Placement interactif de capteurs mobiles dans des environnements tridimensionnels non convexes

Thèse

François-Michel De Rainville

Doctorat en génie électrique
Philosophiæ doctor (Ph.D.)

Québec, Canada

Résumé

La présente thèse propose un système complet de placement de capteurs mobiles dans un environnement pleinement tridimensionnel et préalablement inconnu. Les capteurs mobiles sont des capteurs placés sur des unités robotiques autonomes, soit des véhicules possédant une unité de calcul et pouvant se déplacer dans l'environnement. Le placement de capteur est fondé sur une vue désirée par un utilisateur du système nommé vue virtuelle. La vue virtuelle est contrôlée à distance en changeant les paramètres intrinsèques et extrinsèques du capteur virtuel, soit sa position, sa résolution, son champ de vue, etc. Le capteur virtuel n'est alors soumis à aucune contrainte physique, par exemple il peut être placé à n'importe quelle hauteur dans l'environnement et avoir un champ de vue et une résolution arbitrairement grande. Les capteurs mobiles (réels) ont pour tâche de récupérer toute l'information contenue dans le point de vue virtuel. Ce n'est qu'en combinant leur capacité sensorielle que les capteurs mobiles pourront capter l'information demandée par l'utilisateur.

Tout d'abord, cette thèse s'attaque au problème de placement de capteurs en définissant une fonction de visibilité servant à évaluer le positionnement d'un groupe de capteurs dans l'environnement. La fonction de visibilité développée est applicable aux environnements tridimensionnels et se base sur le principe de ligne de vue directe entre un capteur et la cible. De plus, la fonction prend en compte la densité d'échantillonnage des capteurs afin de reproduire la densité désirée indiquée par le capteur virtuel. Ensuite, ce travail propose l'utilisation d'un modèle de l'environnement pleinement tridimensionnel et pouvant être construit de manière incrémentale, rendant son utilisation possible dans un environnement tridimensionnel non convexe préalablement inconnu. Puis, un algorithme d'optimisation coopératif est présenté afin de trouver simultanément le nombre de capteurs et leur positionnement respectif afin d'acquérir l'information contenue dans la vue virtuelle. Finalement, la thèse démontre expérimentalement dans diverses conditions que le système proposé est supérieur à l'état de l'art pour le placement de capteurs dans le but d'observer une scène bidimensionnelle. Il est aussi établi expérimentalement en simulation et en réalité que les performances se transposent à l'observation d'environnements tridimensionnels non convexes préalablement inconnus.

Abstract

This Thesis proposes a novel mobile sensor placement system working in initially unknown three dimensional environment. The mobile sensors are fix sensors placed on autonomous robots, which are ground and aerial vehicles equipped with computing units. The sensor placement is based on a user-defined view, named the virtual view. This view is manipulated through a virtual sensor intrinsic and extrinsic parameters, such as its position, orientation, field of view, resolution, etc. The virtual sensor is not subject to any physical constraint, for example it can be place where no sensor could be or it possess an arbitrary large field of view and resolution. The mobile (real) sensors have to acquire the entire information contained in this virtual view. It is only by combining the sensory capacity of an unknown number of sensors that they can acquire the necessary information.

First, this Thesis addresses the sensor placement problem by defining a visibility function to qualify a group of sensor configurations in the environment. This function is applicable to three dimensional environments and is based on direct line of sight principle, where we compute the sensor sampling density in its visibility region. Then, this Thesis proposes the use of an incrementally built model of the environment containing all the information needed by the objective function. Next, a cooperative optimization algorithm is put forward to simultaneously find the number of sensors and their respective position required to capture all the information in the virtual view. Finally, the proposed system is experimentally shown to use less sensor to acquire the scene of interest at a higher resolution than state of the art methods in initially known two dimensional environments. It is also shown in simulation and practice that the performance of the system can be transposed to initially unknown non-convex three dimensional environments.

Table des matières

Résumé iii

Abstract v

Table des matières vii

Liste des tableaux xi

Liste des figures xiii

Liste des abréviations xv

1 Introduction 1
- 1.1 Motivation . . . 2
- 1.2 Problème . . . 2
  - 1.2.1 Évaluation de la couverture . . . 3
  - 1.2.2 Optimisation des capteurs . . . 4
  - 1.2.3 Déploiement du réseau de capteurs . . . 5
- 1.3 Travaux connexes . . . 5
  - 1.3.1 Réseaux robotiques . . . 5
  - 1.3.2 Prochaine meilleure vue . . . 7
- 1.4 Solution proposée . . . 9
  - 1.4.1 Qualité d'une configuration de capteurs . . . 10
  - 1.4.2 Optimisation de la configuration de capteurs . . . 10
  - 1.4.3 Déploiement du système . . . 11

I Sous systèmes et intégration 13

2 Qualité du placement de capteurs 15
- 2.1 Qualité d'acquisition d'un capteur . . . 16
- 2.2 Représentation . . . 19
  - 2.2.1 Grille d'occupation . . . 20
  - 2.2.2 Marching cubes . . . 25
- 2.3 Fonction objectif . . . 26
  - 2.3.1 Scène d'intérêt . . . 27

2.3.2 Fonction d'erreur . . . 27
2.3.3 Combinaison des qualités d'acquisition . . . 28
2.3.4 Résolution numérique . . . 29
2.4 Conclusion . . . 30

3 Optimisation **31**
3.1 Stratégies d'évolution . . . 32
3.2 Coévolution coopérative . . . 34
3.3 Coévolution coopérative efficiente . . . 36
3.3.1 Bandit manchot . . . 37
3.3.2 Sélection d'une espèce en coévolution coopérative . . . 40
3.3.3 Résultats préliminaires . . . 43
3.3.4 Conclusions préliminaires . . . 50
3.4 Coévolution efficiente avec stratégie d'évolution à matrice de covariance adaptée . . . 52
3.5 Conclusion . . . 54

4 Intégration du système **55**
4.1 Localisation . . . 56
4.1.1 Localisation avec carte . . . 57
4.1.2 Localisation et cartographie simultanées . . . 59
4.2 Navigation . . . 60
4.2.1 Planification de la trajectoire . . . 61
4.2.2 Évitement de collisions interrobots . . . 63
4.2.3 Navigation des robots aériens . . . 64
4.3 Optimisation . . . 64
4.3.1 Représentation d'une solution . . . 65
4.3.2 Fonction biobjectif et classement . . . 66
4.3.3 Stagnation et contribution . . . 67
4.3.4 Banque de capteurs et initialisation . . . 68
4.4 Reconstruction de la vue virtuelle . . . 69
4.5 Conclusion . . . 70

II Expérimentations et résultats **73**

5 Simulations **75**
5.1 Simulations bidimensionnelles . . . 75
5.1.1 Nombre de capteurs fixe . . . 77
5.1.2 Adaptation du nombre de capteurs . . . 80
5.2 Simulations tridimensionnelles . . . 83
5.2.1 Rover enlisé . . . 86
5.2.2 Station spatiale . . . 88
5.2.3 Bâtiment endommagé . . . 90
5.2.4 Musée . . . 95

5.3 Conclusion 99

6 Expérimentations réelles **101**
6.1 Mosaïque 102
6.2 Porte de voiture 104
6.3 Conclusion 106

7 Conclusion **109**
7.1 Contributions 110
7.1.1 Qualité d'une configuration de capteurs 110
7.1.2 Algorithme d'optimisation coopérative 111
7.1.3 Système complet de placement de capteurs 111
7.1.4 Outils logiciels d'algorithmes évolutionnaires 112
7.2 Perspectives 114
7.3 Impact 115

Bibliographie **117**

A Détails d'implémentation **127**
A.1 Sélection des points d'intérêt 127
A.2 Point de contact dans l'octree 127
A.3 Discrétisation du modèle d'environnement 128
A.4 Filtres appliqués aux données 3D 129
A.5 Prévention de l'observation des robots 129

B Liste des publications **131**
B.1 Revues scientifiques 131
B.2 Conférences 131
B.3 Autres 132

Liste des tableaux

3.1 Paramètres utilisés pour l'appariement de chaînes. 45

5.1 Moyenne des résultats des simulations bidimensionnelles avec nombre de capteurs fixe. 77
5.2 Moyenne de l'erreur normalisée de densité pixels pour les expérimentations avec un nombre variable de capteurs et trois densités de pixels désirées. . . . 82
5.3 Paramètres des algorithmes d'optimisation pour les simulations tridimensionnelles. 85

