses, par exemple, lors de phases ultérieures de la planification consistant à décider quelle partie de l'environnement explorer par la suite. La figure 5.10 permet de visualiser les données de ce tableau, la couleur de chaque sommet représentant sa distance en mètres à la source. L'exploitation de la seconde sera détaillée dans la section suivante.

```python
# Utilisation du tas-binaire de la bibliothèque standard Python
from heapdict import heapdict

def dijkstra(allees, graphe, source):
    # Initialisation du tableau de distances
    dist = np.full(len(graphe), np.inf)
    dist[source] = 0

    # Initialisation de la piste
    piste = {}

    # Initialisation du tas-binaire
    tas = heapdict()
    tas[source] = dist[source]

    # Parcours tant que tas contient des sommets à explorer
    while tas:
        # On récupére le sommet u de distance minimale
        u, _ = tas.popitem()
        # Pour chaque voisin v de u
        for v, a in graphe[u]:
            # On calcule la distance entre la source et v
            d = dist[u] + longueur(a)
            # Si cette distance est inférieure à l'ancienne
            if d < dist[v]:
                # On met à jour pour ce sommet :
                tas[v] = d  #  On ajoute v aux sommets à visiter
                dist[v] = d  #  La distance minimale
                piste[v] = (u, a)  # Le précédent

    return dist, piste
```

Code source 5.2 – Implémentation de Dijkstra par tas binaire minimum

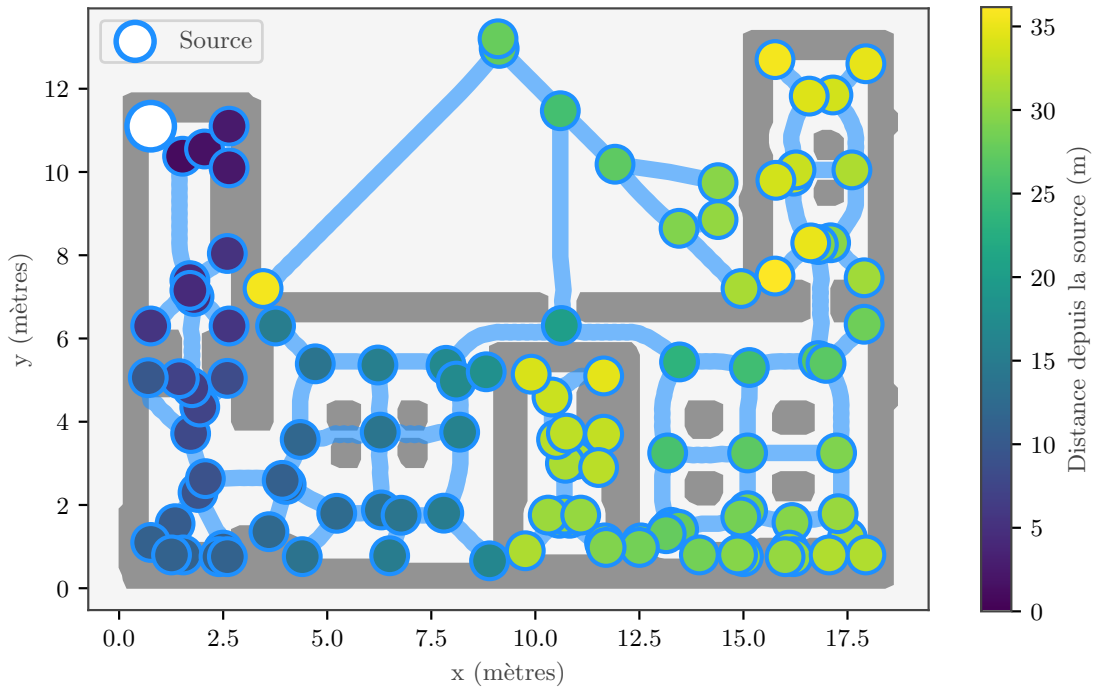

FIGURE 5.10 – Visualisation des distances minimales à la source

5.9 Construction de la solution

Comme vous l'aurez habilement remarqué, nous n'avons toujours pas généré de chemin optimal entre deux points, bien que nous disposions de tous les éléments pour le faire. Afin de construire ce chemin, il suffit de suivre à rebours la piste précédemment créée depuis le point *destination* qui nous intéresse. Ainsi, tant que le sommet actuel *sommet* n'est pas le sommet *source* on se rend au sommet précédent dans la piste, tout en prenant soin d'enregistrer le sommet actuel dans le chemin. On obtient alors le chemin *destination* $\rightarrow$ *source*. Il suffit alors de renverser l'ordre des éléments de ce chemin pour obtenir le chemin qui nous intéresse : *source* $\rightarrow$ *destination*. Le résultat de cette reconstruction est donc une suite ordonnée de couples $(sommet, allée)$ contenant le prochain *sommet* du chemin, ainsi que l'*allée* qu'il faut emprunter pour s'y rendre. Voici le chemin déterminé par la procédure 5.3 dans notre cas pratique :

$$chemin = \big[(111,\ 103),\ (47,\ 102),\ \ldots (18,\ 47),\ (35,\ 54)\big]$$

Nous avons maintenant un chemin composé d'identifiants, à la fois de points et d'allées. Il est néanmoins nécessaire d'y associer des informations géométriques, autant pour visualiser les données que pour plus tard, être capable de fournir une trajectoire réelle. On pourrait à priori penser qu'il suffit de substituer les identifiants par les données géométriques correspondantes. Toutefois, les allées étant des collections de points stockées avec une direction spécifique —bien qu'arpentable dans les deux sens— il convient de vérifier que la direction des données géométriques correspond à la direction que nous empruntons actuellement.

Prenons un exemple, notons $E_a \rightarrow E_b$ l'ensemble ordonné de points permettant de se rendre à l'extrémité E_b depuis l'extrémité E_a. Il est tout à fait possible que l'allée joignant E_g et E_h soit stockée dans le sens $E_h \rightarrow E_g$. C'est à dire d'un coté que le premier point de l'allée est E_h

```python
def rebrousse_chemin(destination, piste):
    # Récupération du couple (sommet, allée) initial
    sommet, allée = piste[destination]
    # Initialisation du chemin
    chemin = []
    while True:
        # Ajout du couple (sommet, allée) au chemin
        chemin += [(sommet, allée)]
        # MAJ du couple (sommet, allée)
        # → on prends le précédent du sommet dans la piste
        sommet, allée = piste[sommet]
        # Cas terminal, on ajoute le dernier sommet
        if sommet == source:
            chemin += [(sommet, allée)]
            # On retourne le chemin renversé
            return chemin[::-1]
```

Code source 5.3 – Reconstruction du chemin obtenu

et de l'autre que le dernier point de l'allée est E_g. Si l'on souhaite aller de E_a à E_h, et que le résultat de l'algorithme 5.3 est le suivant :

$$chemin = \left[(E_b,\ A_{ab}),\quad (E_c,\ A_{bc}),\quad \ldots,\quad (E_g,\ A_{fg}),\quad (E_h,\ A_{hg})\right]$$

En substituant simplement les identifiants d'allées A_{xy} par leurs données géométriques, on obtiendrait alors une liste de sous-chemins sans garantie d'avoir une direction identique pour l'ensemble des sous-chemins :

$$chemin\ géométrique = \left[\ (E_a \to E_b)\ ,\ (E_b \to E_c)\ ,\ \ldots,\ (E_f \to E_g)\ ,\ (E_g \leftarrow E_h)\ \right]$$

Afin de corriger si besoin la direction d'un sous-chemin, il suffit de vérifier si l'identifiant de l'extrémité à laquelle on souhaite se rendre correspond au dernier point de l'allée associée. Dans le cas contraire, il faudra inverser l'ordre des points de l'allée pour obtenir le résultat désiré.

$$chemin = \left[\ (E_b,\ A_{ab}),\ (E_c,\ A_{bc}),\quad \ldots,\ (E_g,\ A_{fg}),\ (E_h,\ A_{hg})\ \right]$$

$$chemin\ géométrique = \left[\ (E_a \to E_b)\ ,\ (E_b \to E_c)\ ,\quad \ldots,\ (E_f \to E_g)\ ,\ inverser(\ (E_g \leftarrow E_h)\ )\ \right]$$

Nous disposons maintenant d'une suite ordonnée de sous-chemins géométriques, informations permettant de générer la figure 5.11. Nous y superposons les données géométriques et les données de distances préalablement calculées. Un zoom permet d'apprécier le respect de la géométrie originale.

5.10 Ré-échantillonage des points

L'ensemble des sous-chemins construits précédemment permet d'ores et déjà de se déplacer de la *source* à la *destination*. Toutefois, un zoom sur les points constituant ces sous-chemins permet de mettre en avant le caractère éparse de ces points. La figure 5.12 permet de constater

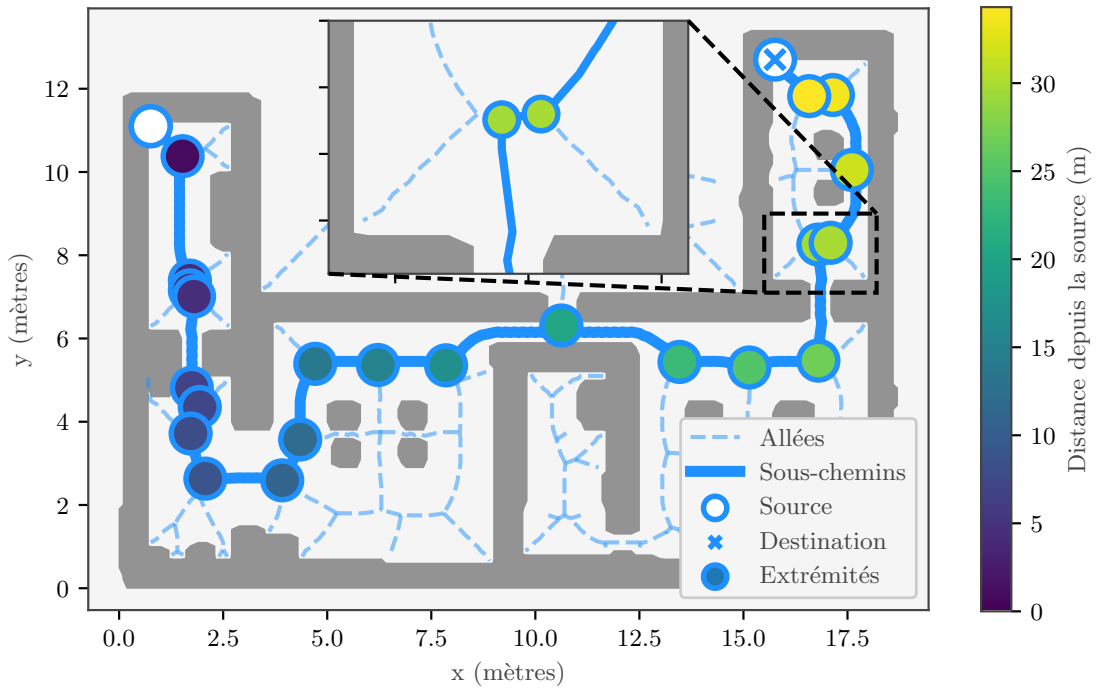

FIGURE 5.11 – Visualisation du chemin optimal reconstruit

le manque de régularité des points générés. En effet, leur concentration sur la trajectoire dépend —par construction— d'une part de la densité des contours des zones restreintes, calculés en section 5.3, et d'autre part de la proximité —et du positionnement plus généralement— de ces zones restreintes par rapport à la trajectoire. Aucun mécanisme ne garantit donc un espacement régulier des points sur la trajectoire. C'est ce mécanisme que nous allons mettre en place au sein de cette section, ayant pour objectif d'obtenir pour chaque sous-chemin C un ensemble de points P_c dont l'espacement est égal à une distance d :

$$P_c = \{(x_k, y_k) \in C \mid \forall\ k \in \mathbb{N} \setminus \{0\},\ \|(x_{k-1},\ y_{k-1}) - (x_k,\ y_k)\| \leq d\ \}$$

$$P_c = \{(x_k, y_k) \in C \mid \forall\ k \in \mathbb{N},\ \|(x_{k+1},\ y_{k+1}) - (x_k,\ y_k)\| \leq d\ \}$$

Il s'agit donc ici de déterminer un modèle de chaque sous-chemin, afin de pouvoir générer, à partir de ce modèle, un nouvel ensemble de points espacés de manière régulière le long de ce sous-chemin. Le modèle largement adopté par l'industrie du design aidé par ordinateur —mais également dans de nombreux secteurs où la manipulation de courbes, surfaces et volumes est nécessaire— est celui des B-Splines rationnelles non uniformes, communément nommées **NURBS** [Piegl, 1991]. Ce modèle NURBS est une génération des B-splines introduites par Schoenberg [Schoenberg, 1946] en 1946, étant elles mêmes une généralisation des courbes de Bézier. L'idée générale est d'utiliser une représentation paramétrique pour exprimer un ensemble géométrique tel une courbe ou une surface. Vous l'aurez compris, nous utiliserons donc ce modèle

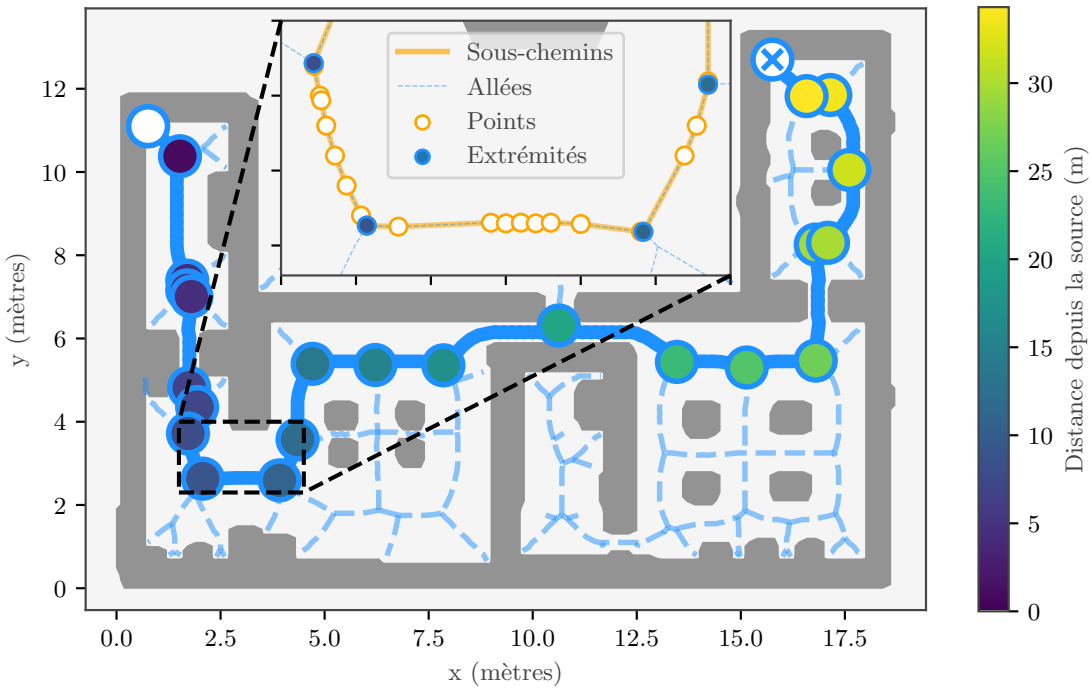

FIGURE 5.12 – Détails sur la trajectoire brute

pour représenter chacun de nos sous chemins.

De nombreuses implémentations de NURBS sont disponibles, mais parmi celles que nous avons évaluées, **NURBS-python** [Bingol and Krishnamurthy, 2019] a retenu notre attention. Cela pour plusieurs raisons :

— Bibliothèque libre et ouverte.
— Interface de programmation en Python.
— Fonctionnalité d'interpolation de courbe, pour obtenir un NURBS à partir d'un ensemble fini de points.
— Aides à la visualisation pensées pour la bibliothèque **Matplotlib** —déjà présentée précédemment— que nous utilisons pour nos rendus.
— Une documentation complète compréhensible.

L'opération d'interpolation, permettant d'obtenir une courbe est simple à mettre en œuvre en suivant la documentation [21]. Une fois cette représentation paramétrique de la courbe obtenue, il suffit de l'évaluer régulièrement afin d'obtenir l'ensemble P_c. Le résultat de cette opération appliquée à l'ensemble des chemins et représenté sur la figure 5.13. Une simple concaténation de ces ensembles de points permet alors d'obtenir une trajectoire composée de points régulièrement répartis, caractéristique simplifiant l'exploitation ultérieure de ces données.

5.11 Lissage de la trajectoire

La suite de sous-chemins géométriques calculée précédemment permet déjà de se rendre du point *source* au point *destination*. En revanche, construire directement une trajectoire avec ces points conduirait à un trajet avec des variations importantes de direction. Une analogie possible

21. Documentation https://nurbs-python.readthedocs.io/en/latest/module_fitting.html

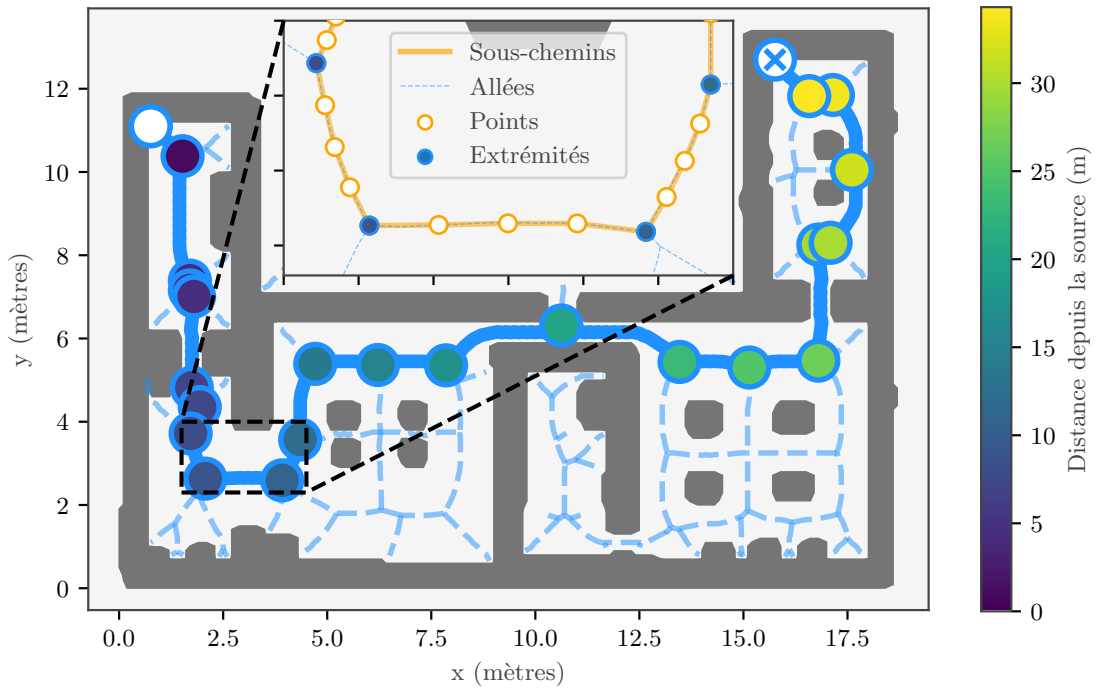

FIGURE 5.13 – Vue des sous-chemins ré-échantillonnés

est celle des lacets dans une route de montagne, virages tellement prononcés qu'ils demandent de freiner de manière conséquente afin de suivre le trajet, sans finir dans le décor. Une telle trajectoire constitue donc une solution sous-optimale pour traverser l'environnement.

Les routes de montagne ayant de fortes contraintes spatiales, il est difficile d'en modifier le tracé afin d'optimiser le temps de trajet. Dans notre situation en revanche, nous disposons d'une marge de manoeuvre suffisante pour limiter ces changement de direction, ayant pour conséquence directe une amélioration de la qualité de la trajectoire ainsi calculée dans le sens d'une optimisation de sa *traversabilité*. Cela revient pour ainsi dire à la rendre plus carrossable, et ce, que l'agent à qui cette trajectoire est destinée soit un humain où un robot.

Nous utiliserons toujours les NURBS dans cette section. Nous avons auparavant interpolé une courbe à partir de points. La courbe passait donc par ces points. Nous allons maintenant générer une courbe différemment, en utilisant des points de contrôle et un *vecteur nodal*. Chaque point de contrôle va influencer la courbe, en quelque sorte attirer la courbe vers lui. Le vecteur nodal détermine quant à lui quelle partie de la courbe est influencée par quel point de contrôle et avec quelle intensité. Cette construction produit une courbe —enfin une représentation paramétrique de celle-ci— qui comporte des propriétés intéressantes. Premièrement, la courbe obtenue ce situe dans l'enveloppe convexe des points de contrôle fournis. Cela se traduit concrètement par l'absence de dépassements du coté extérieur des virages, si l'on reprend la métaphore de la route. Seconde propriété, la courbe commence par le premier point de contrôle fourni et se termine par le dernier point de contrôle, a priori sans passer par les autres points de contrôle qui déterminent néanmoins l'allure de la courbe.

Ainsi, l'ensemble T des points de la trajectoire construit par concaténation des ensembles de

points P_c correspondant à chaque sous-chemin est un candidat parfait pour servir de points de contrôle. Ainsi, les points de départ et d'arrivée seront conservés, tout en relâchant les contraintes sur les points intermédiaires, permettant d'obtenir une courbe plus lisse —qui comprend moins de changements brusques de direction. Il reste maintenant à définir deux autres paramètres, le vecteur nodal et le degré de la représentation paramétrique.

Le vecteur nodal sépare l'espace paramétrique en sections, chaque section étant influencée par les mêmes points de contrôle, le nombre de points de contrôle utilisés dépendant du degré. Classiquement, l'espace est divisé en sections de longueurs égales. La bibliothèque NURBS-Python propose de générer automatique un vecteur nodal permettant une telle division de l'espace [22].

Le degré détermine le nombre de points de contrôle qui influencent simultanément une section de la courbe. Ainsi, augmenter le degré va intensifier le lissage de la courbe, au détriment du respect des points de contrôles —la courbe s'éloignera plus des points de contrôle—. Ses valeurs possibles sont bornées $2 < degré \leq n$ avec n le nombre de points de contrôle. Il est possible de faire varier directement ce paramètre afin d'ajuster l'intensité du lissage. Cependant, nous aimerions utiliser un paramètre qui porte plus de sens que le nombre de points utilisés pour déterminer le comportement de notre système. En utilisant intelligemment les données à notre disposition, nous pouvons calculer le degré en faisant varier un autre paramètre, plus pertinent à notre sens : la longueur l de trajectoire initiale à partir de laquelle puiser les points de contrôle. Soit L la longueur totale de la trajectoire initiale, et n le nombre de points de contrôle, on peut alors définir le *degré* de la manière suivante :

$$degré = \max\left(2, \quad \min\left(n, \quad \left\lceil \frac{l \times n}{L} \right\rceil\right)\right)$$

Où les fonctions *max* et *min* sont des gardes fous permettant de rester dans les bornes des valeurs possibles du degré. Ainsi, le choix du degré ne se base plus sur un nombre arbitraire de points, mais sur la longueur de trajectoire initiale influant sur chacune des portions de la courbe lissée.

Les trajectoires présentées dans les figures 5.14 5.15 sont le résultat de toutes les étapes présentées dans ce chapitre. les différences entre ces trajectoires sont quant à elles uniquement dues au changement d'intensité du lissage. Ainsi, nous pouvons adapter la génération de trajectoire aux spécificités de la mission, et plus particulièrement aux contraintes imposées par la réalité physique de l'agent qui va l'effectuer, comme son rayon de giration et sa capacité de changements d'allure. Lisser avec plus d'intensité la courbe n'est pas sans conséquences, en effet, plus on lisse la courbe, moins on respecte les propriétés attendues de la trajectoire. Comme on peut le constater sur les zooms des figures, lisser écarte la courbe des allées initiales, l'écartant également du centre de l'espace libre. Cela a plusieurs conséquences, premièrement, la couverture de cet espace libre à cartographier diminue. Deuxièmement, dans le cas d'un lissage de forte intensité, on peut se rapprocher dangereusement des zones restreintes. Il est toutefois aisé de détecter une intrusion dans ces zones —qui pourraient se traduire par une collision— et d'éliminer cette trajectoire. C'est le cas des trajectoires représentées en pointillés. Le choix de l'intensité du lissage est donc un compromis entre la simplicité d'en effectuer le suivi et le respect des contraintes initialement imposées. Il est toutefois aisé d'obtenir une trajectoire acceptable sur

22. Documentation https://nurbs-python.readthedocs.io/en/latest/module_knotvector.html

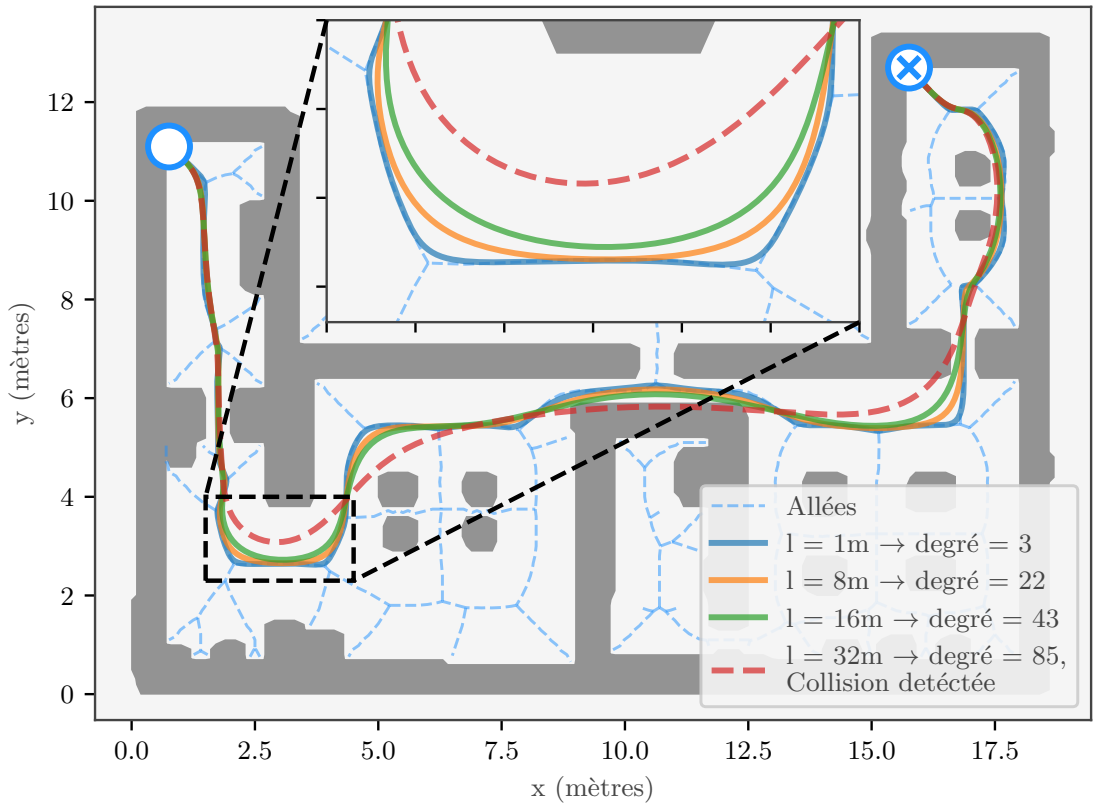

FIGURE 5.14 – Trajectoires obtenues en faisant varier l'intensité du lissage

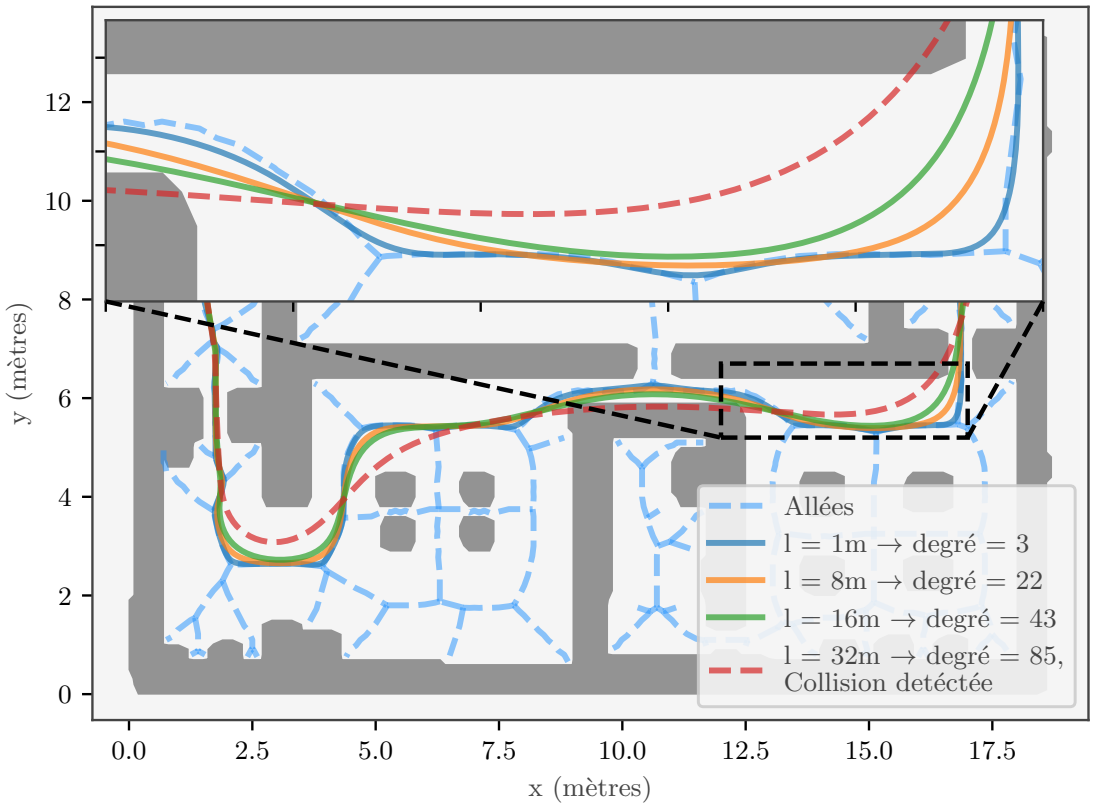

FIGURE 5.15 – Trajectoires obtenues en faisant varier l'intensité du lissage

ces deux contraintes, comme l'illustrent les trajectoires avec un lissage moins agressif.

5.12 Approche discrète

Dans l'approche précédente, pour obtenir des chemins éloignés des bords, le chemin était déduit d'un diagramme de Voronoï explicite (éléments géométriques).

Cependant, on peut également utiliser un processus entièrement basé sur des images numériques ou des structures équivalentes (tableaux 2D) pour le calcul de trajectoire. Pour cela, nous utilisons une information équivalente au diagramme de Voronoï, basée sur la distance aux murs et aux frontières inconnues, mais sans avoir recours à une construction géométrique explicite.

Le résultat attendu est similaire à celui de l'approche géométrique.

5.12.1 Données du problème pour le calcul de la trajectoire du robot

Les données du problème sont les suivantes :
— Une image numérique représentant la carte (2D) connue de l'environnement
— Le rayon du robot (en pixels)
— La position de départ du robot
— La position cible du robot

5.12.2 Processus de calcul du chemin

Dans un premier temps, il est nécessaire de calculer les distances minimales aux zones interdites pour chaque pixel libre (pixel vide) de la zone connue (déjà explorée). On construit donc un tableau de distances, de mêmes dimensions que l'image de la carte. Pour le calcul des distances minimales, on utilise un procédé classique de proche en proche en utilisant les 8 voisins (ensemble V_8) d'un pixel et en pondérant les déplacements aux voisins par leurs distances euclidiennes. Le principe consiste à parcourir la carte et pour chaque pixel, s'il est libre, on met à jour sa distance minimale à un obstacle en prenant le minimum des distances de ses voisins et le coût de déplacement entre ce pixel et son voisin :

$$d(p) = min_{v \in V_8}(d(v) + \|p - v\|)$$

On réitère le parcours de la carte tant qu'il y a des distances mises à jour. Le nombre d'itérations à effectuer dépend directement du pixel le plus éloigné de toutes les zones interdites. Cette étape est la plus longue du processus de calcul de chemin mais certaines optimisations peuvent être appliquées, notamment en ne mettant à jour que les pixels pertinents (par exemple en maintenant à jour une liste). Dans un contexte applicatif où la carte est totalement connue au départ, ce processus ne doit être exécuté qu'une seule fois. Dans un contexte d'exploration où des zones initialement inconnues sont découvertes au fur et à mesure, les distances sont mises à jour par propagation uniquement depuis les nouveaux murs découverts.

Le résultat obtenu est une approximation suffisante de la carte des distances aux obstacles, comme illustré en Figure 5.16. Cette carte représente de manière implicite une approximation



FIGURE 5.16 – Carte des distances aux obstacles.

du diagramme de Voronoï. En effet, les crêtes de la carte des distances correspondent aux arêtes du diagramme. L'avantage de cette représentation est qu'il n'est pas nécessaire de calculer explicitement le diagramme de Voronoï pour déduire des chemins qui l'empruntent entre deux positions données. En effet, on peut utiliser un algorithme en deux étapes :

Construction d'un chemin initial L'algorithme consiste à tracer progressivement le segment rectiligne entre le point de départ et le point d'arrivée. Lorsque l'on rencontre un pixel interdit (un obstacle ou un point trop proche d'un obstacle), un processus de contournement est enclenché. Le pixel interdit rencontré est le *point d'entrée* de cet obstacle. Le contournement consiste dans un premier temps à déterminer le contour de cet obstacle de proche en proche, puis à sélectionner le pixel de ce contour le plus proche de la destination finale, appelé *point de sortie*. Par construction, les obstacles forment un cycle. Il y a donc deux chemins longeant cet obstacle entre le *point d'entrée* et le *point de sortie* de l'obstacle. Le plus court de ces deux chemins est ajouté au chemin global. On reprend ensuite le tracé du segment rectiligne entre le départ et la destination. Ainsi, le chemin initial calculé à cette étape, représenté en jaune dans la figure 5.17 est donc composé de segments du tracé direct entre départ et destination et des contournements d'obstacles rencontrés.

Projection sur le diagramme de Voronoï implicite Pour obtenir le chemin final, on introduit la notion de point de crête maximal pour tout point libre de la carte. Ce point de crête est le point auquel on arrive en partant du point considéré et en suivant localement le gradient maximal dans la carte des distances. Ce point est donc un maximum local

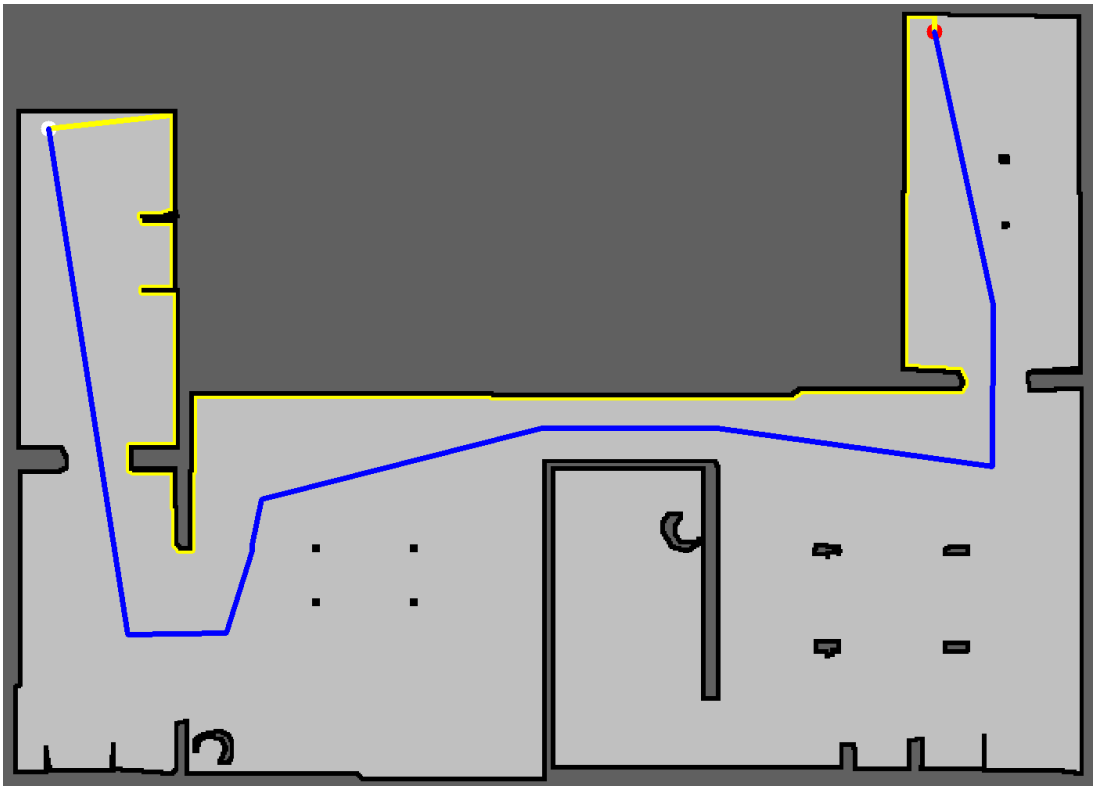

FIGURE 5.17 – Chemin initial (en jaune) et chemin final (en bleu).

dans la carte des distances. Ainsi, le chemin final est obtenu de la façon suivante :

1. Pour le premier point du chemin initial : Mémoriser le chemin pour aller de ce point au point de crête maximal correspondant.
2. Pour les points suivants du chemin initial : Déduire le point de crête maximal et l'ajouter au chemin déjà mémorisé s'il n'en fait pas déjà partie.
3. Pour le dernier point du chemin initial : Construire le chemin pour aller de ce point au point de crête maximal correspondant. Ajouter ce chemin renversé (pour respecter l'ordre du chemin) à la fin du chemin déjà mémorisé.

Un filtrage de simplification permet d'obtenir la trajectoire représentée en bleu dans la figure 5.17. Il consiste à prendre des raccourcis (aller en ligne droite) lorsque cela est possible. Le résultat est similaire à l'approche géométrique proposée précédemment, et il est tout à fait envisageable d'appliquer les méthodes de ré-échantillonnage et de lissage vues respectivement dans les sections 5.10 et 5.11. Nous avons donc mis en place deux approches, l'une géométrique et l'autre discrète, donnant des résultats similaires.

5.13 Constat et évolutions

Nous avons donc jusque là décrit et appliqué un ensemble de solutions algorithmiques permettant, à partir d'une image et d'un couple de points, d'obtenir une trajectoire respectant nos objectifs de départ. À savoir une couverture maximale de l'espace libre d'obstacle, pas d'intrusion dans les zones restreintes, et un trajet optimisé, au sens de la minimisation de la distance

parcourue et des variations d'allure, permettant une réalisation aisée de la mission programmée. L'étude qualitative du résultat obtenu est donc satisfaisante.

Toutefois, il convient de réaliser une analyse quantitative du processus de création de ce résultat. En effet, une rapide analyse du tableau 5.2 permet d'identifier des améliorations possibles. Ces données sont produites en utilisant le module python *Timeit*[23]. L'idée est d'exécuter n répétitions de m boucles, produisant ainsi $n \times m$ mesures du temps d'exécution d'un bloc de code. Seul le meilleur temps (minimum sur l'ensemble des mesures) est présenté, car d'après la documentation, seule cette valeur est pertinente, les variations de temps étant principalement dues à des paramètres extérieurs. Les caractéristiques techniques propres à la machine de test sont présentées dans le tableau 5.3. Il convient toutefois de préciser que la version actuelle de notre implémentation ne tire parti que d'un seul coeur de l'unité de calcul. Les performances pourraient donc êtres grandement augmentées car de nombreux algorithmes présentés précédemment sont hautement parallélisables.

Table 5.2 – Temps d'exécution minimaux observés

Label	Nombre d'exécutions	Meilleur temps	Pourcentage
Création zones restreintes	5×2000	130 μs	0.01
Géométrisation zones restreintes	5×200	2 ms	0.08
Calcul Voronoi	5×50	5 ms	0.21
Filtrage des segments	5×1	2 s	74.01
Peuplement graphe Brut	5×1000	227 μs	0.01
Peuplement graphe d'allées	5×500	742 μs	0.03
Plus Court chemin	5×200	1 ms	0.04
Construction des sous-chemins	5×2000	194 μs	0.01
Re-échantillonage des points	5×50	8 ms	0.30
Lissage	5×1	649 ms	25.31

Table 5.3 – Caractéristiques de la machine de test

Label	Valeur
Nombre de Coeurs Physiques	6
Nombre de Coeurs Logiques	12
Fréquence CPU minimale	3600 Mhz
Fréquence CPU maximale	2200 Mhz
Quantité de Mémoire Totale	16 GB

Une écrasante majorité des ressources nécessaires à l'exécution du pipeline présenté dans ce chapitre est concentrée sur le filtrage géométrique des segments, opération s'appuyant lourdement sur *Shapely*. Ainsi, migrer vers une librairie d'analyse spatiale plus performante –comme *scikit-geometry* évoquée dans la section 5.5– ou simplement structurer les calculs en prenant en compte les opérations internes à la bibliothèque nécessaires à la réalisation de notre filtrage, permettraient de réduire de manière significative le temps total nécessaire. Et par conséquent augmenter les capacités de passage à l'échelle de nos algorithmes —qui est linéaire avec la surface à analyser.

23. Documentation https://docs.python.org/fr/3/library/timeit.html

Autre possibilité, réaliser un lissage adaptatif de la courbe. En effet, le lissage est la seconde opération la plus gourmande en calculs. Or, comme on peut le constater sur la figure 5.14, seules quelques sections de la trajectoire nécessitent un lissage précis. Nous pourrions imaginer un algorithme léger déterminant les parties où appliquer le plus de calculs. Cette approche de calculs adaptatifs est notamment fortement utilisée dans les simulations de dynamique des fluides, bien que la tendance actuelle soit plus dans l'utilisation d'apprentissage profond pour résoudre ce genre de problématiques. L'utilisation de l'apprentissage automatique pour lisser la courbe pourrait également se révéler pertinent, mais il s'agit là d'un tout autre domaine de recherche.

L'approche discrète présente des temps d'exécution du même ordre de grandeur que ceux présentés dans cette section, validant sa pertinence en tant que solution algorithmique alternative. Globalement, les deux approches ont des performances similaires de l'ordre de la seconde sur les exemples traités avec une précision très élevée. On voit donc que nous ne sommes pas directement dans une application temps réel. Cependant, les temps de calcul étant proportionnels à la précision ciblée, une résolution inférieure permettrait de réduire considérablement les temps de calcul tout en conservant une précision suffisante pour la plupart des missions envisageables.

Chapitre 6

Calcul des placements des Phares

La section 5 décrit la génération d'une trajectoire à partir d'un modèle de l'environnent correspondant a une partie des besoins spécifiques de notre système. En effet, seuls les besoins de l'agent principal sont couverts. Il nous reste maintenant à déterminer une stratégie de placement des **phares**, de manière à optimiser —ici minimiser— le nombre de déplacements des phares nécessaire à la réalisation d'une mission donnée. Dans le cadre de ce chapitre, nous utiliserons la trajectoire de l'agent principal déterminée précédemment comme exemple de mission. Dans le cas d'une mission avec plusieurs agents autonomes (routes différentes), il serait nécessaire d'avoir au moins un phare par agent de mission. Ainsi, le problème revient à déterminer les positions de phares indépendemment pour chaque agent.

Un solution au problème global est une liste de couples $(placement, section)$ décrivant comment positionner le phare et quelle section de la trajectoire est couverte par le phare ainsi positionné :

$$Solution = [(placement_0,\ section_0)\ ,\ \ldots\ (placement_n,\ section_n)]$$

L'optimisation au niveau global consiste donc à minimiser le nombre n de couples nécessaires à la réalisation de la mission. Cependant, chaque étape est dépendante de la précédente, le point de départ de la $section_{k+1}$ étant le dernier point de la $section_k$.