Liste des figures

1.1 Désignation de la scène d'intérêt par le capteur virtuel. 3

2.1 Pyramide de visibilité et empreinte d'un capteur caméra sténopé. 16
2.2 Géométrie de l'empreinte d'un capteur pour une cible d'intérêt. 17
2.3 Densité de pixels par unité d'aire calculée pour un environnement simple. . . 19
2.4 Grille d'occupation contenant trois objets. 21
2.5 Décomposition de l'espace tridimensionnel en octree. 21
2.6 Modélisation de l'environnement dans une grille d'occupation. 22
2.7 Les 15 cas fondamentaux de l'algorithme « marching cubes ». 25
2.8 Calcul des surfaces à partir de l'algorithme des marching cubes. 26
2.9 Procédure de projection-rétroprojection. 29

3.1 Aires sous la courbe pour une série d'actions. 39
3.2 Appariement de chaque patron pour la configuration 1. 46
3.3 Appariement de chaque patron pour la configuration 2. 48
3.4 Histogramme du nombre de générations requis pour atteindre l'appariement parfait pour la configuration 1. 49
3.5 Histogramme du nombre de générations requis pour atteindre l'appariement parfait pour la configuration 2. 49
3.6 Résultat médian de l'appariement de chaînes avec la configuration 2. 51

4.1 Intégrations du système proposé. 56
4.2 Localisation Monte-Carlo adaptative. 58
4.3 Cartographie SLAM. 60
4.4 Trajectoires simulées par l'algorithme DWA. 62
4.5 Placement d'un capteur lors de l'envoi d'une position trop près d'un obstacle. 62
4.6 Problème engendré par le dégagement du capteur de télémétrie. 63
4.7 Assemblage d'images de la mosaïque du pavillon Adrien-Pouliot. 71

5.1 Polygone à observer lors des simulations bidimensionnelles. 76
5.2 Erreur densité de pixel du scénario bidimensionnel. 78
5.3 Aire par pixel du scénario bidimensionnel. 79
5.4 Configurations finales lors de l'observation de la scène d'intérêt bidimensionnelle par un nombre fixe de capteurs. 81
5.5 Configurations finales médianes lors de l'observation de la scène d'intérêt bidimensionnelle par un nombre de capteurs et une densité désirée variables. . 84

5.6 Scénario d'inspection d'un astromobile enlisé. 85
5.7 Optimisation du positionnement des robots dans le scénario du rover enlisé. . 87
5.8 Moyenne de l'erreur normalisée de densité de pixels minimale par nombre de capteurs pour le scénario du rover enlisé. 88
5.9 Source de l'erreur constante dans la simulation du rover enlisé. 88
5.10 Images acquises par les quatre Huskys de l'expérimentation du véhicule enlisé. 89
5.11 Scénario d'inspection d'un panneau solaire de la Station spatiale internationale. 89
5.12 Optimisation du positionnement des caméras autour de la station spatiale. . . 91
5.13 Moyenne de l'erreur normalisée de densité de pixels minimale par nombre de capteurs pour le scénario de la station spatiale. 92
5.14 Scénario d'inspection d'une poutre à l'intérieur d'un bâtiment endommagé. . 92
5.15 Optimisation du positionnement des caméras dans le bâtiment endommagé. . 94
5.16 Moyenne de l'erreur normalisée de densité de pixels minimale par nombre de capteurs pour le scénario du bâtiment endommagé. 95
5.17 Source de l'erreur constante dans la simulation du bâtiment endommagé. . . . 96
5.18 Scénario de visite virtuelle d'un musée. 96
5.19 Optimisation du positionnement des caméras dans le musée. 97
5.20 Moyenne de l'erreur normalisée de densité de pixels minimale par nombre de capteurs pour le scénario du musée. 98
5.21 Vue acquises pour les expérimentations du musée. 99

6.1 Scénario d'inspection d'une mosaïque. 102
6.2 Moyenne de l'erreur normalisée de densité de pixels au cours de l'optimisation du placement des capteurs pour l'expérimentation de la mosaïque. 103
6.3 Configuration finale des robots pour l'observation de la mosaïque. 103
6.4 Assemblage des images prises par les quatre robots placés à leur position finale lors de l'inspection de la mosaïque. 104
6.5 Scénario d'inspection d'une porte de voiture. 104
6.6 Moyenne de l'erreur normalisée de densité de pixels au cours de l'optimisation du placement des capteurs pour l'expérimentation de la porte de voiture. . 105
6.7 Configuration finale obtenue pour l'observation de la porte de voiture. 105
6.8 Assemblage d'images pour les expérimentations de la porte de voiture. 106

A.1 Erreur due aux angles de rétroprojection rasants. 128
A.2 Erreur causée par la vue partielle d'un voxel. 128

Liste des abréviations

$\mathbf{c}_v$	Capteur virtuel, page 2
$\mathbf{C}$	Matrice de covariance servant à la génération de solutions candidates, page 33
$\mathbf{c}$	Vecteur de paramètres d'un capteurs réels, page 4
$\mathbf{h}$	Représentant d'une espèce lors de la coévolution coopérative, page 35
$\mathbf{x}$	Solution candidate à l'optimisation, page 33
C	Ensemble de capteurs réels, page 4
$\mathcal{E}$	Empreinte d'un capteur, page 16
$\mathcal{I}$	Cible d'intérêt telle que désignée par le capteur virtuel, page 2
$\mathcal{P}$	Espace de paramètres d'un capteur réels, page 4
$\mathcal{Q}$	Environnement dans lequel travaillent les robots, page 2
$\mathcal{V}$	Pyramide de visualisation d'un capteur, page 16
$\mathfrak{H}$	Ensemble des représentants des espèces lors de la coévolution coopérative, page 35
$\mathfrak{P}$	Population de solutions candidates à l'optimisation, page 32
$\mathfrak{S}$	Espèce participant à la coévolution coopérative, page 35
∂e	Fraction d'empreinte d'un capteur, page 17
σ	Pas d'optimisation, page 33
C	Facteur d'exploration de l'algorithme de sélection d'une action du bandit manchot, page 40
D	Amortissement de la récompense de l'algorithme AUC, page 38
N	Nombre de capteurs disponibles, page 4

T_a	Seuil d'amélioration, page 36
T_c	Seuil de contribution, page 36
T_l	Délai d'amélioration, page 36
W	Longueur de la fenêtre de récompenses de l'algorithme AUC, page 38
ACC	Algorithme de coévolution coopérative, page 75
ACC-MAB	Algorithme de coévolution coopérative efficiente, page 75
AMCL	Localisation adaptative Monte-Carlo, page 57
APP	Aire par pixel, page 79
AUC	Aire sous la courbe, page 38
CGD	Contrôleur à gradient décentralisé, page 75
CMA-ES	Stratégie d'évolution à matrice de covariance adaptée, page 33
DEAP	Outil logiciel d'algorithmes évolutionnaires distribués en Python, page 69
DWA	Planification de trajectoire par l'approche de fenêtre dynamique, page 61
ENDP	Erreur normalisée de densité de pixels, page 28
Global	Algorithme de placement de capteurs global, page 76
MAB	Bandit manchot, page 37
PID	Régulateur proportionnel intégral dérivé, page 64
SLAM	Localisation et cartographie simultanées, page 59
UCB	Algorithme de borne supérieure de confiance, page 37
Vorace	Algorithme de placement de capteurs vorace, page 76
WSVD	Décomposition en valeurs singulières pondérée, page 70

À Ariane et Thomas

Les Trois Lois de la robotique :

1. Un robot ne peut porter atteinte à un être humain, ni, restant passif, permettre qu'un être humain soit exposé au danger.
2. Un robot doit obéir aux ordres que lui donne un être humain, sauf si de tels ordres entrent en conflit avec la Première Loi.
3. Un robot doit protéger son existence tant que cette protection n'entre pas en conflit avec la Première ou la Deuxième Loi.

Isaac Asimov

Remerciements

Cette thèse origine d'une collaboration antérieure à mon passage au doctorat. En effet, c'est lors de mon passage à la maîtrise que j'ai eu l'occasion de rencontrer le professeur Christian Gagné qui a piqué mon intérêt pour la recherche sur les algorithmes évolutionnaires. Il m'a ensuite orienté avec l'aide du professeur Denis Laurendeau sur un projet hors du commun combinant la robotique et cette dernière discipline. Je souhaite remercier mes deux extraordinaires directeurs qui ont joué un rôle déterminant dans la réussite de ce projet. Ils m'ont consacré un temps et une énergie considérable qui m'a permis de réaliser ce document.

J'ai vivement apprécié collaborer avec plusieurs étudiants du laboratoire de vision et systèmes numériques, parmi eux, Audrey Durand, Marc-André Gardner, Yannick Hold-Geoffroy, Julien-Charles Lévesque, Kevin Tanguy, Maxime Tremblay et plus spécialement mon collègue de bureau durant ces sept années qu'ont duré mes études graduées et estimé ami Félix-Antoine Fortin. Je suis grandement reconnaissant de la collaboration du professeur Philippe Giguère lors de l'écriture d'un article portant sur les travaux et de l'implication de Jean-Philippe Mercier lors des expérimentations avec les robots réels. Je souhaite aussi remercier tout le personnel du laboratoire de vision systèmes numériques de m'avoir soutenu lors de mes nombreuses expérimentations.

J'ai eu la chance de faire un stage international à l'INRIA Saclay-Île-de-France. Je désire donc remercier tous les gens qui m'y ont accueilli et avec qui j'ai pu partager mes idées, en particulier les chercheurs Marc Schoenauer et Michèle Sebag qui m'ont ouvert toutes grandes les portes de leur laboratoire.

Cette thèse n'aurait jamais vu le jour sans l'aide précieuse de ma conjointe Ariane Gagnon-Rocque. En plus de sont support moral et de sa patience, elle a su m'aider dans la révision de ce document. Finalement, je souhaite remercier mes parents Christine Bernier et Pierre De Rainville qui m'ont encouragé et soutenu tout au long de mes études. Pour m'avoir transmis leur curiosité et leur persévérance, je leur dois la plus grande partie de mes succès.

Chapitre 1

Introduction

Les systèmes de réseaux de capteurs prennent de plus en plus de place dans plusieurs domaines. Ils permettent, notamment, une observation centralisée et efficace d'édifices et d'endroits publics. La grande majorité de ces systèmes sont composés de capteurs fixes disposés préalablement dans l'environnement pour observer efficacement une surface (ou un volume) tout en minimisant les coûts de déploiement et en tenant compte la géométrie de l'environnement.

Malheureusement, l'utilisation de capteurs fixes n'est pas toujours possible et, si elle l'est, la couverture de l'environnement n'est pas toujours optimale. On n'a qu'à penser à un environnement dynamique, où les capteurs peuvent avoir la vue voilée à tout moment ou encore lorsque ce qui doit être observé peut se déplacer. Dans ces cas, l'utilisation d'un réseau de capteurs mobiles permet une meilleure observation, et ce, malgré les variations de l'environnement. On pense, par exemple, à une visite virtuelle en temps réel d'un lieu touristique, où les capteurs auraient à se reconfigurer lorsque le visiteur veut explorer un endroit différent de l'environnement. Les capteurs mobiles permettent aussi l'exploration et l'observation de milieux qui ne sont pas accessibles à un humain ou dont la configuration a dramatiquement changé à la suite d'un événement externe comme une catastrophe naturelle. Par exemple, après un tremblement de terre, une flotte de capteurs mobiles peut être déployée dans le but de localiser des survivants à l'intérieur d'un édifice en équilibre précaire ou encore repérer les endroits où la structure devrait être renforcée afin d'éviter que les sauveteurs risquent leur vie. Ces exemples ne sont qu'un éventail restreint de toutes les possibilités d'utilisation d'une flotte de capteurs mobiles.

De plus, la robustesse, la flexibilité et l'extensibilité des systèmes multirobots (Şahin, 2005) les rendent parfaitement adaptés aux applications énumérées précédemment. En outre, l'uti-

lisation de multiple robots relativement simples permet généralement la réalisation de tâches d'un seul robot sophistiqué ne peut accomplir (Dudek et collab., 1996). Leur avantage vient à la fois de leur nombre et de leur robustesse. Étant physiquement plus simples, ils sont généralement moins coûteux et peuvent former un essaim de taille appréciable pour le même prix qu'un robot plus complexe. Leur simplicité réduit nécessairement les risques de défaillance mécanique et leur nombre permet une redondance en cas de panne. L'indépendance des individus d'un groupe de robots est aussi une caractéristique estimable. Elle permet, entre autres, d'augmenter radicalement l'espace de travail du système. Là ou un seul robot est limité par sa portée, le groupe de robots peut se disperser avec comme seules contraintes la structure de l'environnement et la portée des dispositifs de communication.

1.1 Motivation

L'objectif de ce travail est de proposer un système de placement d'une flotte de capteurs mobiles (à savoir, des capteurs fixes montés sur des robots mobiles relativement simples) permettant la téléprésence d'un opérateur dans une scène difficilement accessible pour un humain. Par exemple, nous visons des applications ou un groupe de robots relativement simples seraient déployés afin d'inspecter, selon le désir de l'opérateur, un environnement dangereux ou distant, comme une centrale nucléaire (ex. Fukushima) ou un site dans l'espace (ex. Mars).

Nous voulons aussi créer un paradigme de manipulation de la scène à observer facilitant la tâche de l'utilisateur dans son inspection de l'environnement. Concrètement, il serait impensable de déléguer à l'opérateur le choix du positionnement de tous les capteurs. L'utilisation d'un tel paradigme doit permettre non seulement de déplacer la zone d'intérêt à la guise de l'opérateur, mais aussi d'en changer aisément les paramètres comme la résolution, la taille, etc. Le système produirait alors une configuration de capteurs respectant les exigences imposées par les paramètres de la zone d'intérêt.

1.2 Problème

Cette thèse propose d'utiliser une flotte de capteurs mobiles afin d'observer une scène d'intérêt $\mathcal{I}$ dans un environnement tridimensionnel non convexe $\mathcal{Q}$ telle qu'elle serait perçue si un observateur était sur les lieux. La vue désirée par l'utilisateur est nommée *vue virtuelle*. Elle est obtenue en manipulant à distance un *capteur virtuel* $\mathbf{c}_{\mathrm{v}}$ tel qu'illustré à la figure 1.1. Le capteur virtuel est une entité similaire à un capteur physique, il possède des paramètres intrinsèques et extrinsèques définissant sa résolution, son champ de vue et sa position. Tou-

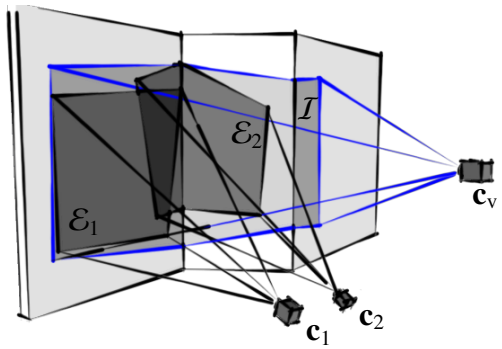


FIGURE 1.1 – Désignation de la scène d'intérêt $\mathcal{I}$ par le placement d'un capteur virtuel $\mathbf{c}_v$ dans l'environnement et placement des capteurs réels $\mathbf{c}_1$ et $\mathbf{c}_2$ pour acquérir l'information contenue dans $\mathcal{I}$ en la couvrant de leur empreinte $\mathcal{E}_1$ et $\mathcal{E}_2$.

tefois, il n'est pas limité par les contraintes physiques habituelles. En effet, le capteur virtuel peut, par exemple, être placé là où aucun autre capteur ne pourrait l'être et posséder des propriétés totalement hors du commun comme un très grand angle de vue, une très haute résolution et même être omniscient tel qu'il peut voir toutes les surfaces d'un environnement sans contrainte de visibilité directe.

En d'autres termes, le problème abordé dans cette thèse est de positionner un nombre limité de capteurs afin de percevoir toute l'information souhaitée dans la vue virtuelle. Nous voulons aussi capter cette information avec une qualité d'acquisition similaire à celle définie par le capteur virtuel. Puisque nous sommes intéressés à l'inspection visuelle, nous définissons cette qualité d'acquisition en termes de densité de pixels couvrant les surfaces de la scène d'intérêt. Dans de cette thèse, nous nous limiterons à cette définition de la qualité d'acquisition. Cependant, il est important de noter que toute autre mesure de précision pourrait remplacer la densité de pixels dans le système proposé.

Dans cette section, nous décomposons le problème de placement de capteurs en trois parties qui ont fait chacune l'objet de recherches. Tout d'abord, nous définissons le problème de l'évaluation de la couverture des capteurs dans un espace tridimensionnel. Puis, nous formulons ce problème sous la forme d'un problème d'optimisation. Enfin, nous explorons les points importants à traiter au niveau du système de déploiement de capteurs.

1.2.1 Évaluation de la couverture

Le problème de couverture abordé dans cette thèse est d'observer avec une densité de pixels donnée et un nombre limité de capteurs une partie d'un environnement tridimensionnel $Q \subset \mathbb{R}^3$ appelé scène d'intérêt $\mathcal{I} \subseteq Q$. Nous ne posons aucune hypothèse sur la géométrie de

Q, c'est-à-dire que sa topographie tridimensionnelle peut engendrer des occlusions. Cette caractéristique est centrale à l'évaluation de la couverture d'un capteur puisqu'elle implique nécessairement l'évaluation de sa visibilité. Le premier problème consiste donc à élaborer une méthode permettant de mesurer, dans un environnement tridimensionnel non convexe, la densité de pixels produite par un capteur couvrant les surfaces de l'environnement en tenant compte de sa visibilité.