Ce problème est équivalent au classique problème de la station essence, où l'objectif est de parcourir un trajet linéaire, en s'arrêtant le moins possible pour faire le plein d'essence. Dans cet exemple classique, la meilleure solution, gloutonne, consiste à toujours aller à la station la plus loin possible, dans les limites de notre stock de carburant. Il en est de même dans notre problème, où l'optimal global peut être obtenu en sélectionnant successivement les meilleures solutions locales. C'est-à-dire choisir le placement du phare qui permet de maximiser la section de trajectoire couverte par ce phare, puis boucler sur le reste de la trajectoire.

Il est possible d'utiliser plusieurs phares pour suivre un même agent. Dans ce cas, le placement revient à choisir le nombre correspondant de positions dans l'ensemble des solutions en respectant les différentes contraintes spatiales entre les phares.

Nous aborderons dans un premier temps la détermination de l'espace de solutions locales, qui est en réalité un ensemble de placements possibles du phare. Puis nous détaillerons le calcul de la pertinence de chacune de ces solutions —sorte de score, déterminé analytiquement— per-

mettant la sélection du meilleur placement et de la section qu'il permet de couvrir. Enfin, nous pourrons construire et analyser les solutions globales produites par ce processus.

6.1 Modélisation du problème local

Afin de mieux appréhender le problème à résoudre, il convient de modéliser la **zone couverte** par un phare. Le phare émettant un laser dans l'environnement, la zone couverte est donc définie par divers éléments :

portée p Distance en mètres à laquelle les récepteurs sont encore capables de capter et d'exploiter les informations encodées dans les lasers.

angle θ Angle délimitant le champ de vision, exprimé en degrés où en radians.

Ainsi, la zone de couverture d'un phare est un secteur circulaire, défini par les inéquations suivantes, dans un repère centré sur le phare :

$$x^2 + y^2 \leq p^2$$
$$-\tan\left(\tfrac{\theta}{2}\right) \leq \frac{x}{y} \leq \tan\left(\tfrac{\theta}{2}\right)$$
$$y \geq 0$$

La figure 6.1 présente le modèle d'un phare, ou plutôt trois déclinaisons de ce modèle, avec une portée identique $p = 5m$ mais avec différentes valeurs de l'angle d'ouverture θ. Lors d'une exploration typique, la trajectoire est majoritairement rectiligne, les variations de direction —les virages— n'étant que des cas particuliers entre deux sections rectilignes. En faisant l'hypothèse d'une trajectoire rectiligne, il devient alors aisé de déterminer comment placer le phare par rapport à la trajectoire. Ainsi, lorsque l'angle d'ouverture est petit : $\theta < 60°$, le plus grand segment inclus dans la zone de couverture est n'importe lequel des rayons du secteur circulaire, de longueur p, c'est le cas de l'exemple représenté avec $\theta = 40°$. Dans le cas où $\theta = 60°$, la corde c et les deux rayons extrêmes délimitant le secteur circulaire forment un triangle équilatéral, donnant donc $c = p$. Enfin, dans le dernier cas, où $\theta > 60°$ le plus grand segment inclus dans le secteur devient la corde c. Sa longueur est calculable par la formule suivante :

$$c = 2p \times \sin\left(\tfrac{\theta}{2}\right)$$

Ainsi, avec les paramètres spécifiques du système physique que nous utilisons, c atteint plus de $9.5m$, alors que la portée du phare n'est que de $5m$! Placer le phare de manière à superposer la corde c du secteur circulaire à la trajectoire permet donc de maximiser la longueur de trajectoire rectiligne couverte par le phare. Et donc, par la même occasion, de minimiser le nombre de déplacements de phare nécessaires à la bonne conduite d'une mission. Afin d'effectuer cette superposition, il convient de placer le phare —orienté perpendiculairement à la direction de la trajectoire— à une distance d de cette dernière, calculée de la manière suivante :

$$d = p \times \cos\left(\tfrac{\theta}{2}\right)$$

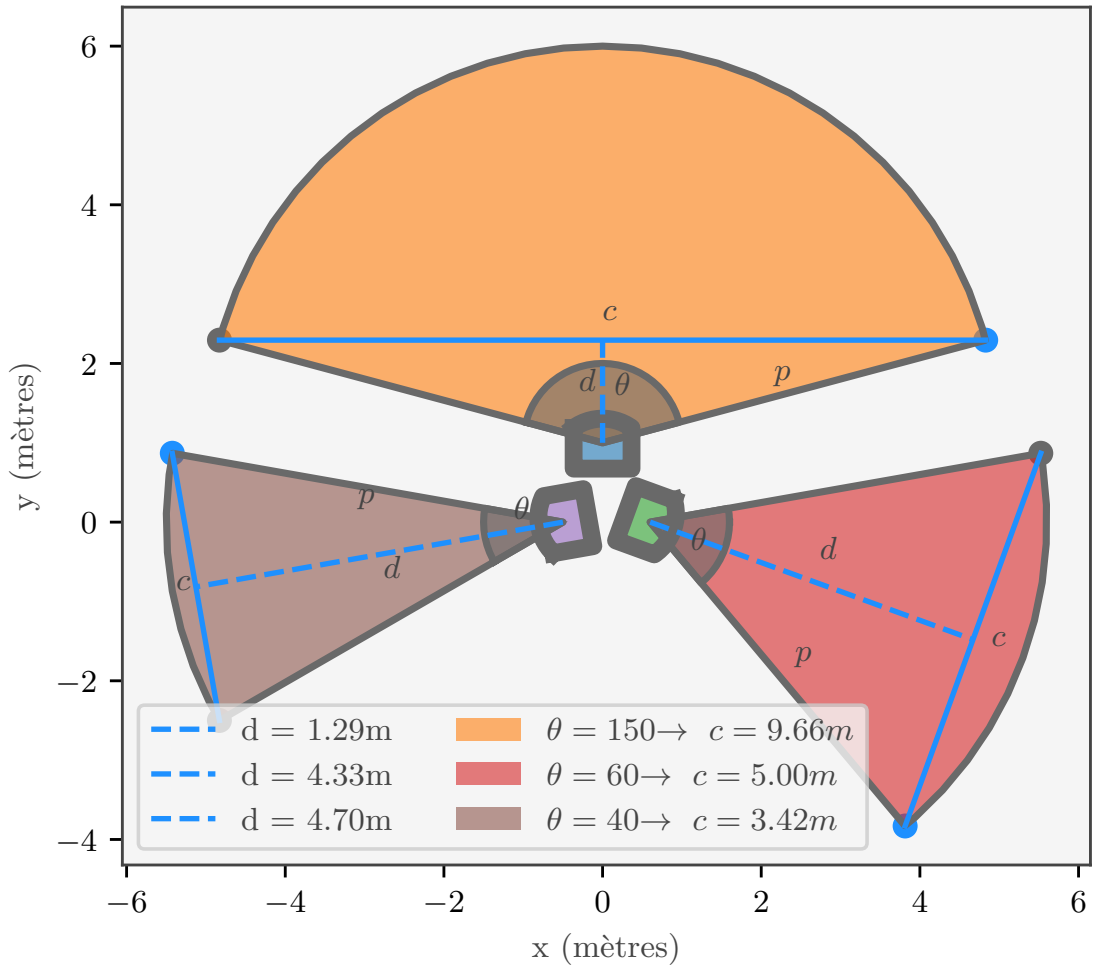

FIGURE 6.1 – Modèles représentant la zone couverte par un phare, θ variant.

Nous savons maintenant comment placer notre phare par rapport à une trajectoire rectiligne. Cependant, même en dehors des virages, les trajectoires telles que nous les construisons sont rarement parfaitement rectilignes, mais plutôt contenues dans une bande réduite, dont la largeur varie selon le scénario et l'intensité du lissage. Ainsi, modéliser la trajectoire par une bande permet de mieux prendre en compte les réalités du terrain. La figure 6.2 représente cette bande, inscrite dans le secteur circulaire. Elle est définie entre une bordure intérieure, la corde c (à distance d du phare) et une bordure extérieure, la corde c' (à une distance $d' = d+b$ du phare).

Augmenter la largeur b de la bande aura pour conséquence de diminuer la longueur maximale de bande couverte, et favorisera la capacité à mieux couvrir les cas où la trajectoire ne serait pas totalement rectiligne, ainsi que les virages bruts de la trajectoire. Cette capacité de tolérance est proportionnelle à l'aire de la bande, qui est définie de la manière suivante :

$$A_{bande} = \underbrace{\frac{p}{2} \times \theta}_{A_{secteur}} - \left(\underbrace{\frac{d \times c}{2}}_{A_{triangle}} + \underbrace{\frac{p^2}{2}(2\alpha - \sin(2\alpha))}_{A_{segment}} \right), \; avec \; \alpha = \cos^{-1}\left(\frac{d'}{p}\right)$$

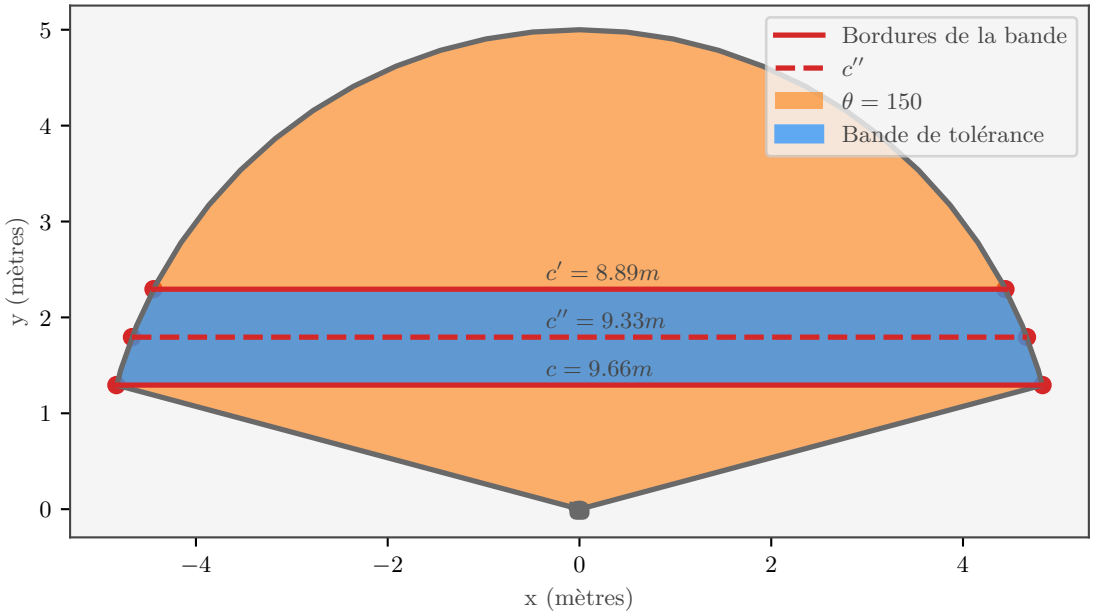

FIGURE 6.2 – Détermination de la zone de tolérance

Le choix de la largeur de bande b est donc un compromis entre longueur maximale couverte et aire maximale couverte. L'évaluation de ces grandeurs en fonction de la largeur de bande est présentée sur la figure 6.3, qui propose un compromis déterminé par l'intersection des courbes *distance* (décroissante) et *aire* (croissante) exprimées dans la même échelle. Ce choix est bien évidemment arbitraire, mais il représente la meilleure option sans avoir plus de connaissances sur les propriétés de la trajectoire qui est elle même déterminée à partir de l'environnement et de la mission choisie.

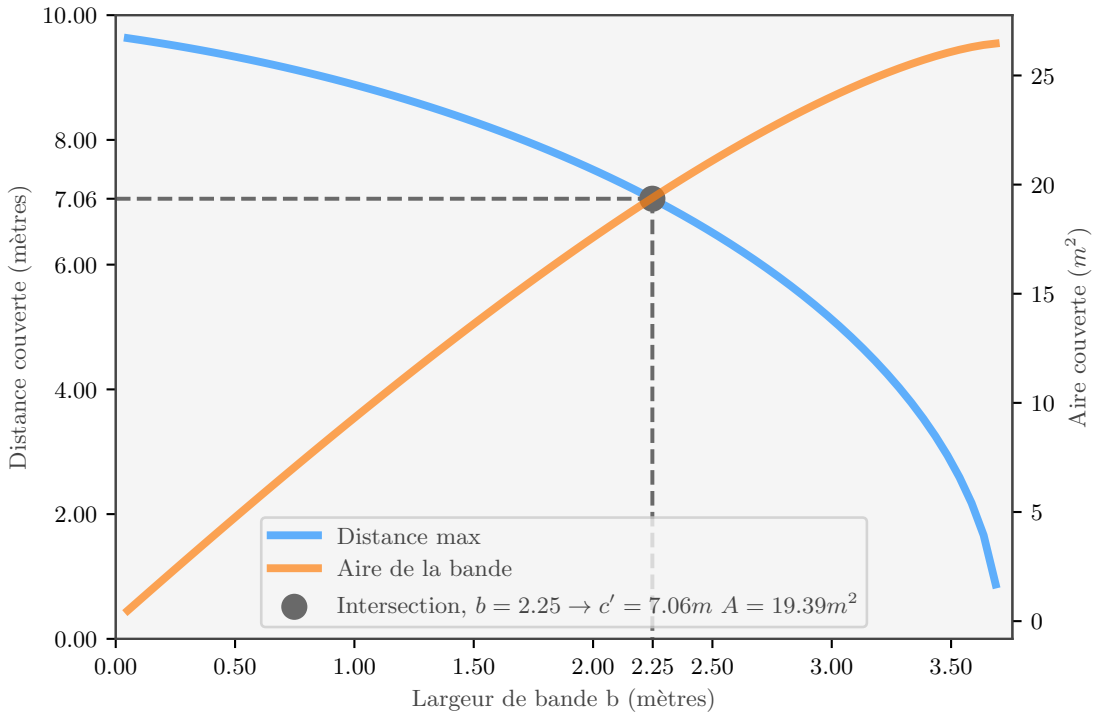

FIGURE 6.3 – Comparaison des largeurs de bandes

Les modèles choisis sont donc pour l'un exact —Un secteur circulaire représente exactement la zone couverte par un phare— et de l'autre une approximation —une bande est une estimation de l'enveloppe dans laquelle se situe une trajectoire. Nous avons ainsi déterminé une heuristique consistant à placer le phare orienté en direction de la trajectoire à une distance constante. Cette distance est déterminée en additionnant la distance d définie précédemment entre le phare et la bordure intérieure et la moitié de la largeur de bande. Ainsi :

$$d_{phare} = \underbrace{p \times \cos\left(\frac{\theta}{2}\right)}_{distance\ d} + \frac{b}{2}$$

Le calcul numérique respectant les caractéristiques intrinsèques du système physique utilisé ainsi que le compromis accepté sur la largeur de bande donne une distance $d_{phare} = 2.418m$. Il convient maintenant d'évaluer la pertinence de cette heuristique à travers divers exemples réels, ce qui fera l'objet des sections suivantes.

6.2 Cas idéal

D'après la modélisation proposée dans la section précédente, on peut définir une nouvelle trajectoire de chaque coté de la trajectoire principale déterminée dans la section 5, que nous nommerons **sentiers**. Chaque sentier est une trajectoire parallèle à la trajectoire principale, décalée d'une distance constante vers la gauche ou vers la droite. Ainsi, un sentier est l'ensemble des solutions possibles à notre problème d'optimisation du placement des phares respectant l'heuristique déterminée dans la section 6.1.

Déterminer un sentier parallèle à la trajectoire principale nécessite quelques calculs géométriques, majoritairement situés dans la détermination des virages intérieurs de la courbe. Heureusement, la bibliothèque *Shapely* propose une fonction de commodité permettant de réaliser automatiquement ces calculs.

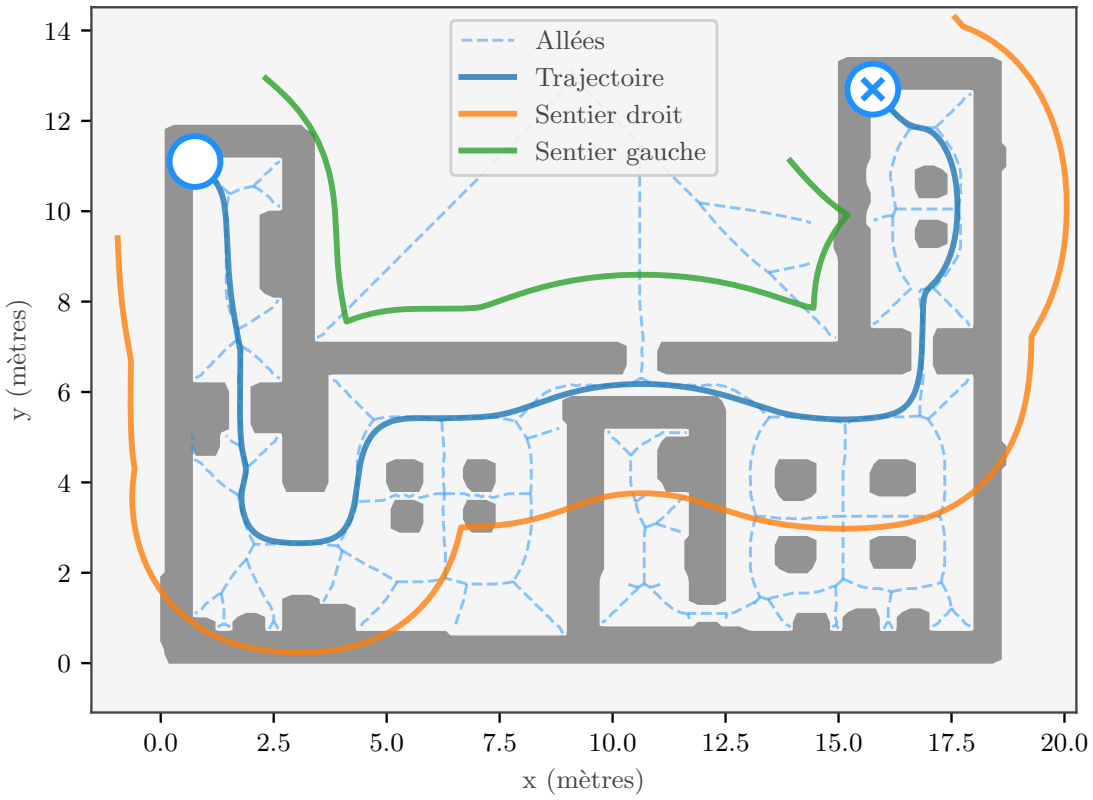

FIGURE 6.4 – Sentiers idéaux, représentés dans l'environnement

Il est alors aisé de constater sur la figure 6.4 que ces sentiers idéaux ne sont pas utilisables en l'état. En effet, non seulement ils traversent des zones restreintes, rendant caduques les solutions dans ces zones, mais on observe également qu'aucune garantie n'est fournie quant à la visibilité du sentier depuis la trajectoire et inversement. Or le maintient de la visibilité entre le phare et l'agent principal —le récepteur— est absolument nécessaire au bon fonctionnement du système de positionnement pour lequel nous essayons de déterminer le comportement. Ces deux problèmes ont une racine commune, la non prise en compte des obstacles dans la détermination des sentiers. Afin de concevoir des sentiers empruntables, il faudra donc modéliser les obstacles et vérifier l'absence de collisions ainsi que le respect de la ligne de vue. Dans les cas où le sentier idéal ne conviendrait pas, il faudra également construire une nouvelle solution à partir des éléments à notre disposition.

6.3 Sentiers adaptés à l'environnement

Afin d'adapter nos sentiers à l'environnement, il convient de savoir comment se concrétise l'environnement. En d'autres termes, quel modèle de l'environnement nous utilisons. Nous avions

introduit la représentation des zones restreintes par des polygones dans la section 5.5, nous allons donc réutiliser ce modèle, Z étant l'ensemble des polygones z.

Pour vérifier les propriétés de visibilité et d'absence de collisions, le plus simple serait d'utiliser un algorithme de *moulage par rayons* —Ray casting en anglais. Cela consisterait en la construction d'un rayon partant de la trajectoire, et suivant la normale de chacun des points de cette trajectoire. Malheureusement, la bibliothèque *shapely* ne propose ni détermination des normales, ni algorithme de moulage par rayons.

Nous allons donc simuler un algorithme de moulage, en utilisant uniquement les opérations ensemblistes à notre disposition. Nous allons construire, pour chaque couple (t_i, t_{i+1}) de points consécutifs de la trajectoire une médiatrice M. Les moitiés gauche M_g et droite M_d de cette médiatrice jouerons le rôle de rayons. Les zones restreintes ainsi que les sentiers idéaux S_g (gauche) et S_d (droit) déterminés précédemment joueront quant à eux le rôle de limites sur lesquelles les rayons s'arrêteront.

Nous ne présenterons que la génération du sentier réel droit, la détermination du côté gauche étant en tous points similaire. L'idée générale de l'algorithme présenté ici est dans un premier temps de déterminer l'ensemble I_d des intersections entre la demi-médiatrice de droite et l'union des zones restreintes et du sentier idéal droit :

$$I_d = M_d \cap (Z \cup S_d)$$

Il s'agit ensuite de déterminer, le point $p \in I_d$ le plus proche de l'origine de la demi-médiatrice M_d. La figure 6.5 illustre notre algorithme avec deux exemples sur une trajectoire sous-échantillonnée, permettant ainsi de dessiner les rayons construits sans saturer l'image. L'ensemble I_d peut ne contenir qu'un seul point (la médiatrice ne traverse pas d'obstacles et atteint S_d) auquel cas ce point sera utilisé pour construire le sentier adapté. C'est le cas le plus simple, visible pour la demi-médiatrice droite entre t_{38} et t_{39}.

Cet ensemble peut également contenir un point et un ensemble de segments issus de l'intersection entre les zones restreintes et la demi-médiatrice. Il faudra alors sélectionner le point le plus proche de l'origine de la demi-médiatrice parmi l'ensemble des points inclus dans ces segments d'intersection. C'est le cas pour la demi-médiatrice droite des points t_{36} et t_{37}.