Une fois la visibilité d'un capteur évaluée et sa densité d'échantillonnage calculée, un second problème persiste. Ces valeurs doivent être combinées en une mesure homogène même si le groupe de capteurs réels $C = \{\mathbf{c}_1, \ldots, \mathbf{c}_n\}$ est hétérogène. En effet, nous ne faisons pas non plus d'hypothèses sur l'homogénéité du groupe de capteurs, c'est-à-dire que chaque capteur $\mathbf{c}_i$ peut avoir un espace des paramètres $\mathcal{P}_i = \mathbb{R}^{s_i}$ défini en un nombre de dimensions s_i différent. La combinaison des densités de pixels et la mesure d'erreur entre le groupe de capteur et le capteur virtuel devra donc faire abstraction du type des capteurs.

1.2.2 Optimisation des capteurs

Lorsque l'évaluation de la couverture d'un réseau de capteurs C est faite, le problème se formule alors en problème d'optimisation. Ainsi, on cherche à trouver la configuration de capteurs C^* parmi l'ensemble des configurations potentielles reproduisant le plus fidèlement possible la densité de pixels qu'aurait un capteur virtuel $\mathbf{c}_\mathrm{v}$ sur l'ensemble de son champ de vue. Ceci revient donc à résoudre le problème suivant

$$C^* = \arg\min_{C \in \mathbb{C}} \int_{\mathcal{I}} f_\mathrm{q}(\mathbf{c}_\mathrm{v}, C, \mathbf{i}) \, \mathrm{d}\mathbf{i}, \tag{1.1}$$

où, $\mathbb{C} = \prod_{i=1}^{n} \mathcal{P}_i$ est l'espace des capteurs et $f_\mathrm{q}(\mathbf{c}_\mathrm{v}, C, \mathbf{i})$ est une fonction donnant la qualité de la configuration du réseau de capteurs C étant donné le capteur virtuel $\mathbf{c}_\mathrm{v}$ en tous points d'intérêt $\mathbf{i}$.

Intrinsèquement à ce problème d'optimisation se pose la sélection d'un nombre adéquat de capteurs n. Une solution dégénérée à l'équation 1.1 survient lorsque $n = \infty$, car la différence de densités est nécessairement minimale. De toute évidence, il est impossible de placer un nombre infini de capteurs. En fait, il est même préférable de minimiser le nombre de capteurs utilisés même lorsque le nombre disponible N est borné. En effet, lorsque $n < N$, les capteurs inutilisés peuvent accomplir d'autres tâches ou apporter de la redondance au système. Ainsi, un deuxième objectif de l'optimisation est de n'utiliser que des capteurs *utiles* pour capter l'information contenue dans la vue du capteur virtuel, où un capteur utile permet d'améliorer d'un certain niveau la densité de pixels de l'ensemble des capteurs. Cet objectif est noté par

la relation de préférence suivante

$$C\backslash\mathbf{c}_i \prec C \quad \text{si} \int_{\mathcal{I}} f_{\text{q}}(\mathbf{c}_{\text{v}}, C\backslash\mathbf{c}_i, \mathbf{i})\,\mathrm{d}\mathbf{i} < \int_{\mathcal{I}} f_{\text{q}}(\mathbf{c}_{\text{v}}, C, \mathbf{i})\,\mathrm{d}\mathbf{i} + \epsilon, \tag{1.2}$$

où l'on préfère exclure le capteur $\mathbf{c}_i$ s'il n'améliore pas la solution C d'un seuil ϵ lorsque f_{q} doit être minimisé.

1.2.3 Déploiement du réseau de capteurs

Le déploiement du réseau de capteurs pose aussi un problème de taille. Comme mentionné précédemment, nous voulons que le système proposé puisse être déployé dans un environnement dont la géométrie exacte est inconnue ou après que celle-ci ait radicalement changé depuis sa modélisation. Ceci a un impact sur deux composantes du système. Premièrement, l'évaluation de la couverture ne pourra de tout évidence pas bénéficier d'une forme analytique de la description de l'environnement. Deuxièmement, le système doit être capable de modéliser l'environnement de manière incrémentale et en cours d'utilisation.

1.3 Travaux connexes

Le problème abordé dans cette thèse est fortement lié connu au bien problème de la galerie d'art (O'Rourke, 1987), qui consiste à trouver l'emplacement optimal d'un nombre minimal de gardes pour observer (couvrir) une galerie d'art. Ce problème, faisant partie de la classe des problèmes NP-difficiles, a été étudié sous différents angles. En effet, Gage (1992) organise ce problème en trois approches distinctes. La première approche est de maximiser l'aire ou le volume total de détection. Une couverture de ce type est nommée **membrane**. L'objectif de la seconde approche est de minimiser la possibilité d'intrusion en couvrant uniquement les points critiques, ce qui revient à une couverture de type **barrière**. La troisième approche possible est de minimiser le temps de recherche d'une cible dans un environnement. On parle alors d'une couverture de type **balayage**. Dans notre cas, Puisqu'on souhaite couvrir entièrement une zone d'intérêt $\mathcal{I}$, le type de couverture à réaliser est une membrane. Ce type de couverture est étudié dans deux courants de recherche principaux : les réseaux robotiques et les stratégies de calcul de la meilleure prochaine vue. Cette section passe en revue les travaux récents de ces deux domaines de recherche.

1.3.1 Réseaux robotiques

Les réseaux robotiques, tel que décrits par Bullo et collab. (2009), représentent le domaine de recherche étudiant le déploiement optimal de robots capables d'observer à la fois leur

propre position, d'échanger des messages, de traiter de l'information et de contrôler leurs mouvements. Ils sont, en quelque sorte, une extension des réseaux de capteurs sans-fils au domaine de la robotique mobile.

Tout d'abord, Howard et collab. (2002) présentent une méthode de déploiement de capteurs par champs de potentiel. Dans cette méthode, les obstacles et les capteurs créent des forces servant à guider les capteurs vers une configuration permettant d'observer entièrement l'environnement. En d'autres mots, les capteurs se comportent comme des particules chargées positivement dans un environnement où les obstacles sont aussi chargés positivement. Cette méthode permet un déploiement de capteurs omnidirectionnels relativement uniforme dans l'environnement et mène donc à une couverture quasi-optimale sous ces conditions.

Ensuite, Cortés et collab. (2004) étudient le placement de capteurs omnidirectionnels dans un environnement bidimensionnel convexe par l'algorithme de Lloyd. Cet algorithme, similaire à l'algorithme de partitionnement en k-moyennes, mais pour des régions géométriques continues, crée une décomposition de l'espace de travail en cellules de Voronoï centroïdales, c'est-à-dire une décomposition où les points générant chaque cellule de Voronoï correspondent aussi au centre de masse de ces mêmes cellules. Il est montré que les capteurs placés au centre de ces cellules de Voronoï couvrent un environnement convexe de manière optimale.

Pimenta et collab. (2008) et Laventall et Cortés (2009) poursuivent dans la voie des travaux de Cortés et collab. (2004) respectivement pour des capteurs omnidirectionnels hétérogènes ayant des rayons de couverture différents et des capteurs directionnels. Plutôt que de faire appel à l'algorithme de Lloyd, ils proposent tous deux de monter le gradient d'une fonction objectif de visibilité pour atteindre une décomposition de l'espace de travail en cellules de Voronoï centroïdales.

Sur un autre ordre d'idée, Breitenmoser et collab. (2010) s'attaquent au problème du placement de capteurs omnidirectionnels homogènes dans un environnement non convexe. Ils montrent que la combinaison des algorithmes de Lloyd et de planification de trajectoire TangentBug (Kamon et collab., 1998) permet d'obtenir une couverture de type Voronoï même pour un environnement non convexe.

Puis, Bhattacharya et collab. (2010) considèrent le placement de capteurs omnidirectionnels dans un environnement initialement inconnu et non convexe. Leur méthode construit une grille d'occupation de l'environnement et divise l'espace en diagramme de Voronoï selon la distance de chaque cellule telle que donnée par l'algorithme de Dijkstra. L'algorithme de Lloyd est ensuite appliqué à cette division de Voronoï pour obtenir un déploiement optimal.

Pour leur part, Schwager et collab. (2009, 2011a) emploient le modèle mathématique de l'empreinte d'une caméra directionnelle pour observer un environnement bidimensionnel non convexe. Ils emploient un algorithme de descente du gradient pour trouver la position optimale de ces capteurs. Cette technique permet de déplacer les capteurs dans l'espace tridimensionnel et de couvrir des régions non connectées. Schwager et collab. (2011b) présentent aussi une fonction de combinaison des champs de vue des capteurs pour les zones couvertes par plus d'un capteur.