Si le résultat obtenu peut sembler éloigné de la réalité du terrain, c'est parce que la trajectoire à été fortement sous-échantillonnée pour produire une figure lisible. En effet, bien que sensible à la densité initiale de la trajectoire, l'algorithme produit des résultats satisfaisants dans sa version actuelle, comme l'illustre la figure 6.6. De plus, il est aisé de générer de nouveaux points de trajectoire par interpolation, permettant ainsi de compenser une éventuelle densité faible de la trajectoire principale, toutefois au prix d'une augmentation linéaire des calculs.

6.4 Réduction de l'espace de solutions

Il est possible et nécessaire de réduire le nombre de solutions à tester, afin de limiter le coût en calcul de notre solution. En effet, dans les ensembles G et D, il peut exister des points qui sont trop éloignés du début de la trajectoire à suivre à l'étape k —que nous nommerons o_k pour

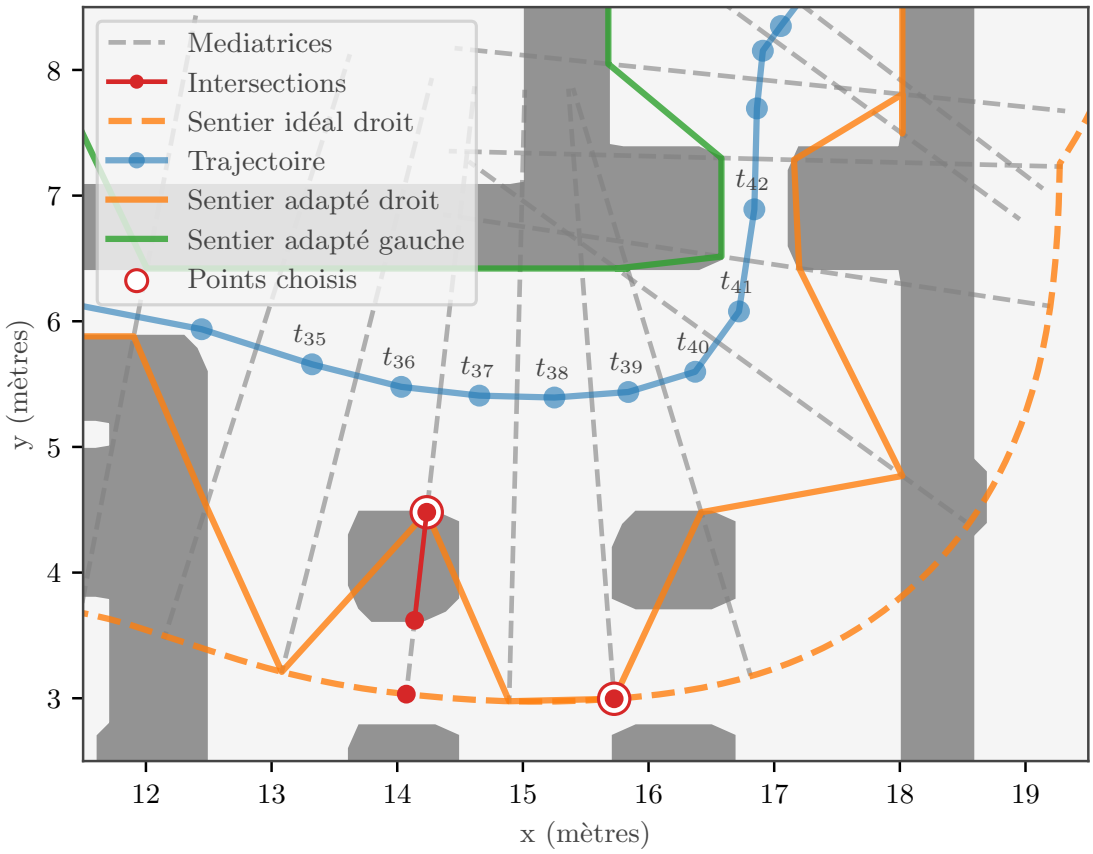

FIGURE 6.5 – Sentiers adaptés à l'environnement, trajectoire sous-échantillonnée

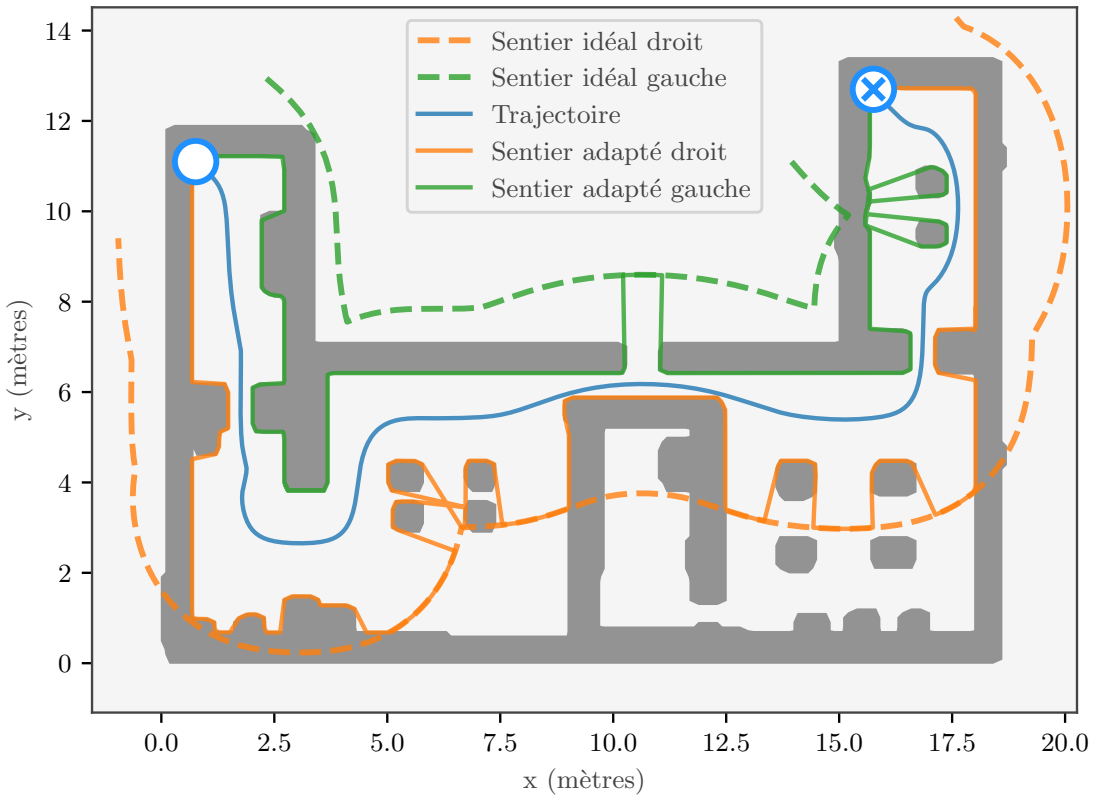

FIGURE 6.6 – Sentiers adaptés calculés en utilisant tous les points

origine. Il est facile de déterminer cet ensemble $G_k \subset G$ pour le sentier gauche par exemple, composé uniquement des points de G à portée de o_k :

$$G_k = \{g \in G \mid \|o_k - g\| \leq p\}$$

La contrainte précédemment modélisée est qu'un point du sentier n'est considéré comme solution potentielle que s'il est à portée de l'origine. Une autre contrainte modélisable est l'impossibilité de couvrir une trajectoire infinie dans la zone de diffusion d'un phare —sauf si cette trajectoire tourne en boucle—. En effet, nous avions déterminé dans la section 6.1 que la longueur maximale couvrable était la petite corde c'. L'ensemble $T_k \subset T$ des points de la trajectoire à suivre qu'il est possible de couvrir en même temps que le point o est défini de la manière suivante :

$$T_k = \{t \in T \mid index(t) > index(o_k), \|t - o_k\| \leq c'\}$$

Nous avons déterminé précédemment deux ensembles de solutions —les sections de sentiers adaptés gauche G_k et droit D_k — qui sont une liste de positions (x, y) possibles, parmi lesquelles il va falloir choisir le meilleur candidat. Définissons plus en détails notre problème. Ici, il s'agit de choisir le couple $(p,\ \alpha)$ — respectivement position (un point) et orientation du phare— optimisant le nombre de points de la trajectoire dans la zone de diffusion en ne considérant que la section de trajectoire déterminée précédemment.

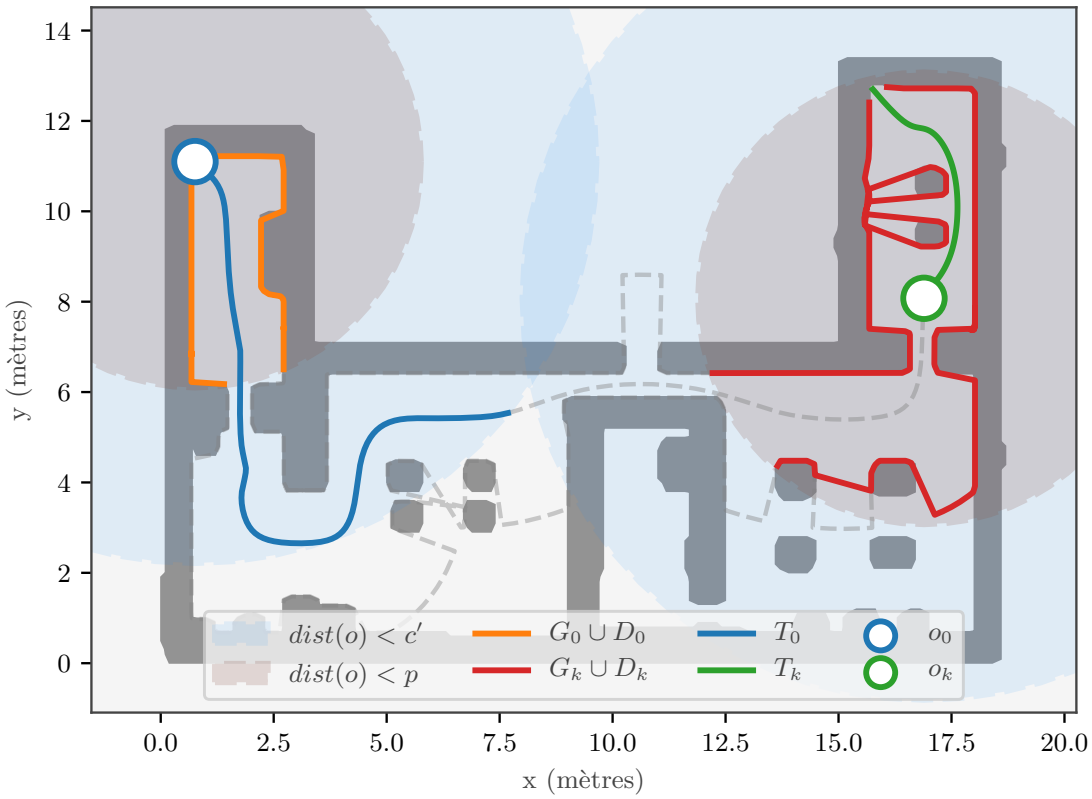

FIGURE 6.7 – Deux sections distinctes générées indépendamment

Il existe de très nombreux couples à évaluer, même en discrétisant l'ensemble A des angles α possibles en incréments de quelques degrés. En effet, le nombre n de candidats est le produit des grandeurs de nos différents ensembles finis :

$$n_k = \left(\underbrace{Card(G_k) + Card(D_k)}_{= Card(G_k \cup D_k)} \right) \times Card(A)$$

Le tableau 6.1 présente l'évolution de la cardinalité n de l'espace de solutions en fonction de la discrétisation d'angle choisie. Ces valeurs permettent de constater que la cardinalité explose rapidement lorsque l'on augmente la précision d'angle, ce qui a très probablement pour conséquence d'écarter la possibilité de trouver une solution précise —au niveau angulaire— dans un temps raisonnable. Nous allons donc par la suite étudier des heuristiques permettant de déterminer directement l'angle α du phare de manière analytique. Nous avons bon espoir que cela aidera à la fois à trouver une solution angulaire décente et à réduire de manière drastique le nombre de solutions à évaluer lors de l'étape d'optimisation. Ainsi la cardinalité de l'ensemble des solutions sera donnée par $n = Card(G_k \cup D_k)$.

6.5 Vers une heuristique d'orientation des phares

Comme nous l'avons démontré précédemment, inclure l'orientation du phare dans la construction de l'ensemble des solutions rend le calcul de minimisation très coûteux en pratique. nous

TABLE 6.1 – Évolution du nombre de solutions en fonction de la discrétisation d'angle

Discrétisation α	$Card(A)$	$Card(G_k \cup D_k)$	n
45°	8	218	$\approx 10^3$
10°	36	218	$\approx 10^3$
1°	360	218	$\approx 10^4$
0.1°	3600	218	$\approx 10^5$
0.01°	36000	218	$\approx 10^6$

allons donc essayer dans cette section de déterminer, d'une manière analytique, une seule orientation pour chacun des points de l'ensemble $G \cup D$.

Nous avions évoqué dans la section de modélisation du problème 6.1 la possibilité d'orienter le phare de telle manière que l'axe du milieu du champs de diffusion du phare —notre secteur circulaire— soit perpendiculaire à la trajectoire. Il est important de préciser ici que nous nous intéressons uniquement à une section T_k de la trajectoire déterminée à la section précédente.

La trajectoire n'étant pas rectiligne, il faut dans un premier temps en déterminer une approximation linéaire —une droite R — via une classique régression par moindres carrés. Nous utiliserons ici une implémentation proposée dans le module *polynomials* de *Numpy*[24].
Une fois l'équation de la droite R déterminée, il est aisé de calculer son vecteur normal $\vec{u}$, qui va nous servir à déterminer l'orientation α du phare, en effectuant un calcul avec le vecteur $\vec{v} = \begin{pmatrix} 0\,, & 1 \end{pmatrix}$ représentant l'orientation initiale du phare. nous allons pour cela introduire la fonction Γ permettant d'obtenir un angle signé représentant la rotation entre deux vecteurs (on utilise ici une fonction auxiliaire Λ de normalisation) :

L'angle α est donc maintenant simple à évaluer :

$$\alpha = \Gamma\left(\vec{u},\ \vec{v}\right)$$

La figure 6.8 illustre le calcul précédent, et en montre le résultat obtenu pour le point SG_{23}. Il est alors aisé de constater que le résultat obtenu n'est pas optimal. En effet, le point t_{31} pourrait être couvert en inclinant légèrement le phare dans sa direction. L'heuristique proposée précédemment n'est donc pas idéale, en plus de demander de nombreux calculs pour effectuer la régression linéaire.

La seconde heuristique consiste à déterminer l'orientation du phare en fonction de la position du premier et du dernier point de la section considérée. Soit $\vec{u}_{min}$ le vecteur partant de la position candidate g_i pour aller à la position t_j, premier point de la section de trajectoire considérée et $\vec{u}_{max}$ le vecteur partant de g_i vers la position t_k, dernier point de cette section. Introduisons la fonction $A(\vec{u}_{min},\ \ \vec{u}_{max})$, calculant l'orientation du phare à partir des deux vecteurs précédemment déterminés :

$$A : \left| \begin{array}{rcl} \mathbb{R}^2 \times \mathbb{R}^2 & \longrightarrow &]-\pi,\ \pi] \\ \vec{u}_{min} \times \vec{u}_{max} & \longmapsto & \Gamma\left(\begin{pmatrix}0\\1\end{pmatrix},\ \ \frac{\frac{\vec{u}_{min}}{\|\vec{u}_{min}\|} + \frac{\vec{u}_{max}}{\|\vec{u}_{max}\|}}{2}\right) \end{array} \right.$$

24. Documentation https://numpy.org/doc/stable/reference/routines.polynomials.package.html#module-numpy.polynomial

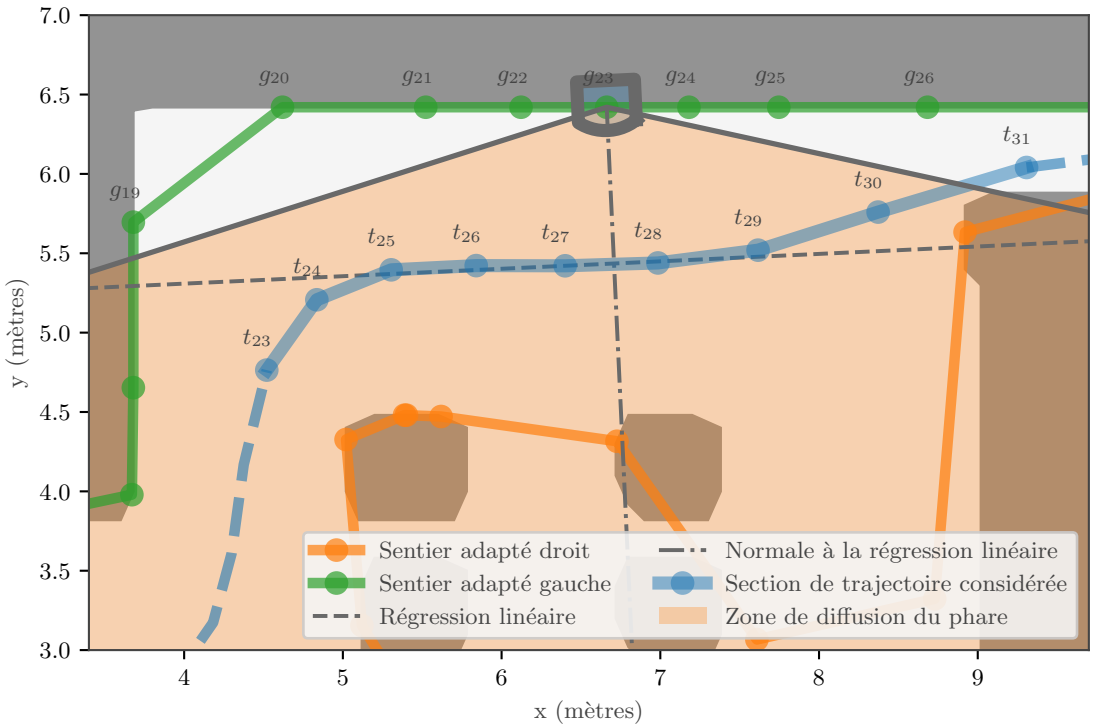

FIGURE 6.8 – Determination de l'orientation du phare

L'orientation α du phare est donc simplifiée par la fonction A :

$$\alpha = A\left(\vec{u}_{min}, \vec{v}_{min}\right)$$

Le résultat de la seconde approche présentée est visible sur la figure 6.9. Bien que fournissant de manière générale de meilleurs résultats que la première version, cette heuristique n'offre pas par définition la garantie d'inclure tous les points intermédiaires de la section. En effet, si l'on considère la trajectoire alternative représentée dans la figure, l'orientation calculée sera la même, mais certains points ne seront pas couverts, alors même qu'il existe une solution d'angle optimale. Cette sous-optimalité de l'angle aura pour conséquence directe de diminuer fortement le potentiel de la position considérée, qui avec un angle adéquat pourrait être une excellente solution.

Il devient donc évident que la détermination analytique de l'orientation du phare devra prendre en compte l'ensemble des points de la section de trajectoire pour fournir une solution optimale. Or, nous allons déjà parcourir tous ces points lors de la phase d'évaluation d'un point candidat, il fait donc sens de confier le calcul de cette orientation à la fonction d'évaluation de la pertinence d'un point. Qui par conséquent fournira également l'angle optimal pour ce point.

6.6 Un modèle des occlusions de l'environnement

Nous avions déterminé précédemment par dilatation des obstacles (5.2) et calcul des contours (5.3), des zones restreintes permettant d'éviter les obstacles en toute sécurité. Nous avons utilisé ces zones restreintes dans de nombreux algorithmes, mais elles ne correspondent pas à nos besoins ici. En effet, les zones restreintes sont potentiellement beaucoup plus grandes que les obstacles

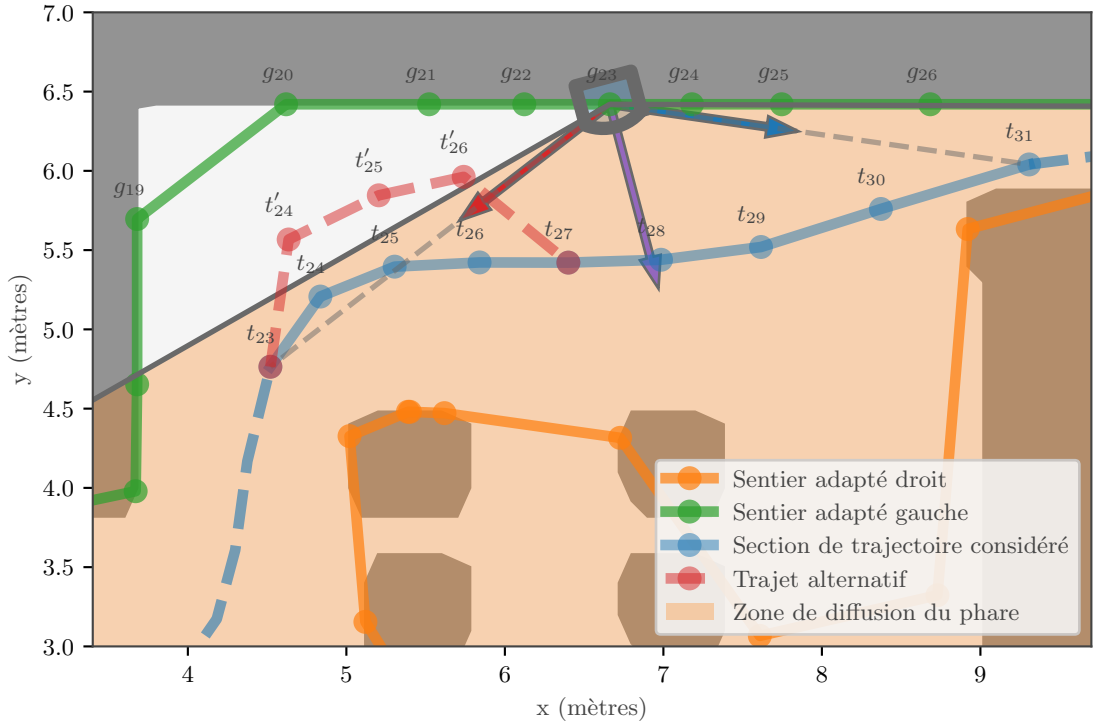

FIGURE 6.9 – Seconde heuristique de calcul de l'orientation du phare

qui ont permis de les générer. Or nous nous intéressons désormais au maintien de la ligne de vue, nous ne pouvons utiliser directement ce modèle de l'environnement.

La figure 6.10 présente le modèle d'occlusion obtenu de manière analogue à la détermination des zones restreintes. Nous avons donc un ensemble O de polygones nous permettant de réaliser des tests de visibilités qui correspondent à la réalité du terrain.

6.7 Fonction d'évaluation de solution locale

Afin d'évaluer la pertinence d'un point p, il faut déterminer une heuristique de calcul de l'orientation α du phare, permettant de transformer ce point en une solution (p, α). nous avons vu que les heuristiques proposées précédemment, qui ne prenaient pas en compte complètement la géométrie des points ne permettaient pas d'obtenir une solution optimale. Nous allons donc cette fois construire une fonction qui parcourt l'ensemble des points de la section de trajectoire.

Lorsque l'on considère un seul point de l'un des sentiers, l'ensemble des points de la trajectoires est compris dans un secteur circulaire *virtuel*, défini par deux rayons et une portée. Nous allons modéliser ces rayons par deux vecteurs $\vec{u}_{min}$ et $\vec{u}_{max}$. Nous reviendrons sur la portée plus tard. L'idée globale de l'algorithme est d'agrandir le secteur circulaire virtuel pour qu'il inclue chacun des points de T_k, et ce tant que le secteur virtuel n'a pas atteint les dimensions maximales de la zone de diffusion du phare.

La figure 6.11 présente différentes étapes de l'agrandissement du secteur circulaire *virtuel*. Les vecteurs $\vec{u}_{min}$ et $\vec{u}_{max}$ sont mis à jour au besoin, à chaque ajout de point t_k, si le point



FIGURE 6.10 – Visualisation du modèle "zones d'occlusions" de l'environnement

est à l'extérieur du secteur circulaire. Si mise à jour il y a, on vérifie que l'angle d'ouverture θ du secteur virtuel est toujours inférieur à l'angle maximal permis par le phare. La distance entre le point à ajouter et la position candidate du phare est également calculée. Si les deux conditions $\theta \leq \theta_{max}$ et $\|p - t_k\| \leq \textit{portée}$ sont respectées, il est possible de couvrir t_k depuis p. Nous nommons Ψ la fonction qui calcule les vecteurs $\vec{u}_{min}$ et $\vec{u}_{max}$:

$$\Psi : \left| \begin{array}{rcl} \mathbb{R}^2 \times T & \longrightarrow & \mathbb{R}^2 \times \mathbb{R}^2 \\ (x, y) \times t \in T & \longmapsto & \vec{u}_{min} \text{ et } \vec{u}_{max} \end{array} \right.$$

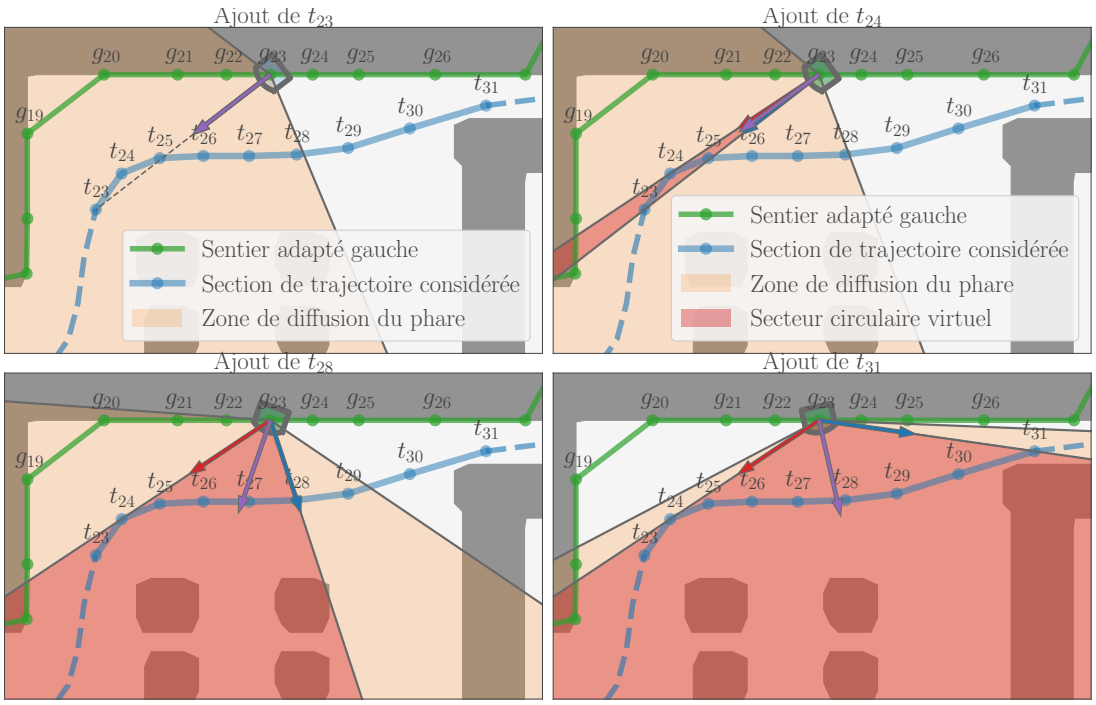

FIGURE 6.11 – Visualisation de l'algorithme d'agrandissement du secteur virtuel

Bien que les vérifications précédentes garantissent la présence du point t_k dans le secteur de diffusion du phare positionné en p, les occlusions potentielles dues à l'environnement ne sont actuellement pas prises en comptes. En effet, un obstacle pourrait occulter la ligne de vue, rendant la solution inefficace en pratique. Nous disposons d'un modèle d'occlusions, un ensemble O de polygones représentant les obstacles, déterminé précédemment. Il nous faut maintenant déterminer comment exploiter ces polygones efficacement. De manière analogue aux tests d'angles et de portée, nous allons donc construire un test de visibilité. Afin d'accepter un nouveau point t_k dans la section couverte, il faut non seulement que ce point soit visible depuis p, mais également que le segment allant de t_{k-1} à t_k soit intégralement visible pour que l'on puisse se rendre de l'un à l'autre sans perdre le lien visuel avec le phare. Pour garantir cette visibilité, il faut qu'aucun obstacle ne se trouve dans le triangle Δ défini par les points t_{k-1}, t_k et p. Nous pouvons pour cela utiliser l'opérateur d'intersection et ainsi vérifier la propriété de visibilité :

$$\forall\ o \in O, \qquad o \cap \Delta = \varnothing$$

La figure 6.12 présente les triangles ainsi générés, et une intersection avec la zone d'occlusion. Chaque triangle libre d'intersection garantit la visibilité entre le point candidat et un segment de la trajectoire. Ainsi, l'algorithme s'arrêtant à la première intersection, la visibilité sur l'ensemble

des points précédents de la trajectoire —et des segments entre ces points— est garantie.

Nous savons maintenant comment tester un à un les points de la trajectoire pour savoir s'ils sont couverts depuis le point candidat. Il faut toutefois traduire les deux vecteurs en une orientation grâce à la fonction A définie dans la section précédente.

Discutons maintenant de l'évaluation à proprement parler, c'est à dire de la manière de calculer le *score* de chaque solution. Comme évoqué au début de ce chapitre, l'objectif de la maximisation locale que nous essayons de réaliser est de couvrir le plus de trajectoire possible. Le score doit donc être proportionnel à cette grandeur physique mesurable. Deux options s'offrent à nous, nous pouvons simplement compter le nombre de points couverts où nous pouvons calculer directement cette distance en sommant au fur et à mesure les distances $\|t_{k-1} - t_k\|$. Soit l l'indice du premier point t_l non couvert depuis la position p, le score est donné par la somme suivante :

$$\Phi : \left| \begin{array}{ccc} \mathbb{R}^2 & \longrightarrow & \mathbb{R} \\ (x,y) & \longmapsto & \sum_{k=1}^{l-1} \|t_{k-1} - t_k\| \quad \text{avec} \quad t_k \in T \end{array} \right.$$

Il est également possible d'intégrer d'autres contraintes par pondération des scores. Notamment, on peut prendre en compte le coût énergétique du déplacement des phares ou encore la disparité de la précision du positionnement dans la zone de captation du phare. Le système utilisé dans nos expériences présentant une précision homogène, il n'a pas été nécessaire d'inclure ce critère dans le score.

6.8 Optimisation sur les sentiers

L'optimisation locale consiste à évaluer tous les points de l'ensemble des solutions possibles et sélectionner le(s) meilleur(s) —nous maximisons ainsi la longueur de trajectoire couverte. L'algorithme est donc un simple parcours de l'espace des solutions, en évaluant le point grâce à la fonction définie précédemment.

$$P^* = \underset{p \in P}{\operatorname{argmax}}\, \Phi(p) \; = \{(p,\alpha) \mid \forall q \in P, \quad \Phi(q) \leq \Phi(p) \quad avec \quad \alpha = A(\Psi(p))\}$$

La figure 6.13 permet de visualiser le score de chaque point des sentiers via une échelle de couleurs. Un phare et sa zone de diffusion sont dessinés au placement (p, α) qui maximise la trajectoire couverte. Le résultat obtenu sur cette section de trajectoire est satisfaisant, étudions maintenant la construction globale à partir des briques que nous venons de définir.

6.9 Construction d'une solution globale

Comme évoqué lors de l'introduction à ce chapitre, la construction d'une solution globale optimale à notre problème de minimisation des déplacements de phares nécessaires au suivi complet de la trajectoire est simple. Il suffit en effet de résoudre le problème local de maximisation de trajectoire couverte en commençant par le début de la trajectoire. Il faut ensuite réduire la trajectoire à traiter en soustrayant la partie couverte précédemment. On boucle ainsi tant qu'il reste des points dans la trajectoire à traiter.

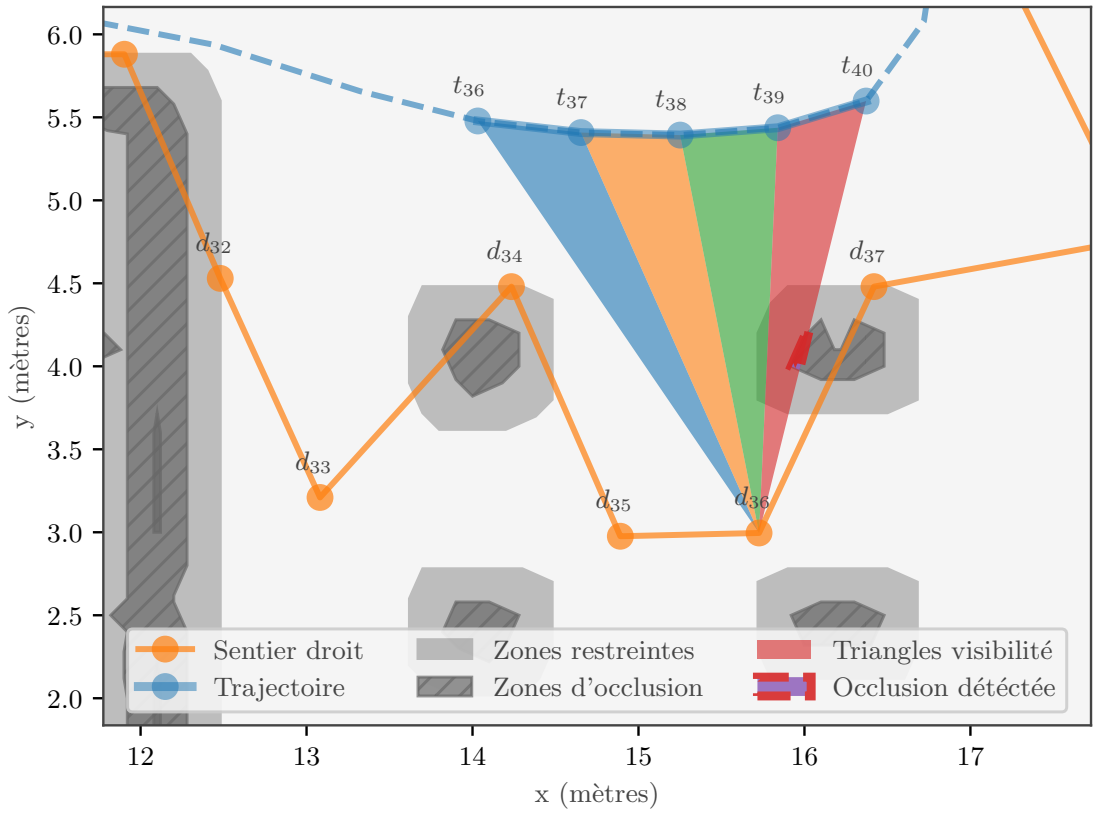

FIGURE 6.12 – Tests de visibilité par intersection de triangles

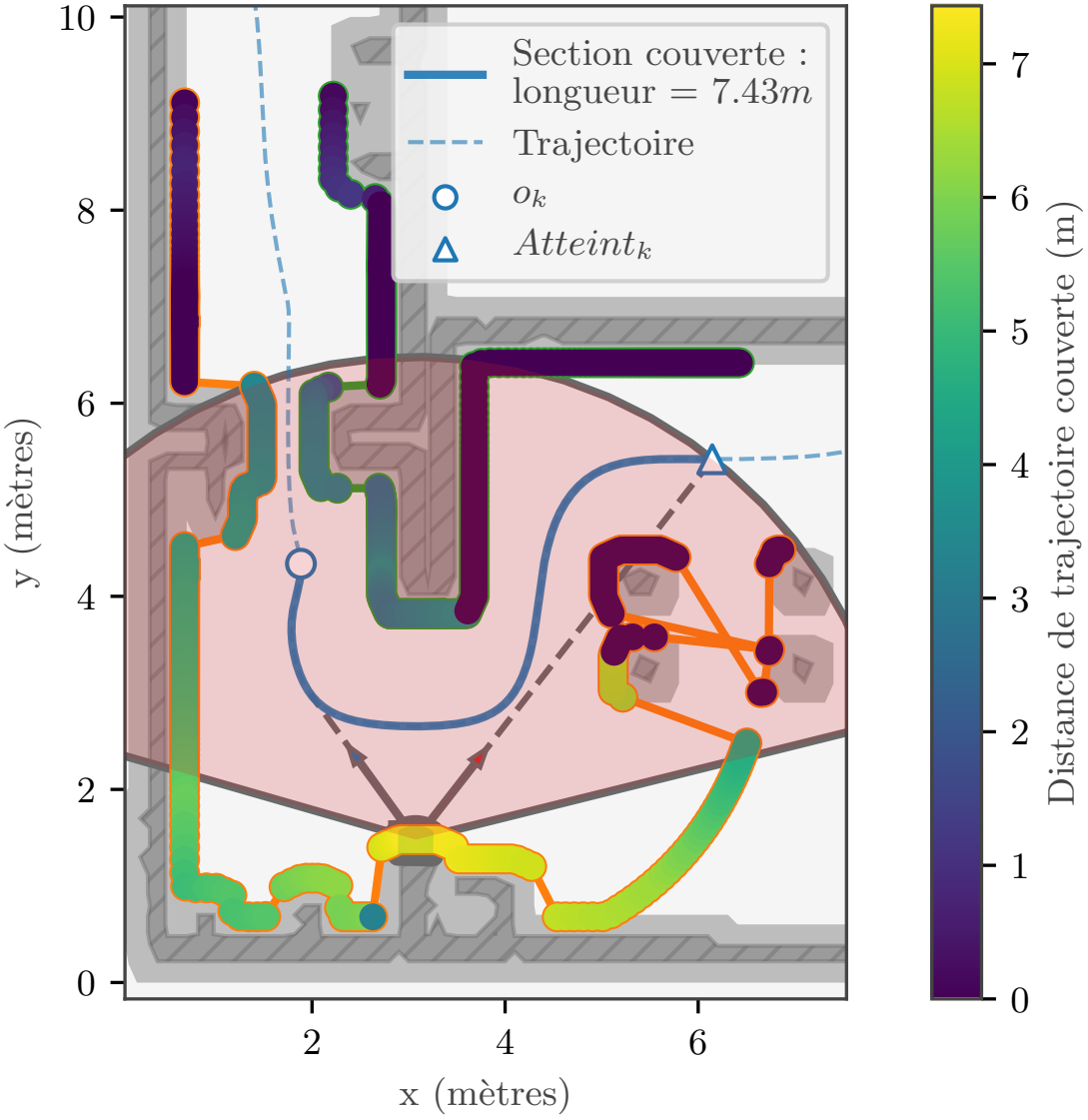

FIGURE 6.13 – Scores obtenus pour un section arbitraire

La figure 6.14 permet de visualiser les différentes étapes nécessaires au suivi de la trajectoire de bout en bout. Les points évalués localement ainsi que leurs scores associés sont dessinées sur chaque étape. La solution optimale est choisie parmi ces points —elle est représentée par le secteur circulaire de diffusion sur la figure— permettant par la même occasion de déterminer la section couverte. Ce processus est ensuite répété jusqu'au bout de la trajectoire.

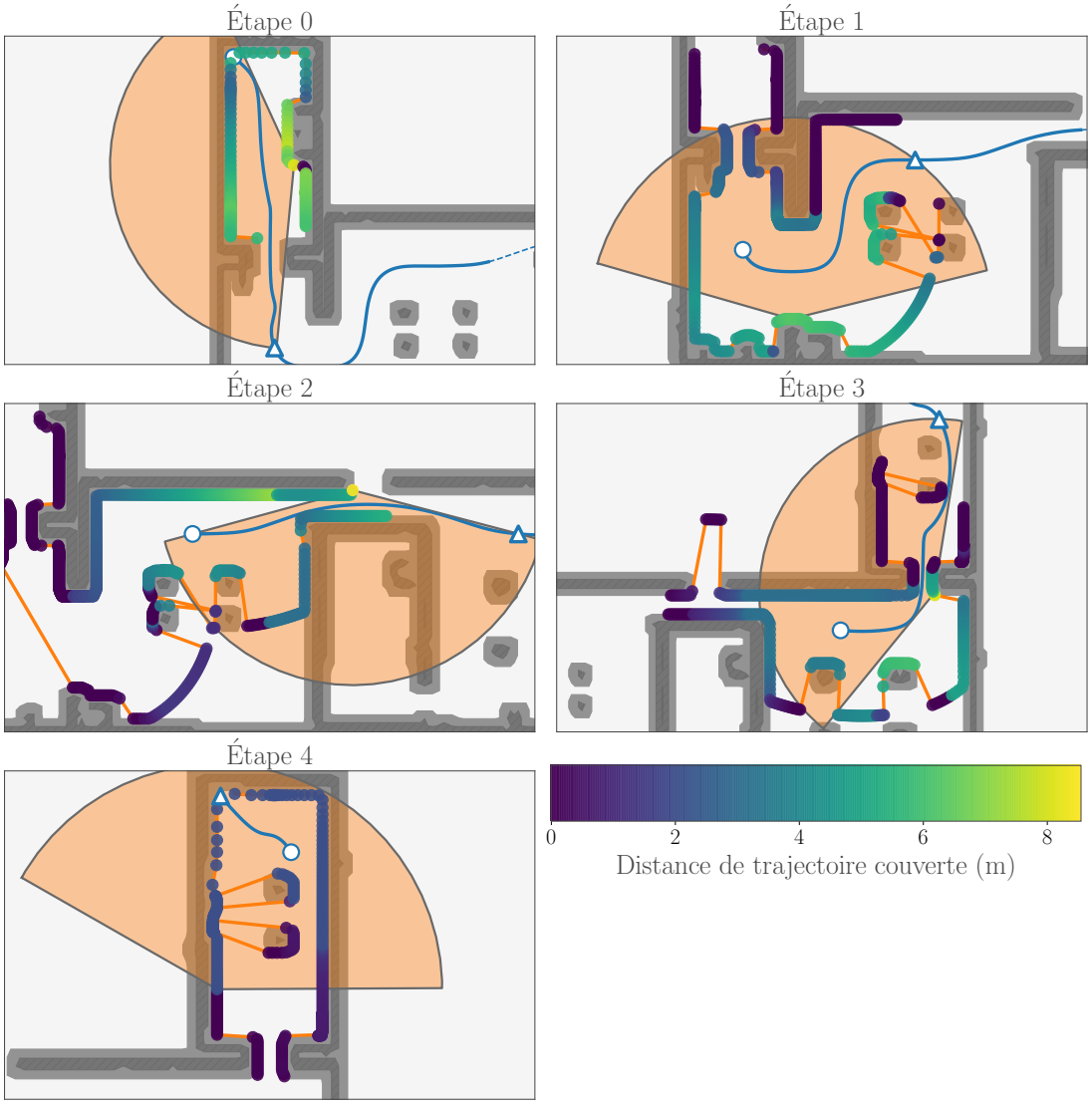

FIGURE 6.14 – Succession d'étapes d'optimisations locales constituant une solution globale

La solution globale ainsi obtenue est résumée dans la figure 6.15, où les étapes successives sont superposées sur la carte, formant une solution optimale à notre problème initial. La pertinence de cette solution s'évalue par le nombre d'étapes nécessaires à la réalisation de la mission.