Plus tard, Marier et collab. (2012), Kantaros et collab. (2014) et Klodt et collab. (2014) proposent d'utiliser une mesure de visibilité plutôt qu'une mesure de distance lors de la décomposition de l'environnement en cellules de Voronoï. Cette mesure leur permet d'effectuer le déploiement de capteurs dans des environnements contenant des trous et des non convexités sans avoir à recourir à un algorithme de planification de trajectoire.

Enfin, Akbarzadeh et collab. (2011, 2013) dérogent significativement de la littérature en réseaux robotiques. Ils proposent un modèle probabiliste anisotropique directionnel de capteur dont le placement est fait dans un système d'information géographique prenant en compte le caractère tridimensionnel de l'environnement. Plusieurs algorithmes d'optimisation globale sont mis de l'avant afin d'optimiser la position des capteurs. Ils réussissent à couvrir des terrains à fort dénivelé et des zones comprenant des bâtiments en plaçant un grand nombre de capteurs à des élévations stratégiques.

1.3.2 Prochaine meilleure vue

Les travaux en inspection industrielle utilisent une tout autre approche à la sélection des points de vue pour observer un élément d'intérêt. En effet, plutôt que d'optimiser la position de tous les points de vue à la fois, les observations sont faites séquentiellement en choisissant à chaque itération le prochain point de vue qui maximise l'acquisition d'information, d'où leur nom de prochaine meilleure vue.

Tout d'abord, Connolly (1985) introduit deux algorithmes pour l'inspection d'objets inclus dans une sphère sur laquelle peut se déplacer un capteur. Les deux algorithmes sont incrémentaux et construisent en temps réel un modèle de l'environnement dans un octree d'occupation. Pour évaluer la qualité d'une vue, chaque algorithme simule à sa façon la vue du capteur dans l'octree et emploie la force brute pour trouver la meilleure prochaine vue.

Suivant les travaux de Connoly, Tarbox et Gottschlich (1995) formulent le problème comme une recherche dans un graphe où chaque noeud représente une opération d'acquisition et

possède un coût associé au nombre de surfaces observées par cette opération. Le nombre de surfaces observées pour chaque point de vue est déterminé en lançant des rayons [1] entre le point de vue et chaque voxel dans l'octree du modèle de référence de l'objet à inspecter. Trois algorithmes sont proposés. Les deux premiers choisissent de manière vorace le meilleur noeud représentant la prochaine vue en maximisant la fonction objectif sans ne jamais retourner en arrière. Le troisième emploie une stratégie totalement différente. L'algorithme débute avec n vues aléatoires et le vecteur de ces vues est optimisé par un recuit simulé. S'il apparaît impossible après un certain nombre d'itérations du recuit de couvrir l'ensemble de toutes les surfaces du modèle, n est incrémenté et le processus d'optimisation est redémarré. Les différents tests montrent que l'algorithme de recuit simulé produit les plans nécessitant le moins de points de vue.

Ensuite, Pito (1999) fait sensiblement appel aux mêmes principes de simulation du capteur dans un modèle de l'espace. Cependant, plutôt que de construire le modèle de l'objet observé dans un octree d'occupation, il construit une triangulation des surfaces vues et des surfaces inconnues. L'algorithme proposé choisit la prochaine meilleure vue en maximisant le nombre de surfaces inconnues observées par force brute.

Gonzáles-Baños et collab. (2000) et Nüchter et collab. (2003) étendent le domaine d'application des méthodes de prochaine meilleure vue en éliminant la contrainte du volume englobant. Ceci leur permet d'exploiter cette méthode pour explorer un environnement intérieur. Tous deux utilisent un modèle bidimensionnel de l'environnement où les obstacles sont représentés par des lignes. Dans leur méthode, la meilleure prochaine vue est sélectionnée en simulant un grand nombre de points de vue aléatoires et en choisissant celui voyant le plus grand espace inconnu. Nüchter et collab. (2003) proposent une extension tridimensionnelle à leur approche 2D en incorporant un algorithme de fusion des polygones.

Pour leur part, Low et Lastra (2006) proposent un algorithme hiérarchique exploitant la similarité entre points de vue rapprochés pour diminuer le nombre de points de vue à évaluer. L'algorithme débute avec une grille grossière de points de vue à évaluer. Puis, cette grille est subdivisée récursivement dans les zones où les points de vue répondent à une série de critères établis. Parmi ces critères, on trouve l'angle d'incidence, la densité d'échantillonnage et la visibilité. L'évaluation des points de vue est faite par lancer de rayons dans un octree modélisant l'environnement.

Finalement, Blaer et Allen (2009) et Dornhege et collab. (2013) présentent chacun un système automatisé d'acquisition de grandes scènes tridimensionnelles. Ils utilisent une représentation

1. Traduction libre de *Ray Tracing*

de l'environnement en voxels tous initialement inconnus et planifient les prochaines vues en simulant le capteur dans le modèle. Ils choisissent parmi un ensemble aléatoire la prise de vue observant le plus grand nombre de voxels inconnus.

1.4 Solution proposée

Comme énoncé dans la dernière section, les réseaux robotiques se concentrent principalement sur la coordination de capteurs pour l'observation, à de rares exceptions près, d'environnements bidimensionnels. Cette limitation est principalement due à l'utilisation de la descente du gradient pour atteindre une configuration Voronoï centroïdale. En effet, la dérivation de la fonction objectif nécessite la connaissance de la géométrie de la vue des capteurs. Cette géométrie dépend, en trois dimensions, de la géométrie de l'environnement. Malheureusement, cette dernière n'est pas connue analytiquement dans le cadre du problème formulé dans cette thèse. En conséquence, la fonction objectif ne pourra être dérivée et la descente du gradient sera impossible.

De leur côté, les approches de meilleure prochaine vue s'attaquent au problème de manière vorace en choisissant à chaque itération le meilleur point de vue sans se soucier des itérations subséquentes. Il s'en résulte une fragmentation de l'espace à observer et une augmentation du nombre de points de vue requis pour observer la scène tel que noté par Tarbox et Gottschlich (1995).

Individuellement, ces deux champs de recherche ne parviennent pas à dénouer le problème posé dans cette thèse. En effet, comme le mentionnent Yap et Yen (2014) dans leur revue des systèmes de placement de capteurs visuels, aucune solution n'est proposée à un problème combinant des capteurs directionnels, un environnement pouvant occasionner des occlusions et la qualité de perception du système. Pour résoudre le problème abordé dans cette thèse, nous proposons de conjuguer la capacité d'évaluation de la qualité du positionnement d'un capteur en trois dimensions des stratégies de calcul de la meilleure prochaine vue à la combinaison de la qualité d'un ensemble de capteurs des réseaux robotiques. On note que le problème proposé possède même deux autres caractéristiques importantes : la détermination automatique du nombre approprié de capteurs et la modélisation en ligne de l'environnement. Ceci en fait un problème de placement de capteurs particulièrement difficile à résoudre.

1.4.1 Qualité d'une configuration de capteurs

Tout d'abord, au chapitre 2, nous proposons une méthode novatrice afin de calculer la qualité de placement d'une caméra. De manière analogue au rendu par lancer de rayons et aux méthodes de calcul de la meilleure prochaine vue, nous simulons la projection des pixels d'une caméra dans l'environnement. La simulation suggérée reproduit les caractéristiques d'une caméra sténopé imageant sur un plan rectangulaire tessellé de pixels. Celle-ci surpasse le réalisme de tous les modèles de capteurs visuels présentés jusqu'à maintenant. Cette simulation donne l'aire de la surface couverte par les pixel dans l'environnement. En inversant cette aire, nous obtenons la densité de pixels par unité d'aire effective du capteur en tout point de l'environnement. Cette évaluation de la qualité du placement d'un capteur est à la base du système proposé.

Ensuite, toujours au chapitre 2, nous élaborons une méthode de combinaison des vues des capteurs se basant sur les recherches en réseaux robotiques. Cette combinaison permet d'obtenir une fonction objectif agnostique du nombre et du type des capteurs du réseau. En effet, la combinaison se base uniquement sur le résultat de la simulation des vues des capteurs pour donner une densité de pixels unifiée en tous points de l'environnement. De cette composition unifiée, nous introduisons une fonction objectif permettant de qualifier la qualité du placement de capteurs par rapport à la vue désirée. Cette quantification de la qualité de placement d'un réseau de capteurs constitue la première contribution majeure de la thèse de par sa possibilité d'application à des environnement tridimensionnels non convexes dont on ne connaît pas la géométrie exacte. Elle a été publiée à la conférence internationale ICRA 2015 (De Rainville et collab., 2015).