6.10 Prendre en compte plus de solutions

Bien que les résultats obtenus précédemment soient satisfaisant, nous nous demandons si le fait de ne considérer que les sentiers —qui représentent l'optimal pour une trajectoire purement

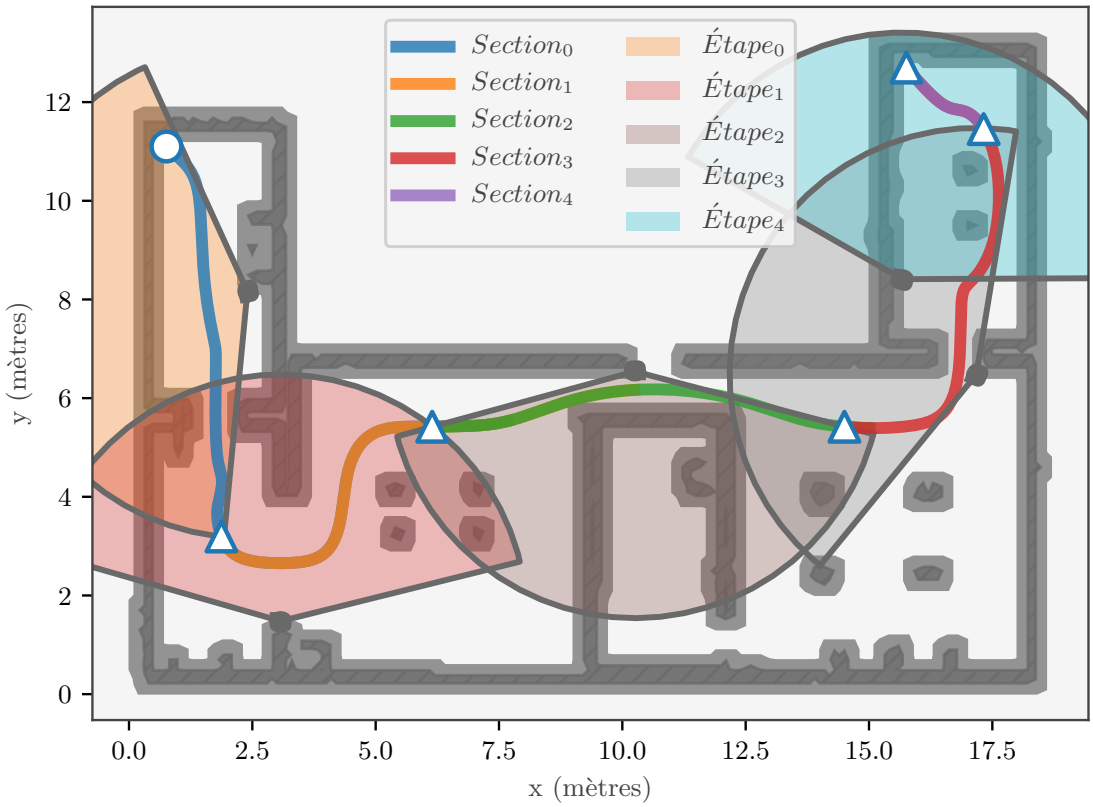

FIGURE 6.15 – Solution globale obtenue à partir des sentiers

rectiligne— n'écarte pas de meilleures solutions. En effet, dans de nombreux cas pratiques, la trajectoire à suivre est loin d'être rectiligne, étendre l'espace de solutions à toute la surface entre les sentiers permettrait de mieux couvrir les sections à forte courbure. Augmenter ainsi l'espace de solutions aura évidemment un coût en calculs, mais il convient d'établir si le gain de performance justifie la diminution de l'efficience de l'algorithme.

Afin de déterminer la surface de solutions, nous utiliserons les sections de sentiers G_k et D_k déterminées précédemment 6.4. Ainsi, la surface S_k sera modélisée par un polygone dont les sommets sont l'ensemble des points $(G_k \cup D_k)$. Toutefois, le phare ne devant pas gêner l'agent principal dans le suivi de sa trajectoire, nous devons écarter les points trop proches de cette dernière. Notons R_k cette zone réservée, qui est définie par la contrainte de respect de la distance de sécurité d de la manière suivante :

$$R_k = \left\{ q \in \mathbb{R}^2 \mid \forall t \in T_k, \quad \|t - q\| < d \right\}$$

La figure 6.16 présente les surfaces et zones réservées de deux sections distinctes de la trajectoire principale. Ainsi l'ensemble P_k des solutions pour une section k sera défini par la différence entre ces deux surfaces :

$$P_k = S_k - R_k$$

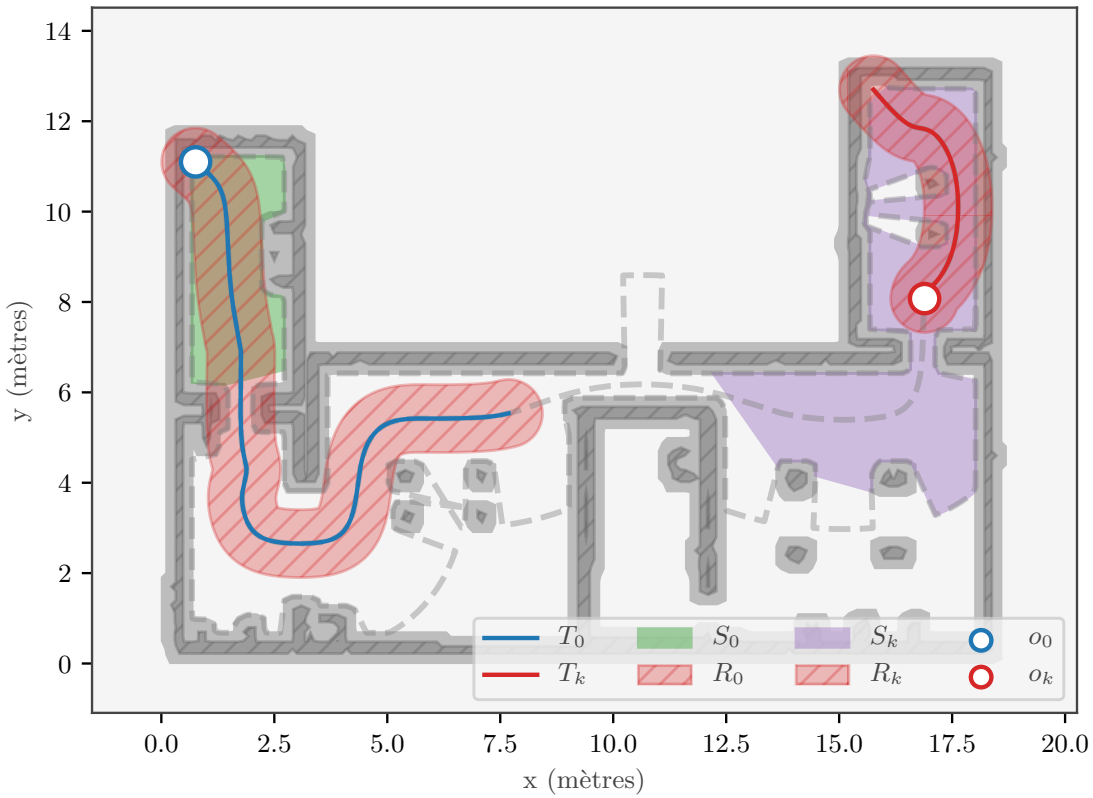

FIGURE 6.16 – Génération d'une surface de solutions depuis deux points distincts

La surface P_k sera alors modélisée par l'ensemble des polygones résultants de l'opération de différence. Nous avons ainsi déterminé un espace de solutions plus large que les précédents sentiers, offrant probablement de meilleures solutions locales.

6.11 Une meilleure solution locale

Nous avons déterminé précédemment un nouvel espace de solutions, non plus sous la forme d'un ensemble de points comme c'était le cas pour les sentiers, mais sous la forme d'une surface délimitée par un ensemble de polygones. Or, la fonction Φ proposée dans la section 6.7, ne permet de fournir un score que pour un point donné. Il nous faut donc échantillonner la surface P_k, nous allons pour cela utiliser une grille uniformément espacée, générant ainsi un sous-ensemble de points $E_k \subset P_k$. Ayant maintenant affaire à des points, il est simple d'évaluer leur pertinence respective.

La figure 6.17 présente la grille de points évalués, ainsi que leur score. Une fois de plus, la discrétisation —ici la distance entre chaque échantillon testé— est volontairement choisie pour simplifier la compréhension et la visualisation des résultats. Il est toutefois possible de faire varier cette distance afin de tester plus de candidats, et ainsi ajuster le rapport coût/performance de l'algorithme d'optimisation locale.

La détermination des solutions optimales consiste à choisir une solution parmi l'ensemble E^* :

$$E^* = \underset{e \in E}{\operatorname{argmax}}\, \Phi(e) \ = \{(e, \alpha) \mid \forall q \in E, \quad \Phi(q) \leq \Phi(e) \quad avec \quad \alpha = A(\Psi(e))\}$$

La solution locale trouvée est meilleur dans cet exemple, car le point choisi couvre une portion plus longue de la trajectoire. Comme ce approche traite plus de points, il est possible de trouver une meilleure solution que dans l'approche sentiers. En contrepartie, le coût total de l'optimisation augmente linéairement avec le nombre de points testés, mais l'ordre de grandeur du nombre de points testés est égal au carré du nombre de points testés pour la version sentiers, pour une résolution Δd égale.

$$Card(P) \leq \frac{\|G_k\| + \|D_k\|}{\Delta d}$$
$$Card(E) \leq \left(\frac{portée}{\Delta d}\right)^2$$

6.12 Échantillonage multi-échelle

Afin de diminuer le coût, nous avons mis en place une stratégie d'évaluation multi-échelle permettant de réduire drastiquement le nombre de points à évaluer. En effet, il est possible d'obtenir une résolution fine, sans tester l'ensemble de la grille à haute résolution. L'astuce consiste à faire un premier échantillonage à faible résolution, procéder à l'évaluation des points, sélectionner le meilleur, puis répéter le processus avec une résolution plus fine, mais dans une zone limitée autour du meilleur point précédemment sélectionné.

Ainsi, pour obtenir une résolution de $1cm = 0.01m$, dans une portée de $5m$ avec une seule échelle, il faudrait :

$$Card(E) \leq \left(\frac{portée}{\Delta d}\right)^2 \approx 10^5$$

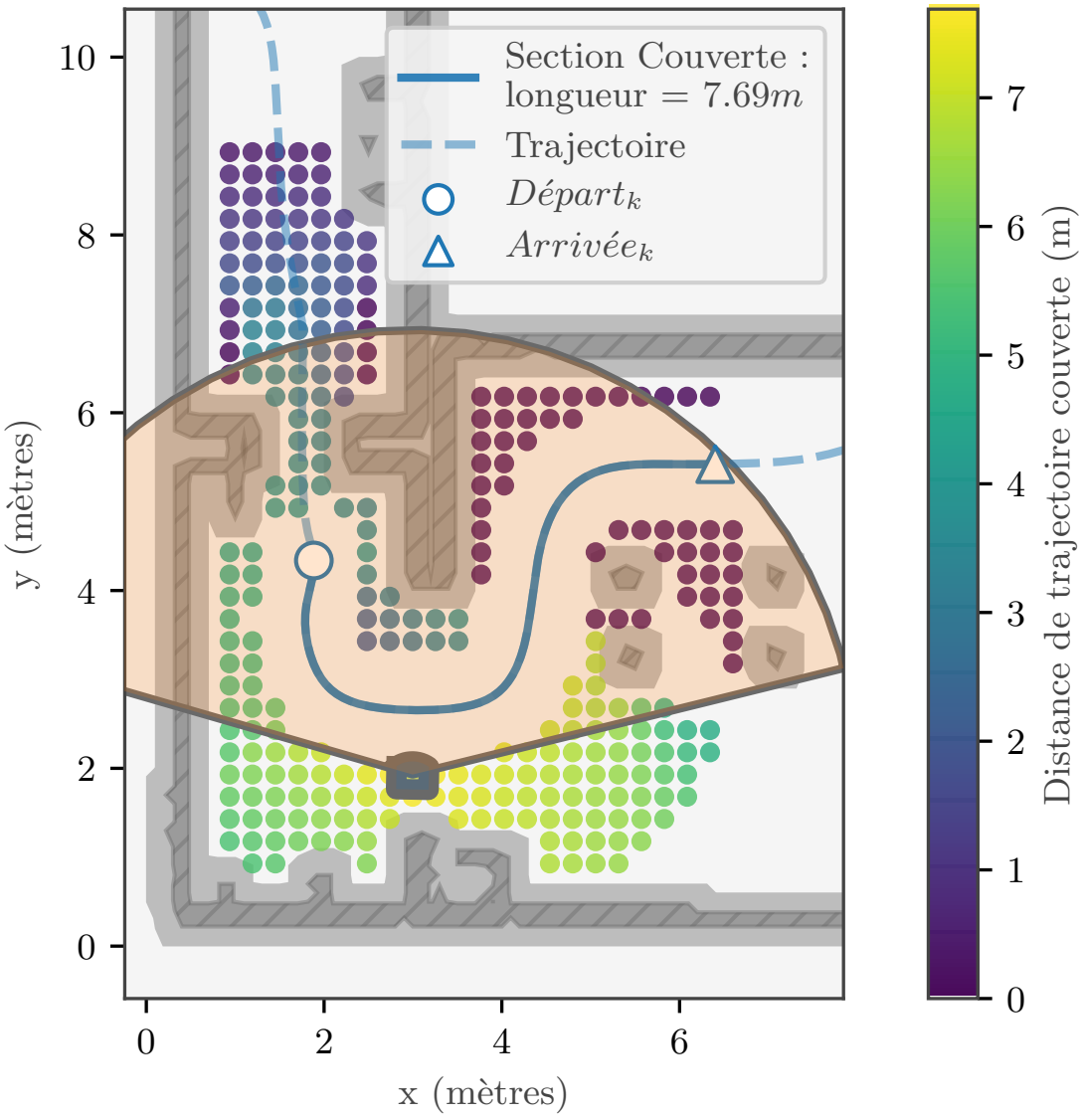

FIGURE 6.17 – Échantillonnage régulier et optimisation locale

En Appliquant cette fois ci une recherche avec une succession d'échelles (par exemple $d = 1, 0.1, 0.01$) le nombre d'échantillons à tester est déterminable de la manière suivante :

$$Card(E_{multi}) \leq \left(\frac{portée}{\Delta d_0}\right)^2 + \sum_{n=1}^{card(d)} \left(\frac{\Delta d_{n-1}}{\Delta d_n}\right)^2 \approx 10^2$$

Cette approche permet donc de limiter très fortement le nombre de calculs nécessaires pour obtenir une résolution fine. Toutefois, ne testant pas l'ensemble des points de la grille à résolution fine, il est possible de tomber dans un extremum local, dont la valeur peut être éloignée de l'optimum global qui aurait été trouvé avec l'approche mono-résolution. Il est alors possible de sélectionner plus d'un point à chaque étape, limitant ainsi ce risque. Bien que ce garde fou ait un coût en calculs, il reste très limité par rapport au coût de l'approche mono-résolution :

$$Card(E_{multi}) \leq \left(\frac{portée}{\Delta d_0}\right)^2 + n \times \sum_{i=1}^{card(d)} \left(\frac{\Delta d_{i-1}}{\Delta d_i}\right)^2 \approx 10^3$$

La figure 6.18 présente l'approche multi-échelle et les résultats obtenus à chaque résolution, ainsi que le résultat obtenu avec l'approche mono-résolution avec $\Delta d = 0.1$. Une rapide analyse quantitative et qualitative permet de valider l'approche, qui permet d'obtenir des résultats aussi précis. Il est même possible, au vu de la réduction du coût en calcul, de calculer à une résolution supérieure, ce qui prendrait trop de temps en pratique pour l'approche mono-échelle.

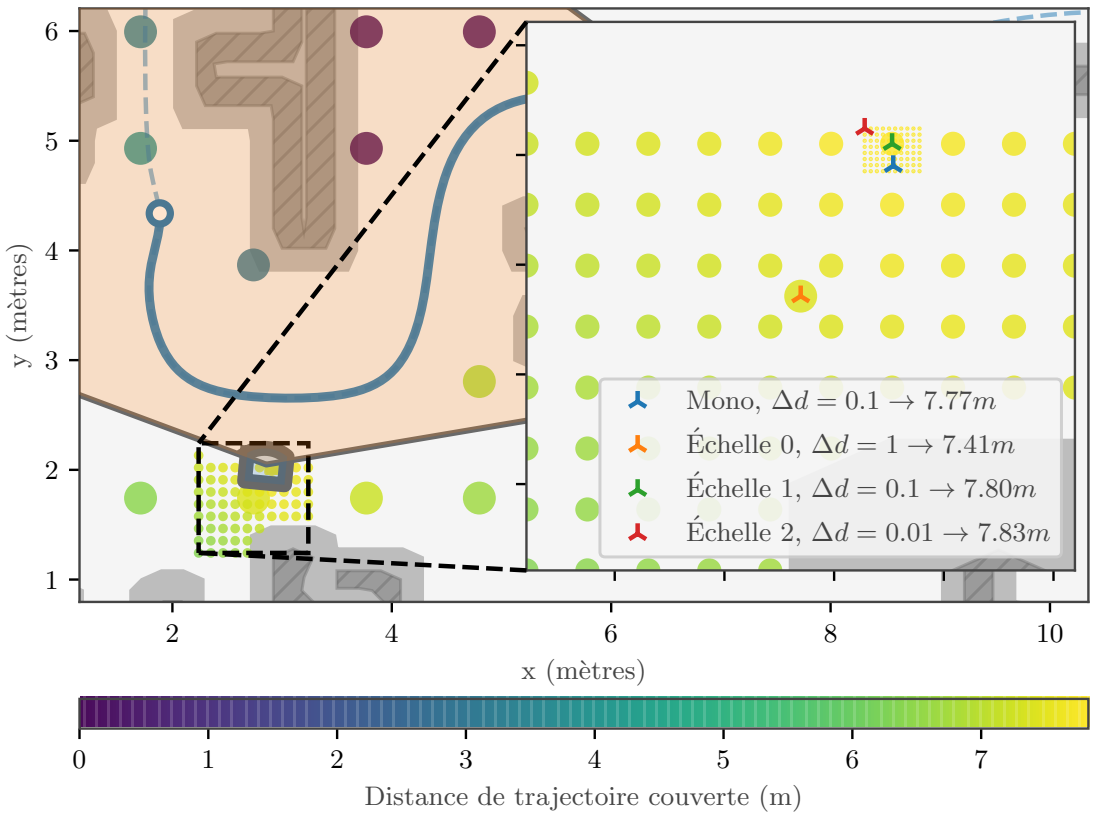

FIGURE 6.18 – Visualisation de l'échantillonage progressif

6.13 Une meilleure solution globale ?

La succession d'optimisations locales obtenues avec l'approche par surface est présentée sur la figure 6.19. Bien que la discrétisation choisie soit volontairement grossière afin de faciliter la visualisation, on peut constater une amélioration des positions choisies, particulièrement autour des virages important de la courbe principale.

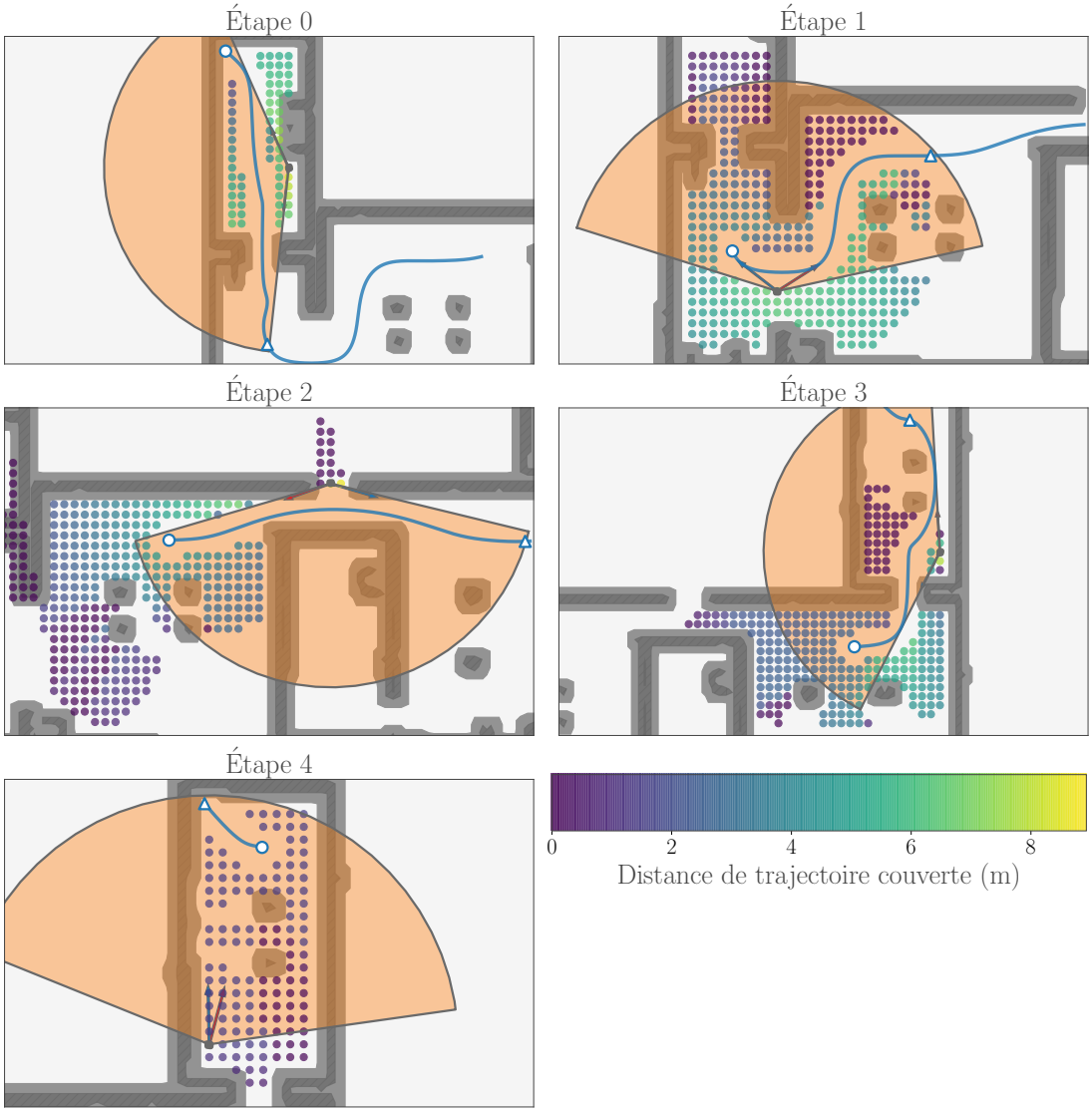

FIGURE 6.19 – Succession d'étapes locales, avec résolution faible pour visualisation

Maintenant que nous avons montré que l'optimisation locale basée sur une surface produit de meilleurs résultats que celle basée uniquement sur les sentiers, il convient d'étudier l'impact de cette amélioration locale sur l'optimisation globale.