1.4.2 Optimisation de la configuration de capteurs

Au chapitre 3, nous suggérons l'utilisation d'une stratégie d'évolution pour l'optimisation des capteurs. Ces algorithmes d'optimisation à base de population sont de type boîte noire, ils offrent l'avantage de ne requérir que la valeur de la fonction objectif pour un vecteur d'entrée donné afin de trouver une solution optimale au problème. De plus, ces algorithmes sont particulièrement adaptés à l'optimisation de problèmes à nombres réels, car ils adaptent à l'espace de recherche une densité d'échantillonnage afin de maximiser la génération de solutions de qualité supérieure.

Ensuite, nous intégrons la stratégie d'évolution dans un méta-algorithme de coévolution coopérative. Ce dernier utilise une procédure diviser-pour-régner afin de trouver une solution à un problème modulaire. Nous employons donc cet algorithme pour trouver le nombre adé-

quat de capteurs pour acquérir l'information contenue dans la zone d'intérêt avec la densité d'échantillonnage désirée. Cette combinaison d'algorithmes répond naturellement au besoin d'optimisation du nombre et du placement des capteurs, chose généralement difficile à réaliser simultanément avec d'autres algorithmes d'optimisation. De plus, nous suggérons une variante à l'algorithme coopératif permettant d'améliorer les performances lorsque les problèmes deviennent plus difficiles, en l'occurrence lorsque le nombre de sous-problèmes augmente. L'amélioration à l'algorithme coopératif original constitue la seconde contribution majeure de la thèse. Elle a été publiée à la conférence internationale GECCO 2013 (De Rainville et collab., 2013).

1.4.3 Déploiement du système

La solution avancée au problème de placement de capteurs formulé précédemment prend la forme d'un système complet incluant tous les éléments nécessaires au bon fonctionnement d'un système robotique mobile. Au chapitre 4, nous proposons d'intégrer une série de mécanismes permettant la localisation, la navigation et la reconstruction de la vue virtuelle à l'algorithme d'optimisation pour former un système complet de déploiement de capteurs. Ce système est finalement testé aux chapitres 5 et 6 lors de trois types d'expérimentations. Tout d'abord, les premières expérimentations se déroulent en simulation avec un environnement bidimensionnel où les résultats sont comparés à l'état de l'art en réseaux robotiques. Ensuite, les secondes expériences, également en simulation, sont conduites dans des environnements tridimensionnels non convexes où la capacité du système à parvenir à une solution satisfaisante est mise à l'épreuve. Finalement, les troisièmes expériences généralisent les résultats de déploiement du système obtenus en simulation à un système réel employant de vrais robots. Cette système original complet, testé en simulations et en expérimentations, constitue la troisième contribution majeure de la thèse. Il a été publié à l'occasion du symposium internationale I-SAIRAS 2014 (De Rainville et collab., 2014b).

Première partie

Sous systèmes et intégration

Chapitre 2

Qualité du placement de capteurs

Le présent chapitre suggère une nouvelle fonction objectif pour le placement de capteurs dans un environnement tridimensionnel non convexe misant sur l'estimation de la densité de pixels par unité d'aire. L'originalité de notre proposition réside dans la combinaison de plusieurs caractéristiques des deux domaines principaux de la recherche en placement de capteurs, soit la prochaine meilleure vue et les réseaux de capteurs. Tout d'abord, à la section 2.1, nous présentons une méthode d'estimation de la qualité d'acquisition d'un capteur basé sur la simulation de points de vue dans l'environnement telle que traitée dans les approches prochaine meilleure vue. Contrairement à ce qui est employé généralement, cette simulation utilise un modèle de caméra sténopé plutôt qu'un simple calcul de visibilité. Cette méthode a l'avantage de fournir une bonne estimation de la résolution obtenue par le capteur plutôt qu'une réponse booléenne sur la visibilité en tous points de l'environnement. Ensuite, à la section 2.2, une représentation de l'environnement permettant la modélisation efficace d'un environnement inconnu et contenant toute l'information nécessaire au calcul de la qualité d'acquisition est mise de l'avant. Finalement, ces préalables établis, nous aborderons, à la section 2.3, la fonction objectif qui servira au placement de capteurs. Pour ce faire, nous traiterons d'abord, à la section 2.3.1, de la méthode pour délimiter la scène d'intérêt du capteur virtuel. Par la suite, à la section 2.3.2, nous proposerons une fonction d'erreur permettant de guider les capteurs vers l'observation de la scène d'intérêt avec une résolution au moins égale à celle demandée. Puis, nous introduirons, au point 2.3.3, une fonction de combinaison des vues, provenant de la littérature en réseaux robotiques, afin de combiner la qualité d'acquisition de multiples capteurs en une qualité d'acquisition pour le groupe de capteurs. Enfin, à la section 2.3.4, nous suggérons une méthode résolution numérique de l'intégrale de la fonction objectif.

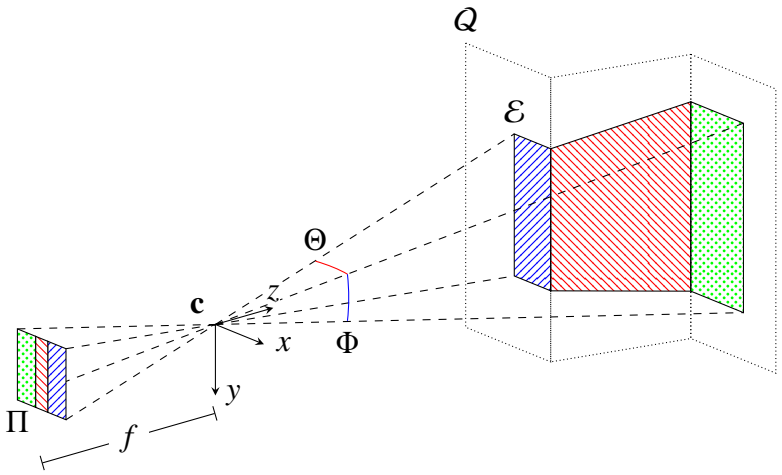


FIGURE 2.1 – Pyramide de visibilité et empreinte $\mathcal{E}$ d'un capteur caméra sténopé inverseur **c** de champ de vue $\Theta°$ par $\Phi°$ et de plan image Π à une distance focale f.

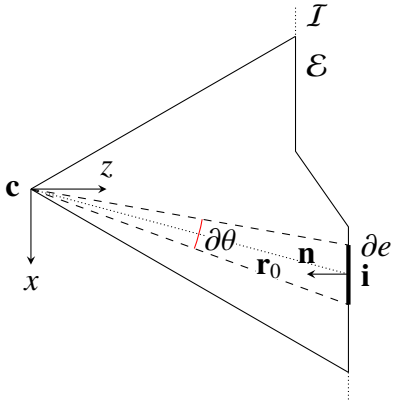
2.1 Qualité d'acquisition d'un capteur

La qualité d'acquisition d'un capteur représente sa capacité à observer son environnement avec précision. L'unité de base déterminant la netteté de ce qui est observé est le pixel. Plus le nombre de pixels observant un objet est grand, plus le rendu de ce dernier sera net. En effet, lorsqu'un pixel d'une caméra observe une surface contenant une grande quantité d'information (à savoir une image de faible résolution), l'entièreté de celle-ci (la couleur et la luminosité) est réduite en une seule valeur, soit la moyenne de tout ce qui se trouve dans le champ de vue du pixel ; l'information haute fréquence est donc perdue. À l'inverse, plus la quantité de pixels par unité d'aire sur les surfaces de l'environnement est grande (a savoir une image haute résolution), moins l'information sera moyennée et plus la qualité d'acquisition sera élevée. Cette idée gouverne la mesure de la qualité d'acquisition, en densité de pixels par unité d'aire, des capteurs proposée dans cette section.

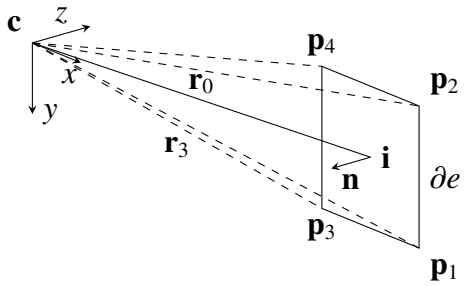
Pour calculer la densité de pixels par unité d'aire d'un capteur, nous utilisons un modèle de caméra sténopé et une surface rectangulaire tessellée de pixels pour représenter le plan image. En conséquence, la densité de pixels par unité d'aire d'un capteur est finie et inversement proportionnelle au carré de la distance du capteur. Les distorsions et les flous causés par les lentilles et les objets hors foyer ne sont donc pas considérés dans ce travail.