La figure 6.20 présente les résultats obtenus par l'approche surface, avec $\Delta d = 0.1$. Si l'on se fie uniquement à la métrique initialement proposée dans ce chapitre —le nombre d'étapes nécessaires à la réalisation de la mission— il est impossible de distinguer les différentes approches proposées. En effet, la carte d'exemple proposée ne contient pas de parcours suffisamment longs

pour apprécier pleinement le gain de performance de l'approche surface. On peut toutefois noter qu'ici, la dernière étape est vraiment courte, ce qui traduit directement le fait que les autres étapes sont plus longues. On peut donc s'intéresser à une autre métrique, la longueur moyenne des sections de trajectoire couvertes.

Nous mettrons de coté la première et la dernière étape, qui sont des cas particuliers et ne sont donc pas représentatives. Le tableau 6.2 présente les résultats et coûts moyens obtenus sur notre exemple.

TABLE 6.2 – Comparatifs en coûts et performance pour les différentes approches.

	sentiers	surface	taux d'augmentation
score moyen	$7.65m$	$7.94m$	$\approx 3.7\%$
coût moyen	$244.21pts$	$2025.00pts$	$\approx 729.2\%$

Cette amélioration peut paraître dérisoire, toutefois malgré l'augmentation des calculs nécessaires il est toujours possible de faire ces calculs en pratique, et ces petits gains mis bout à bout lors d'une mission de plusieurs kilomètres résulteront en un gain significatif de temps et d'énergie. De plus, le scénario proposé comporte quelques virages, mais on peut facilement imaginer des environnements plus complexes, dans lesquels l'approche surface se distingue plus de son pendant linéaire.

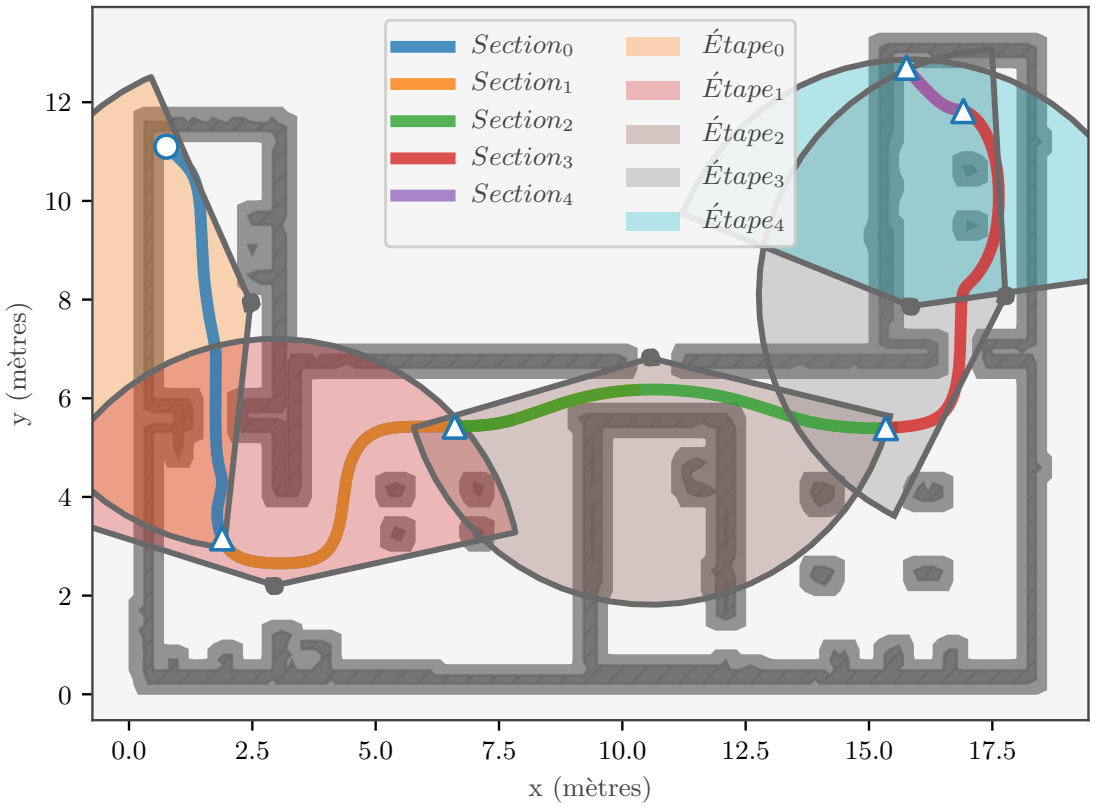

FIGURE 6.20 – Solution globale obtenue à partir des sentiers

6.14 Approche alternative discrète

Cette section décrit un processus alternatif de placement de phare le long d'une trajectoire donnée pour un robot de référence, mais cette fois entièrement basée sur des opérations discrètes sur l'image initiale. La nature discrète du processus vient de la représentation des images numériques par des tableaux de pixels.

6.14.1 Données du problème

Les données du problèmes sont les suivantes :
— Une image numérique représentant la carte (2D) connue de l'environnement
— Une trajectoire : suite ordonnée de positions dans l'image
— Le rayon des robots (en pixels) pour permettre une intégration transparente de robots de différentes tailles
— La portée du phare (en pixels) et l'ouverture angulaire du phare (en degrés)
— Le pas d'interpolation pour le parcours du disque de visibilité (nombre de pixels)

6.14.2 Résultats attendus

Les résultats attendus sont les mêmes que pour l'approche géométrique : deux listes de positions clés. Une pour le robot explorateur et une pour le phare. Par définition du problème, on sait que les positions de départ et d'arrivée du robot sont les première et dernière positions de la trajectoire spécifiée en entrée.

Avant de commencer le processus de calcul, il est nécessaire d'effectuer une dilatation des murs similaire à celle abordée en 5.2. Si le chemin du robot explorateur a été calculé en utilisant la représentation implicite de Voronoï (carte des distances) décrite en 5.12, cette étape peut être omise car elle correspond à un simple seuillage sur la distance aux obstacles. La figure 6.21 présente la carte obtenue à l'issue de cette opération.

6.14.3 Processus itératif de calcul

Le calcul des positions clés est un processus itératif similaire à l'approche géométrique qui s'arrête lorsque le robot a atteint la position cible (dernière position de la trajectoire d'entrée). Chaque itération est composée d'une successions d'étapes :

Extraction d'une section de trajectoire : Dans le but de réduire les calculs, seule une section couvrable —en fonction de la zone de diffusion du phare— de la trajectoire principale est considérée, de manière analogue au calcul de la section de trajectoire principale présentée en 6.4.

Détermination de l'espace de solutions : Toujours dans un souci de minimisation des calculs, la zone de recherche pour la position du phare est circonscrite à une sous-partie du disque autour de la position courante du robot de référence dont le rayon est la portée du phare. Le principe est d'utiliser la portée maximale pour un intervalle d'angles centré dans la direction globale des points du sous-chemin, donc vers la position de leur barycentre. Un rayon inférieur est utilisé pour l'intervalle complémentaire des angles, comme on peut le voir avec le disque partiel autour de la position de départ en 6.21.

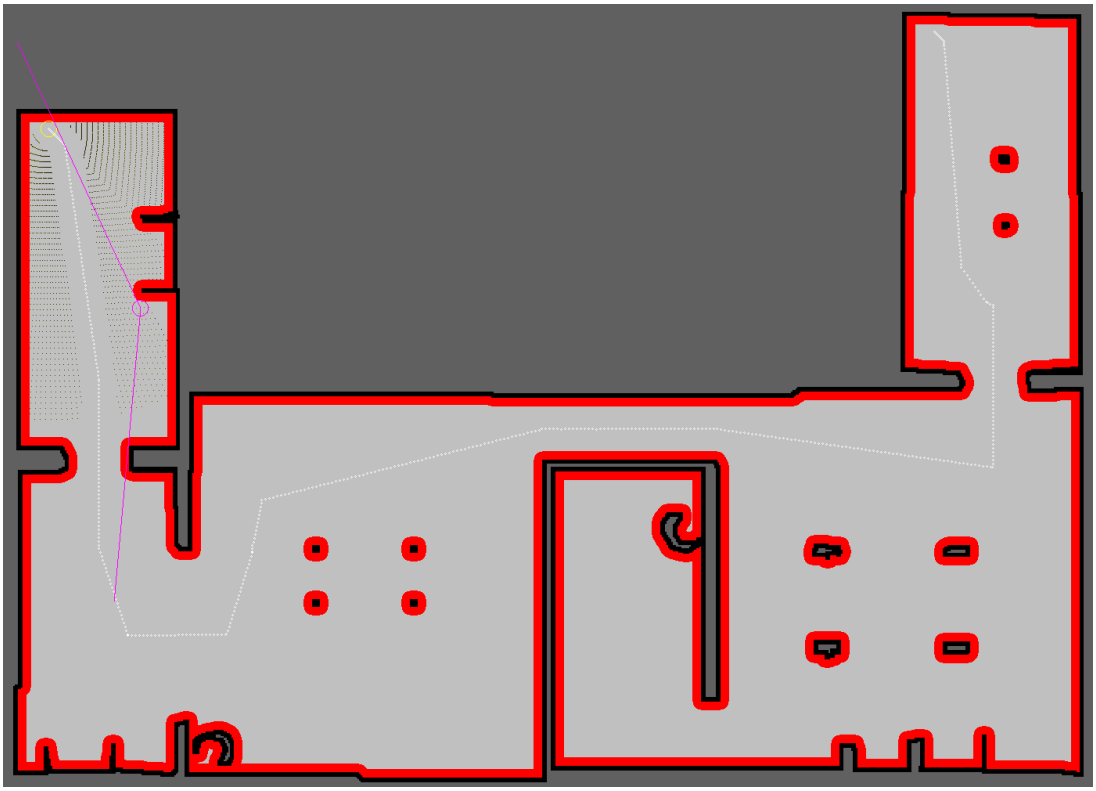

FIGURE 6.21 – Première recherche de position du phare. La couleur rouge indique les contours interdits des murs. Les points blancs sont les points interpolés du chemin. La zone de recherche est définie par le disque partiel autour de la position initiale du robot de référence. Le cercle rose est la position trouvée. Les segments roses indiquent l'orientation de l'ouverture du phare.

Élimination du contour du chemin : Étant donné que l'on cherche une position du phare permettant au robot de référence d'effectuer un déplacement maximal, il est important de veiller à ce que le phare lui-même ne gêne pas le robot. Ainsi, il faut interdire toutes les positions qui pourraient gêner le déplacement du robot le long de son chemin de manière analogue à la zone réservée au robot principal proposée en 6.10.

Balayage des solutions La recherche de la position optimale du phare s'effectue dans un disque partiel autour de la position courante du robot. Comme on l'a vu précédemment, le rayon est maximal dans un intervalle d'angles centré vers le barycentre du sous-chemin et le rayon est plus faible pour les autres angles. Pour balayer le disque, on utilise le pas d'interpolation du disque —qui est un paramètre de l'algorithme— afin de déterminer les différents rayons à parcourir. De même, lors du parcours d'un rayon, ce même pas est utilisé pour sélectionner les positions à évaluer. Afin de tenir compte de la contrainte de visibilité entre le robot et le phare, les rayons sont parcourus depuis le robot de référence vers l'extrémité du disque et le parcours est interrompu dès qu'un mur est rencontré. Lorsqu'une zone interdite est rencontrée (autour d'un mur ou du chemin), le parcours continue sans calculer de score pour cette position. Cette partie de l'algorithme est similaire à l'approche surfacique d'exploration des solutions, présentée en 6.10, mais où l'échantillonnage est directement lié à l'image de la carte.

Fonction d'évaluation : La fonction d'évaluation du score d'une position candidate est identique a celle proposée en 6.7, à ceci près que la contrainte de visibilité n'est pas vérifiée de manière continue entre deux points successifs de la trajectoire. Un tracé de rayon est effectué pour chacun de ces points. Il est toutefois possible de faire des tracés supplémentaires pour obtenir une discrétisation plus fine, au prix d'une augmentation des coûts en calculs.

Sélection du meilleur candidat : Le candidat avec le meilleur score est choisi, puis la position maximale atteignable par le robot principal est déduite de ce meilleur candidat. Si la position maximale est le dernier point de la trajectoire à couvrir, le processus est terminé. Sinon, on répète les étapes successives en prenant la position maximale atteinte dans cette itération comme point de départ du robot pour l'itération suivante.

La figure 6.22 présente les placements obtenus pour une trajectoire et une carte données. On constate que les résultats sont similaires à l'approche géométriques, avec une approche entièrement basée sur des opérations discrètes sur l'image d'entrée. Toutefois, la vérification de la contrainte de visibilité, étant également discrète, peut poser problème dans le cas d'obstacles aux dimensions réduites, qui ne seraient alors pas pris en compte par l'algorithme. Ce problème peut être limité en échantillonant le chemin principal au niveau du pixel.

6.15 Constat et évolutions

Nous avons présenté dans ce chapitre les différents algorithmes originaux permettant de construire progressivement une solution au problème de placement des phares lors du suivi d'une mission. Le processus proposé permet d'obtenir ces positions optimales, tout en étant

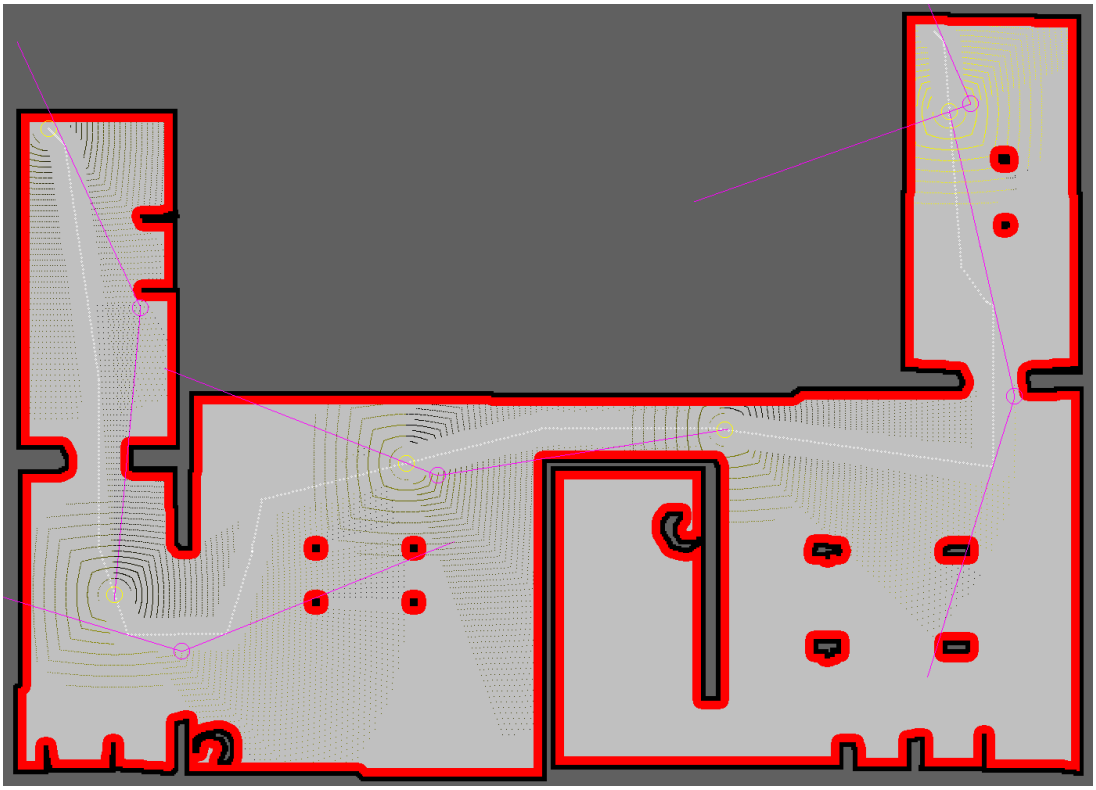

FIGURE 6.22 – Résultat obtenu avec l'approche discrète

fortement paramétrable en fonction des besoins et des contraintes de temps de calcul.

Les différentes approches d'optimisations locales sont interchangeables, présentant chacune un compromis nouveau entre performance et coût. Toutefois, bien que distinctes, ces approches sont toutes robustes et répondent aux contraintes initialement posées.

Ainsi, il est désormais possible de déterminer un ensemble de placements de phares permettant la réalisation complète d'une mission, à partir d'un modèle de l'environnement et d'une trajectoire et ce tout en garantissant une ligne de vue constante entre le phare et l'agent principal.

Bien que répondant aux besoins énoncés précédemment, il reste de nombreuses opportunités d'amélioration du processus proposé. Le premier me venant à l'esprit étant l'efficience de la solution logicielle proposée. Les temps d'exécutions sont raisonnables —quelques minutes au total pour une carte comme celle présentée en exemple— mais loin d'être optimaux. En effet, comme évoqué dans le chapitre précédent, de nombreux calculs sont liés à des opérations ensemblistes géométriques, et nous gagnerions certainement un temps considérable en optimisant cette partie, soit en préparant les géométries concernées avant opérations, ou en parallélisant les tâches, voire les deux.

Il serait également simple d'ajouter dans le processus de sélection du meilleur candidat local d'autres métriques, permettant de sélectionner un point peut être moins optimal, mais qui réponde à d'autres critères, comme la minimisation de la distance à parcourir par le phare. En acceptant une réduction de l'optimalité, nous pourrions ainsi adapter notre solution aux spéci-

ficités de la mission.

Un autre point d'évolution possible réside dans le test du processus proposé dans un panel plus large d'environnements. Nous n'avons en effet testé nos algorithmes que dans un nombre limités d'exemples à notre disposition. Bien que représentatifs des environnements auxquels nous sommes familiers —bureaux, maisons, bâtiments publics— ils ne le sont pas forcément d'environnement spécifiques comme les mines, les bâtiments industriels ou encore de milieux naturels comme les forêts. Une possibilité serait de générer des environnements de manière procédurale afin d'y évaluer la performance de notre système.

Conclusions et perspectives

1 Contexte

Ce travail de thèse a été effectué dans le cadre du projet européen GRone, dans lequel nous avons identifié l'après-mine comme un axe de recherche important pour le développement de l'utilisation des drones dans la grande région (Belgique, Luxembourg, Rhénanie, Grand-Est). L'après-mine qui désigne tous les travaux et études nécessaires à l'entretien des sites désaffectés, comprend un conséquent labeur de cartographie irréalisable au regard du nombre de sites à explorer. L'exploration et la cartographie autonome permettrait aux pouvoirs publics de disposer d'une meilleure image de la réalité du terrain, et ainsi de mettre en place les actions nécessaires à l'atténuation des risques pour la population.

L'exploration autonome par un système cyber-physique demande la mise en place de solutions de positionnement et de cartographie simultanés, classiquement appelés *simultaneous localization and mapping (SLAM)* dans la littérature. En l'absence de GPS, ces solutions exploitent majoritairement les données issues de caméras et de centrales inertielles, parfois de lidars. Ils obtiennent de bons résultats si l'environnement à explorer respecte certains critères. Or, notre contexte particulier d'opération —décrit dans la section 1.1 — est particulièrement exigeant. En effet, la majorité des mines qui ne sont plus en exploitation ne sont en effet pas équipées d'éclairage suffisants pour permettre l'utilisation de caméras. L'utilisation des solutions classiques basées sur des lidars, capables de percevoir l'environnement sans apport de lumière extérieure seraient envisageables si ce dernier n'était pas redondant. Or, il se trouve qu'une mine de calcaire, composées de piliers identiques, est extrêmement redondante (pensez aux mines de la *Moria* avec des dimensions humaines). De plus, ces solutions classiques de SLAM nécessitent un lourd paramétrage qui impacte radicalement leurs performances. Paramétrage qui dépend des particularités de l'environnement et ne peut être fait que par un spécialiste.

Nous avons donc identifié ce problème de SLAM, et plus particulièrement de positionnement comme le verrou théorique principal à l'exploration autonome de mines abandonnées. Le travail présenté a pour objectif principal la conception et la réalisation d'un système de SLAM nouveau ne reposant sur aucune hypothèse sur le milieu à explorer. Cette indépendance vis à vis de l'environnement est essentielle pour réaliser l'exploration autonome d'un milieu inconnu.

2 Contributions

C'est dans le contexte particulier de milieu inconnu que nous avons conduit nos travaux de recherche. En s'inspirant des techniques des géomètres, et tout particulièrement du réseau

géodésique français, il s'est avéré possible de concevoir un système de positionnement (chapitre 3) complètement indépendant de l'environnement. Ce dernier, basé sur un réseau de robots communiquants, permet en s'inspirant des phares d'effectuer des mesures coopératives de positions relatives. Cette collaboration active pour la mesure permet de supprimer totalement le problème d'association des données, qui impacte fortement les systèmes classiques de SLAM. Nous avons conçu, réalisé et évalué ce système de positionnement, travaux valorisés à travers une publication [Daugé et al., 2019]. Les résultats obtenus sont meilleurs que ceux de l'état de l'art, avec toutefois la contrainte d'utilisation de plusieurs robots. La pertinence de la solution réside néanmoins plus dans son indépendance totale vis à vis de l'environnement que dans la précision de ses estimations de position. La fréquence élevé ($250Hz$) permet également d'envisager une large gamme d'applications potentielles. Dernier élément essentiel de la solution proposée, l'accumulation d'erreur d'estimation de position ne se fait que lorsque le phare se déplace. Dans les systèmes classiques, cette accumulation se fait en permanence, conduisant à une importante dérive.

Afin de compléter l'approche de positionnent proposée tout en entérinant sa validité, nous avons intégré un lidar permettant de percevoir l'environnement. L'intégration de ce nouvel équipement ainsi que la mise en place du processus de traitement associé ont permis de réaliser un système de SLAM complet (chapitre 4). Les modèles obtenus en temps réel sont exploitables, bien que souffrant dans les cas défavorables des fortes erreurs en rotation. Le processus de traitement hors ligne proposé permet toutefois de corriger automatiquement ces erreurs au prix de nombreux calculs effectués après l'exploration. La réalisation manuelle des expériences de validation ne permet pas de profiter du plein potentiel du système. Il est en effet nécessaire d'optimiser le placement des différents éléments pour minimiser les erreurs de positionnement, et donc de reconstruction.

C'est dans l'objectif d'automatisation du système que deux contributions supplémentaires ont été réalisées. Premièrement, la détermination d'une trajectoire maximisant les données de l'environnement acquises en fonction d'une carte partielle —qui peut être générée en temps réel avec des algorithmes classiques— de l'environnement est proposée (chapitre 5). Deuxièmement, l'optimisation du placement des phares pour couvrir au mieux la mission permet de tirer pleinement partie du système de positionnement. La mise en place de ces algorithmes originaux lève les derniers verrous théoriques à l'emploi du système de manière totalement autonome.

La dernière des contributions effectuées au cours de cette thèse consiste à la valorisation du travail effectué par transfert technologique. Nous avons en effet collaboré avec la *SATT Sayens*[25] à travers une étude de marché, ainsi que des prises de contacts avec des entreprises intéressées. L'intérêt des entreprises nous a poussé à effectuer un dépôt du riche travail de développement effectué auprès de l'agence pour la protection des programmes[26]. Le brevetage des concepts et algorithmes proposés est également en cours, ce qui retarde la publication des travaux correspondants.

25. https://sayens.fr/
26. APP https://www.app.asso.fr/

3 Perspectives

3.1 Amélioration des propositions

Bien que les solutions de positionnement et de cartographie proposées soient fonctionnelles, de nombreuses améliorations peuvent y être apportées.

La qualité des reconstructions découlant directement de la précision du système de positionnement, les améliorations se concentrent sur ce dernier. Nous l'avions évoqué dans le chapitre 3, la portée limitée et le pilote propriétaire du matériel de positionnement local sont des limitations pratiques importantes. Un pilote alternatif de la bibliothèque open source *libsurvive* est déjà en place, mais la précision et la stabilité des résultats obtenus sont largement perfectibles. Fort de l'expérience acquise dans l'utilisation du matériel, contribuer directement à l'amélioration de cette bibliothèque serait une élégante façon de procéder. Cela permettrait de profiter du meilleur des deux mondes : la qualité du matériel commercial et la liberté de contrôle du système via un **pilote ouvert**.

Nous l'avons vu dans le chapitre 4 la réalisation d'un **meta-marqueur** permettrait une augmentation drastique de la précision des orientations fournies par le système de positionnement. Les outils théoriques nécessaires ont été identifiés, la réalisation d'un support mécanique de précision pour combiner des marqueur permettra rapidement de corriger les erreurs de rotations observées, tout en conservant les capacités temps-réel de la solution proposée.

3.2 Étendre le champ d'application

La limitation de portée, liée au matériel, semble être l'axe majeur d'amélioration du système identifié par les industriels interrogés. Le remplacement du matériel par du matériel spécialisé, conçu par des spécialistes de la métrologie serait idéal. Des discussions sont en cours afin d'établir un partenariat avec un géant de la métrologie. Une telle opération permettrait d'étendre largement les applications possibles de notre système de positionnement. Parmi ces dernières, le suivi de drones dans des environnement semi-ouverts deviendrait possible, en permettant au drone de voler plus haut —l'altitude est actuellement limitée par la portée du système. Une autre application industrielle identifiée réside dans le suivi de tombereaux dans les mines, afin de les rendre autonomes. Toutefois, la portée limitée du système ne correspond pas aux distances de freinage importantes de ces lourds engins de chantier, ne permettant pas à un élément de surveiller la zone de parking tout en étant à portée du tombereau à suivre. Enfin, la dernière application sollicitée rendue possible par une extension de la portée serait l'exploration de cavités accessibles par un forage vertical très étroit. Il serait alors envisageable de positionner un marqueur au fond de ce forage en une seule étape, conservant ainsi la précision de l'estimation de pose. Le reste du système pourrait par la suite être descendu pour effectuer l'exploration autonome de la cavité.

Des efforts d'ingénierie nous ont permis d'assurer l'autonomie énergétique du système. Toutefois, ce dernier n'est pas encore embarqué sur de vrais robots. Des travaux préliminaires nous ont permis de valider la faisabilité de l'embarquement du phare dans un robot chenillé, permettant un suivi stable et efficace énergiquement parlant. De plus, l'utilisation de drones miniatures ($\approx 10cm$) dotés de marqueurs accordera au système une flexibilité incroyable. Ces petits drones pourraient êtres utilisés comme références, supprimant la nécessité pour l'agent principal de marquer des arrêts. On peut également imaginer poser ces drones-marqueurs à des points d'intérêt

comme des carrefours pour effectuer des détections de boucles sûres. En effet, les algorithmes actuels de détection de boucle sont également soumis au problème d'association des données. Rencontrer un marqueur positionné précédemment permettrait de limiter fortement la dérive d'estimation sur toute la boucle effectuée entre le dépôt du drone-marqueur et sa récupération. Nous avons pour cela effectué des travaux préliminaires avec des *Crazyflyes*[27] de l'entreprise suédoise *Bitcraze*. Nous avons ainsi validé le positionnement de ces éléments avec la technologie locale de positionnement utilisée, validant ainsi la possibilité d'en effectuer le suivi avec notre système. Ainsi, ces drones-marqueurs pourraient effectuer bon nombre de missions —permises par leur taille limitée— tout en étant suivis précisément par notre système, leur donnant la capacité de venir se recharger sur le robot chenillé devenu porte drones.

Cette thèse s'est révélée être une expérience intense et très enrichissante, durant laquelle j'ai eu l'occasion d'effectuer de nombreux travaux de natures variées, allant de la mécanique à l'algorithmique, en passant par l'électronique. Nous avons réussi, en partant de zéro et avec des moyens limités, à concevoir, mettre en œuvre, et évaluer un système complet de SLAM. L'ensemble des verrous théoriques identifiés ont été levés et un effort conséquent de développement et de documentation du logiciel a été accompli. Les clés fonctionnelles nécessaires à la réalisation complète d'un système cyber-physique ont donc été apportées. L'aboutissement final du projet par la fabrication d'un prototype autonome sera facilité par les partenariats industriels suscités par le transfert technologique en cours. L'idée de voir le système prendre vie et ainsi concrétiser ces années de travail créée un sentiment de grande satisfaction.

27. Présentation des mini-drones https://www.bitcraze.io/products/crazyflie-2-1/

Résumé

Ce document retrace l'ensemble du travail et des contributions effectuées dans le cadre de l'exploration robotique autonome d'environnements inconnus. Une analyse de l'état de l'art nous a poussé à nous éloigner des paradigmes classiques des algorithmes de *positionnement et cartographie simultanés (SLAM en anglais)*. Ces approches nécessitent en effet un lourd paramétrage maîtrisable uniquement par des spécialistes. De plus, les systèmes actuels sont tous soumis au problème *d'association de données*. Le système de positionnement proposé s'inspire ainsi de techniques ancestrales de géomètres, ainsi que de la mesure en coopération active entre les bateaux et les phares. Ainsi, l'emploi de plusieurs robots mesurant collaborativement leurs positions relatives, formant ainsi un *réseau de positionnement* a permis de construire un système de positionnement robuste, indépendant de l'environnement. Ce travail présente l'évaluation du système, ainsi que son application à la cartographie, conduisant à la mise en place d'un système de SLAM complet. Des algorithmes originaux permettant l'optimisation du placement des éléments formant le réseau de positionnement dans l'espace sont également présentés. Le travail présenté a ainsi abouti à un prototype fonctionnel, testé et évalué dans des conditions réelles.

Mots-clés : Positionnement mutuel, cartographie, exploration autonome

Abstract

This document retraces all the work and contributions made in the context of autonomous robotic exploration of unknown environments. An analysis of the state of the art has led us to move away from the classical paradigms of *simultaneous positioning and mapping (SLAM)* algorithms. These approaches require a heavy parameterization that can only be mastered by specialists. In addition, current systems are all subject to the problem of *data association*. The proposed positioning system is inspired by ancestral techniques of techniques of surveyors, as well as the measurement in active cooperation between ships and lighthouses. Therefore, the use of several robots measuring their relative positions collaboratively, thus forming a *positioning network* has allowed to build a robust positioning system, independent of the environment. This work presents the evaluation of the system, as well as its application to cartography, leading to the implementation of a complete SLAM system. Several original algorithms allowing the optimization of the placement of elements forming the positioning network in space are also presented. The presented work has thus led to a functional prototype, tested and evaluated in real world.

Keywords : Mutual localization, mapping, SLAM, autonomous exploration

Bibliographie

[Arulampalam et al., 2002] Arulampalam, M. S., Maskell, S., Gordon, N., and Clapp, T. (2002). A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking. *IEEE Transactions on Signal Processing*, 50(2) :174–188.

[Babin et al., 2019] Babin, P., Giguere, P., and Pomerleau, F. (2019). Analysis of robust functions for registration algorithms. In *Proceedings - IEEE International Conference on Robotics and Automation*, volume 2019-May, pages 1451–1457.

[Bay et al., 2006] Bay, H., Tuytelaars, T., and Van Gool, L. (2006). SURF : Speeded up robust features. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 3951 LNCS, pages 404–417. Springer, Berlin, Heidelberg.

[Beis and Lowe, 1997] Beis, J. S. and Lowe, D. G. (1997). Shape indexing using approximate nearest-neighbour search in high-dimensional spaces. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1000–1006. IEEE.

[Besl and McKay, 1992] Besl, P. J. and McKay, N. D. (1992). A Method for Registration of 3-D Shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2) :239–256.

[Bingol and Krishnamurthy, 2019] Bingol, O. R. and Krishnamurthy, A. (2019). NURBS-Python : An open-source object-oriented NURBS modeling framework in Python. *SoftwareX*, 9 :85–94.

[Bradshaw et al., 2013] Bradshaw, J. M., Hoffman, R. R., Woods, D. D., and Johnson, M. (2013). The seven deadly myths of 'autonomous systems'. *IEEE Intelligent Systems*, 28(3) :54–61.

[Burri et al., 2016] Burri, M., Nikolic, J., Gohl, P., Schneider, T., Rehder, J., Omari, S., Achtelik, M. W., and Siegwart, R. (2016). The EuRoC micro aerial vehicle datasets. *International Journal of Robotics Research*, 35(10) :1157–1163.

[Calonder et al., 2010] Calonder, M., Lepetit, V., Strecha, C., and Fua, P. (2010). BRIEF : Binary robust independent elementary features. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 6314 LNCS, pages 778–792. Springer Verlag.

[Chebrolu et al., 2021] Chebrolu, N., Labe, T., Vysotska, O., Behley, J., and Stachniss, C. (2021). Adaptive Robust Kernels for Non-Linear Least Squares Problems. *IEEE Robotics and Automation Letters*, 6(2) :2240–2247.

[Chen and Medioni, 1991] Chen, Y. and Medioni, G. (1991). Object modeling by registration of multiple range images. In *Proceedings - IEEE International Conference on Robotics and Automation*, volume 3, pages 2724–2729. Publ by IEEE.

[Christian Franck, Romuald Salmon, Christophe Didier, Yves Paquette,] Christian Franck, Romuald Salmon, Christophe Didier, Yves Paquette, Z. P. Évaluation des aléas miniers. Technical report.

[Courant and Wigner, 1960] Courant, R. and Wigner, E. P. (1960). The Unreasonable Effectiveness of Mat hematics in the Natural Sciences. *COMMUNICATIONS ON PURE AND APPLIED MATHEMATICS*, XIII :1–14.

[Daugé et al., 2019] Daugé, V., Contassot-Vivier, S., and Ciarletta, L. (2019). NAPS : a Nomadic and Accurate Positioning System. In *Aerial Swarms/ IROS*.

[Dellaert and Kaess, 2006] Dellaert, F. and Kaess, M. (2006). Square root SAM : Simultaneous localization and mapping via square root information smoothing. In *International Journal of Robotics Research*, volume 25, pages 1181–1203.

[Delmerico and Scaramuzza, 2018] Delmerico, J. and Scaramuzza, D. (2018). A Benchmark Comparison of Monocular Visual-Inertial Odometry Algorithms for Flying Robots. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2502–2509. IEEE.

[der Walt et al., 2014] der Walt, S., Sch"onberger, J. L., Nunez-Iglesias, J., Boulogne, F., Warner, J. D., Yager, N., Gouillart, E., and Yu, T. (2014). scikit-image : image processing in Python. *PeerJ*, 2 :e453.

[Dijkstra, 1959] Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1) :269–271.

[Dorai et al., 1998] Dorai, C., Wang, G., Jain, A. K., and Mercer, C. (1998). Registration and integration of multiple object views for 3D model construction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(1) :83–89.

[Fischler and Bolles, 1981] Fischler, M. A. and Bolles, R. C. (1981). Random sample consensus : A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. *Communications of the ACM*, 24(6) :381–395.

[Fogel and Teillaud, 2015] Fogel, E. and Teillaud, M. (2015). The Computational Geometry Algorithms Library CGAL. *ACM Communications in Computer Algebra*, 49(1) :10–12.

[Förstner and Wrobel, 2016] Förstner, W. and Wrobel, B. P. (2016). *Photogrammetric Computer Vision*.

[Fredman et al., 1986] Fredman, M. L., Sedgewick, R., Sleator, D. D., and Tarjan, R. E. (1986). The pairing heap : A new form of self-adjusting heap. *Algorithmica*, 1(1-4) :111–129.

[Fredman and Tarjan, 1987] Fredman, M. L. and Tarjan, R. E. (1987). Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM (JACM)*, 34(3) :596–615.

[Geiger et al., 2012] Geiger, A., Lenz, P., and Urtasun, R. (2012). Are we ready for autonomous driving ? the KITTI vision benchmark suite. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 3354–3361.

[Gillies and Others,] Gillies, S. and Others. Shapely : manipulation and analysis of geometric objects.

[Godin et al., 1994] Godin, G., Rioux, M., and Baribeau, R. (1994). Three-dimensional registration using range and intensity information. In *Videometrics III*, volume 2350, pages 279–290. SPIE.

[Griffiths et al., 2010] Griffiths, T. L., Chater, N., Kemp, C., Perfors, A., and Tenenbaum, J. B. (2010). Approaches to cognitive modeling Probabilistic models of cognition : exploring representations and inductive biases. *Trends in Cognitive Sciences*, 14 :357–364.

[Harris and Stephens, 1988] Harris, C. and Stephens, M. (1988). A COMBINED CORNER AND EDGE DETECTOR. In *Alvey vision conference*.

[Horaud et al., 1989] Horaud, R., Conio, B., Leboulleux, O., and Lacolle, B. (1989). An analytic solution for the perspective 4-point problem. *Computer Vision, Graphics and Image Processing*, 47(1) :33–44.

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

[Huber, 1992] Huber, P. J. (1992). Robust Estimation of a Location Parameter. pages 492–518.

[Kaess et al., 2012] Kaess, M., Johannsson, H., Roberts, R., Ila, V., Leonard, J. J., and Dellaert, F. (2012). ISAM2 : Incremental smoothing and mapping using the Bayes tree. *International Journal of Robotics Research*, 31(2) :216–235.

[Kaess et al., 2008] Kaess, M., Ranganathan, A., and Dellaert, F. (2008). iSAM : Incremental smoothing and mapping. *IEEE Transactions on Robotics*, 24(6) :1365–1378.

[Kalman, 1960] Kalman, R. (1960). A new approach to linear filtering and prediction problems.

[LaValle, 1998] LaValle, S. M. (1998). Rapidly-exploring random trees : A new tool for path planning.

[Leutenegger et al., 2011] Leutenegger, S., Chli, M., and Siegwart, R. Y. (2011). BRISK : Binary Robust Invariant Scalable Keypoints. In *International Conference on Computer Vision*.

[Li et al., 2012] Li, S., Xu, C., and Xie, M. (2012). A robust O(n) solution to the perspective-n-point problem. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(7) :1444–1450.

[Loeliger, 2004] Loeliger, H. A. (2004). An introduction to factor graphs. *IEEE Signal Processing Magazine*, 21(1) :28–41.

[Lorensen and Cline, 1987] Lorensen, W. E. and Cline, H. E. (1987). Marching cubes : A high resolution 3D surface construction algorithm. Technical Report 4.

[Lowe, 2004] Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2) :91–110.

[Madkour et al., 2017] Madkour, A., Aref, W. G., Ur Rehman, F., Rahman, M. A., and Basalamah, S. (2017). A survey of shortest-path algorithms.

[Masuda et al., 1996] Masuda, T., Sakaue, K., and Yokoya, N. (1996). Registration and integration of multiple range images for 3-D model construction. In *Proceedings - International Conference on Pattern Recognition*, volume 1, pages 879–883. Institute of Electrical and Electronics Engineers Inc.

[Mazhar et al., 2017] Mazhar, F., Khan, M. G., and Sällberg, B. (2017). Precise Indoor Positioning Using UWB : A Review of Methods, Algorithms and Implementations.

[Nistér, 2004] Nistér, D. (2004). An efficient solution to the five-point relative pose problem. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(6) :756–770.

[Nistér and Stewénius, 2006] Nistér, D. and Stewénius, H. (2006). Scalable recognition with a vocabulary tree. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2, pages 2161–2168.

[Piegl, 1991] Piegl, L. (1991). On NURBS : A Survey. *IEEE Computer Graphics and Applications*, 11(1) :55–71.

[Pomerleau et al., 2013] Pomerleau, F., Colas, F., Siegwart, R., and Magnenat, S. (2013). Comparing ICP variants on real-world data sets. *Autonomous Robots*, 34(3) :133–148.

[Qin et al., 2017] Qin, T., Li, P., Yang, Z., and Shen, S. (2017). Technical Report : VINS-Mono : A Robust and Versatile Monocular Visual-Inertial State Estimator. 34(4) :1–14.

[Rosten and Drummond, 2006] Rosten, E. and Drummond, T. (2006). Machine learning for high-speed corner detection. In *European Conference on Computer Vision*.

[Rublee et al., 2011] Rublee, E., Rabaud, V., Konolige, K., and Bradski, G. (2011). ORB : An efficient alternative to SIFT or SURF. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2564–2571.

[Rusinkiewicz and Levoy, 2001] Rusinkiewicz, S. and Levoy, M. (2001). Efficient variants of the ICP algorithm. *Proceedings of International Conference on 3-D Digital Imaging and Modeling, 3DIM*, pages 145–152.

[Russell Ackoff, 1989] Russell Ackoff (1989). From Data to Wisdom. Technical report.

[Sanneman et al., 2015] Sanneman, L., Ajilo, D., DelPreto, J., Mehta, A., Miyashita, S., Poorheravi, N. A., Ramirez, C., Sehyuk, Y., Kim, S., and Rus, D. (2015). A Distributed Robot Garden System. *Proceedings - IEEE International Conference on Robotics and Automation*, 2015-June(June) :6120–6127.

[Schoenberg, 1946] Schoenberg, I. J. (1946). Contributions to the problem of approximation of equidistant data by analytic functions. Part B. On the problem of osculatory interpolation. A second class of analytic approximation formulae. *Quarterly of Applied Mathematics*, 4(2) :112–141.

[Schubert et al., 2018] Schubert, D., Goll, T., Demmel, N., Usenko, V., Stuckler, J., and Cremers, D. (2018). The TUM VI Benchmark for Evaluating Visual-Inertial Odometry. *IEEE International Conference on Intelligent Robots and Systems*, pages 1680–1687.

[Segal et al., 2009] Segal, A. V., Haehnel, D., and Thrun, S. (2009). Generalized-ICP. *Robotics : science and systems*, 2(4) :435.

[Shao et al., 2019] Shao, W., Vijayarangan, S., Li, C., and Kantor, G. (2019). Stereo Visual Inertial LiDAR Simultaneous Localization and Mapping. In *IEEE International Conference on Intelligent Robots and Systems*, pages 370–377. Institute of Electrical and Electronics Engineers Inc.

[Shepard, 1987] Shepard, R. N. (1987). Toward a universal law of generalization for psychological science. *Science*, 237(4820) :1317–1323.

[Shi and Tomasi, 1994] Shi, J. and Tomasi, C. (1994). Good features to track. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 593–600. Publ by IEEE.

[Simon, 1996] Simon, D. A. (1996). *Fast and Accurate Shape-Based Registration*. PhD thesis.

[Sims, 2018] Sims, C. R. (2018). Efficient coding explains the universal law of generalization in human perception. *Science*, 360(6389) :652–656.

[Thorup, 1999] Thorup, M. (1999). Undirected single-source shortest paths with positive integer weights in linear time. *Journal of the ACM*, 46(3) :362–394.

[Thrun, 2002] Thrun, S. (2002). Robotic Mapping : A Survey. *Science*, 298(February) :1–35.

[Thrun, 2003] Thrun, S. (2003). Learning occupancy grid maps with forward sensor models. *Autonomous Robots*, 15(2) :111–127.

[Turk and Levoy, 1994] Turk, G. and Levoy, M. (1994). Zippered polygon meshes from Range images. In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1994*, pages 311–318. Association for Computing Machinery, Inc.

[Valiant, 2013] Valiant, L. (2013). *Probably Approximately Correct : Nature's Algorithms for Learning and Prospering in a Complex World*.

[Van der Merwe and Wan, 2003] Van der Merwe, R. and Wan, E. (2003). Gaussian mixture Sigma-Point particle filters for sequential probabilistic inference in dynamic state-space models. In *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, volume 6, pages 701–704.

[Wan and Van Der Merwe, 2000] Wan, E. A. and Van Der Merwe, R. (2000). The unscented Kalman filter for nonlinear estimation. In *IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium, AS-SPCC 2000*, number 3, pages 153–158.

[Wanderley et al., 2019] Wanderley, F., Misra, A., and Singh, V. B. (2019). Localization Techniques in the Future IoT : A Review. In *Proceedings - 2019 19th International Conference on Computational Science and Its Applications, ICCSA 2019*, pages 57–64. Institute of Electrical and Electronics Engineers Inc.

[Weik, 1997] Weik, S. (1997). Registration of 3-D partial surface models using luminance and depth information. In *Proceedings of the International Conference on Recent Advances in 3-D Digital Imaging and Modeling*, pages 93–100. IEEE.

[Yang and Medioni, 1992] Yang, C. and Medioni, G. (1992). Object modelling by registration of multiple range images. *Image and Vision Computing*, 10(3) :145–155.

[Zhang and Scaramuzza, 2018] Zhang, Z. and Scaramuzza, D. (2018). A Tutorial on Quantitative Trajectory Evaluation for Visual(-Inertial) Odometry. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7244–7251. IEEE.