La géométrie du champ de vue du capteur sténopé inverseur **c** est présentée à la figure 2.1. La visibilité de ce capteur est représentée par une pyramide de visualisation [1] $\mathcal{V} \subset \mathbb{R}^3$ délimitée par son champ de vue en pointillé sur la figure. L'empreinte $\mathcal{E}$ du capteur **c** est définie par

1. Traduction libre de *viewing frustum*



(a) Représentation de l'empreinte $\mathcal{E}$ (en ligne pleine) et de la fraction d'empreinte ∂e (en ligne épaisse) produites respectivement par le champ de vue de la caméra **c** et l'angle $\partial\theta$ observant le point d'intérêt $\mathbf{i} \in \mathcal{I}$. Dans cette figure, $\partial\phi$ est caché par la projection orthogonale.



(b) Vue tridimensionnelle de la fraction d'empreinte ∂e et de ses composants.

FIGURE 2.2 – Géométrie de l'empreinte du capteur **c** pour la cible d'intérêt $\mathcal{I}$.

tous les points **p** visibles sur les surfaces de l'environnement $\mathcal{Q}$,

$$\mathcal{E} = \{\mathbf{p} \in \mathcal{V} \cap \mathcal{Q} \mid \nexists \mathbf{q} \in \mathcal{Q} \wedge \lambda \in [0,1] \wedge \mathbf{q} = \mathbf{c} + \lambda(\mathbf{p} - \mathbf{c})\}. \tag{2.1}$$

p fait partie de l'empreinte $\mathcal{E}$ s'il n'existe aucun point $\mathbf{q} \in \mathcal{Q}$ se trouvant sur le vecteur $\mathbf{p} - \mathbf{c}$ à une distance inférieure de **c** (à savoir le point **p** n'est pas occulté par aucune surface de l'environnement). Cette même pyramide de visualisation en deux dimensions est présentée à la figure 2.2(a). Elle est délimitée par les lignes pleines, soit par le champ de vue du capteur et par la surface de l'environnement. Afin d'estimer la densité de pixels en tous points d'intérêt **i** dans l'environnement, la base du tronc de visualisation est divisée en parties ∂e pour lesquelles la densité de pixels locale est calculée. Ces ∂e sont sous-tendues par les angles $\partial\theta$ et $\partial\phi$. On suppose que $\partial\theta$ et $\partial\phi$ sont suffisamment petits pour que l'empreinte ∂e soit plane au point **i**. L'aire des sous-empreintes ∂e est calculée comme suit.

Tout d'abord, l'équation du plan supportant l'empreinte ∂e est

$$(\mathbf{p} - \mathbf{i}) \cdot \mathbf{n} = 0 \tag{2.2}$$

où $\mathbf{p} \neq \mathbf{i}$ est un point quelconque sur ce plan et **n** est la normale de la surface au point **i**. Ce plan forme la base d'une sous-pyramide de visualisation associée au point **i**, comme montré à la figure 2.2(b). Les quatre arêtes $\mathbf{r}_1$ à $\mathbf{r}_4$ partant de l'apex de la pyramide sont obtenues en appliquant les rotations extrinsèques [2] R_y et R_x, respectivement d'angle $\pm\frac{\partial\phi}{2}$ et $\pm\frac{\partial\theta}{2}$, à $\mathbf{r}_0$ un

2. Application de la rotation par pré-multiplication. Les rotations sont effectuées dans le repère du sténopé.

vecteur entre le capteur $\mathbf{c}$ et le point d'intérêt $\mathbf{i}$, tel que

$$\mathbf{r}_1 = R_y\left(\tfrac{\partial\phi}{2}\right) R_x\left(\tfrac{\partial\theta}{2}\right) \mathbf{r}_0, \tag{2.3}$$

$$\mathbf{r}_2 = R_y\left(\tfrac{\partial\phi}{2}\right) R_x\left(-\tfrac{\partial\theta}{2}\right) \mathbf{r}_0, \tag{2.4}$$

$$\mathbf{r}_3 = R_y\left(-\tfrac{\partial\phi}{2}\right) R_x\left(\tfrac{\partial\theta}{2}\right) \mathbf{r}_0, \tag{2.5}$$

$$\mathbf{r}_4 = R_y\left(-\tfrac{\partial\phi}{2}\right) R_x\left(-\tfrac{\partial\theta}{2}\right) \mathbf{r}_0. \tag{2.6}$$

Par souci de lisibilité, seul $\mathbf{r}_0$ et $\mathbf{r}_3$ sont annotés à la figure 2.2(b). Ensuite, les arêtes $\mathbf{r}_1$ à $\mathbf{r}_4$ coupent le plan de l'équation 2.2 aux points

$$\mathbf{p}_k = \delta_k \cdot \mathbf{r}_k \qquad k = 1, \ldots, 4, \tag{2.7}$$

avec

$$\delta_k = \frac{\mathbf{i} \cdot \mathbf{n}}{\mathbf{r}_k \cdot \mathbf{n}}. \tag{2.8}$$

Puis, l'aire a du quadrilatère ∂e ayant pour sommet les points $\mathbf{p}_1$ à $\mathbf{p}_4$ est donnée par

$$a(\partial e) = \frac{\|\overline{\mathbf{p}_1\mathbf{p}_4}\|\|\overline{\mathbf{p}_2\mathbf{p}_3}\|}{2} \sin\varphi, \tag{2.9}$$

où $\|\cdot\|$ signifie la norme d'un vecteur et φ, l'angle entre les vecteurs $\overline{\mathbf{p}_1\mathbf{p}_4}$ et $\overline{\mathbf{p}_2\mathbf{p}_3}$. Puis, le nombre de pixels ρ dans la fraction d'empreinte ∂e est donné par les paramètres intrinsèques du capteur $\mathbf{c}$

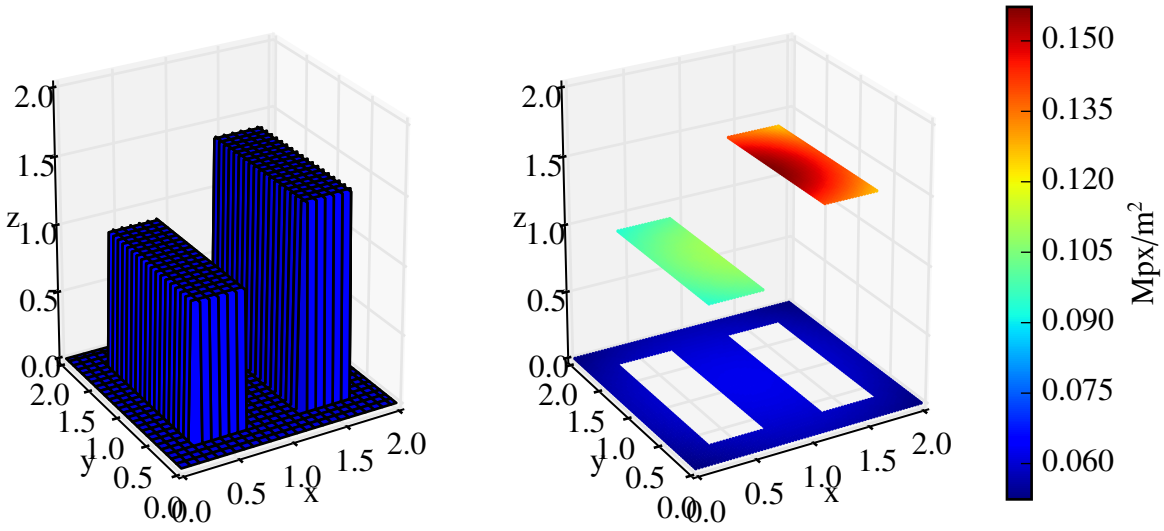
$$\rho = \frac{4f^2 \tan\left(\tfrac{\partial\theta}{2}\right) \tan\left(\tfrac{\partial\phi}{2}\right)}{uv}, \tag{2.10}$$

où f est la longueur focale de la caméra, u et v représentent la hauteur et la largeur d'un pixel, et $\partial\theta$ et $\partial\phi$ sont choisis arbitrairement. Finalement, la densité de pixels par unité d'aire $\hat{d}$ est donnée en tous points de l'environnement avec

$$\hat{d}(\mathbf{c}, \mathbf{i}) = \begin{cases} \frac{\rho}{a(\partial e)} & \text{si } \mathbf{i} \in \mathcal{I} \cap \mathcal{E}, \\ 0 & \text{sinon.} \end{cases} \tag{2.11}$$

L'équation 2.11 ne requiert que quatre variables, soit les paramètres intrinsèques des capteurs, la visibilité du point d'intérêt $\mathbf{i}$, la distance entre le capteur et ce point, et la normale à surface à ce point $\mathbf{n}$. Les paramètres intrinsèques sont constants pour chaque capteur, alors que la visibilité, la distance et la normale peuvent facilement être obtenues avec l'aide d'un modèle adéquat de l'environnement, tel que présenté dans la prochaine section.

La figure 2.3 présente la densité de pixels par unité d'aire telle que calculée par l'équation 2.11 pour environnement simple comprenant deux blocs de 0,5 m par 1,5 m et de 1 m et 1,5 m de hauteur. La caméra utilisée est située au centre de l'environnement à une altitude de



(a) Environnement

(b) Densité de pixels par unité d'aire

FIGURE 2.3 – Densité de pixels par unité d'aire calculée pour un environnement tridimensionnel comprenant deux blocs, l'un d'une hauteur de 1 m et l'autre 1,5 m.

4 m. Ses paramètres sont une focale de $f = 10\,\text{cm}$, une largeur de pixel de $u = v = 10\,\mu\text{m}$ et un champ de vue de $\Theta = 60°$ par $\Phi = 60°$. L'effet de dégradé sur les surfaces planes est dû à une combinaison de l'augmentation de la distance entre la caméra et la surface observée et au changement d'angle d'incidence de $\mathbf{r}_0$ par rapport la surface.

2.2 Représentation

Dans ce travail, la représentation de l'environnement est la structure dans laquelle est entreposée l'information du monde où évoluent les robots. Elle représente donc l'espace de travail tel que vu par les capteurs pour les algorithmes. Par conséquent, elle doit entreposer toute l'information nécessaire au calcul de la densité de pixels présenté à la section précédente. Les succès de l'algorithme de placement de capteurs dans un monde initialement inconnu et en mouvement reposent sur une modélisation de l'environnement et une collecte de l'information nécessaire efficace telles que présentées dans cette section.

La représentation d'un environnement a été traitée dès le début des années 1980. Dès lors, Chatila et Laumond (1985) identifient trois représentations de l'environnement complémentaires : les représentations géométrique, topologique et sémantique. Chacune propose un niveau d'abstraction différent permettant la réalisation de tâches différentes.

Géométrique Ce modèle est le plus près des données brutes des capteurs. On le construit le plus souvent par un simple nuage de points ou des polylignes décrivant les contours des objets de l'environnement. Cette représentation est fréquement utilisée pour la navigation par coordonnées, car elle permet facilement l'évitement d'obstacles et la planification de la trajectoire à emprunter.

Topologique Contrairement à la représentation géométrique, le modèle topologique met plutôt en valeur la relation entre les différents éléments du monde. En effet, la représentation topologique sera le plus souvent construite à partir du modèle géométrique en associant les différents éléments ensemble pour former des objets reliés par un graphe de connectivité. Par exemple, la représentation contiendra l'information « la tasse est sur la table » ou « le corridor est accessible par la porte ».

Sémantique Alors que le modèle topologique met en évidence les relations des objets entre eux, le modèle sémantique en décrit les attributs comme le nom, la fonction, la couleur, la dureté, etc. Ce modèle ajoute un sens aux éléments des modèles précédents.

Selon Brooks (1985), la sélection du type de modèle doit être faite en accord avec les caractéristiques du problème :

> A representation of the world is not something from which the world need be reconstructable. Rather a representation of the world is a statement of facts deducable from observations, and ideally includes enough facts that anything deducable from past observations is also deducable from the representation. A representation is not an analogous structure to the world; it is a collection of facts about the world.

Tel que décrit à la section 1.2, le placement de capteurs requiert la connaissance de la géométrie de l'environnement. En l'occurrence, on doit être capable de déterminer :

— la visibilité entre un capteur et la cible,
— la distance entre un capteur et la cible, et
— l'orientation des surfaces de l'environnement.

Le modèle géométrique, et, plus précisément, la grille d'occupation décrite dans cette section, contient à lui seul toute cette information.

2.2.1 Grille d'occupation

Introduite par Moravec (1988) et Elfes (1989), la grille d'occupation est une subdivision de l'espace en cellules, d'aire (ou de volume) égale, possédant chacune une probabilité indépendante des autres cellules d'être occupée. La figure 2.4 montre une grille d'occupation

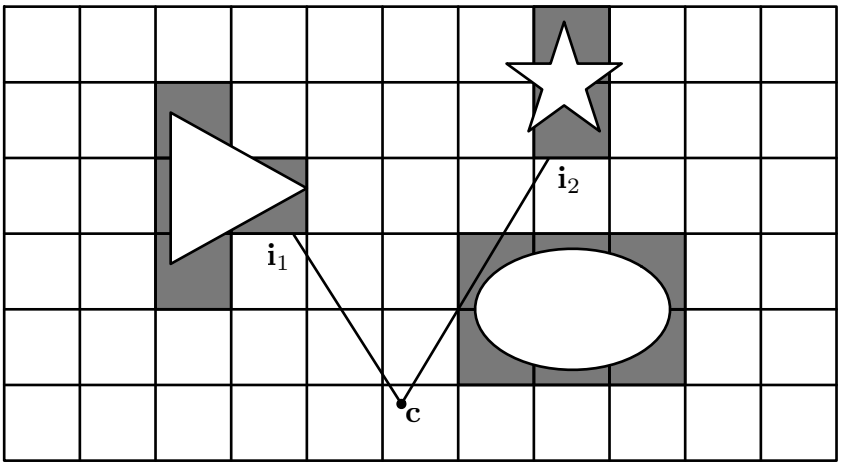


FIGURE 2.4 – Grille d'occupation pour un environnement contenant plusieurs objets. Les cellules blanches et grises sont respectivement libres et occupées. Le capteur **c** voit le point d'intérêt $\mathbf{i}_1$, mais pas $\mathbf{i}_2$ due à la discrétisation de l'environnement.

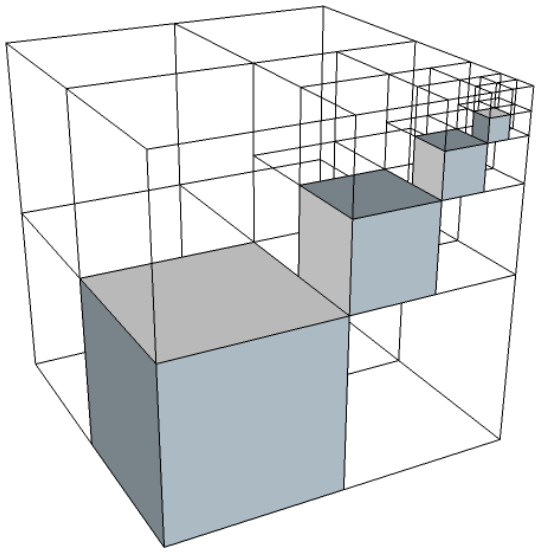


FIGURE 2.5 – Décomposition de l'espace tridimensionnel en octree à quatre niveaux. Chaque cube possède la même probabilité d'occupation dans l'entièreté de son volume, les cubes transparents représentent l'espace vide alors que les cubes gris représentent l'espace occupé.

contenant trois objets, les cases blanches représentant l'espace libre et les cases grises, l'espace occupé. Pour déterminer la visibilité entre deux points, il suffit de tracer un rayon entre ces deux points et de déterminer s'il traverse une case occupée. En effet, un capteur placé au point **c** peut voir le point $\mathbf{i}_1$ mais pas le point $\mathbf{i}_2$. Plus la grille est fine, moins il y aura d'erreurs de discrétisation, mais le nombre de données augmente selon $O(n^d)$, avec d le nombre de dimensions.

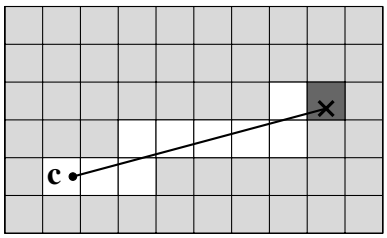


FIGURE 2.6 – Modélisation de l'environnement dans une grille d'occupation. Le capteur **c** effectue une mesure de distance correspondante à la marque en croix. Tandis que la probabilité d'occupation diminue pour les cases blanches traversées par le rayon entre le capteur et la cible, elle augmente pour la case gris foncé, endroit où un obstacle est détecté. La probabilité est non affectée pour les cases gris pâle qui sont non observées.

Pour pallier au problème de dimension des données, les grilles d'occupation tridimensionnelles sont généralement encodées dans un octree. Un octree est une décomposition hiérarchique de l'espace en cellules de probabilité d'occupation égale. Comme son nom l'indique, la décomposition se fait en arbre (« tree ») et chaque noeud possède huit enfants (« octo »). Le nombre de niveaux de l'arbre dépend de la taille de l'environnement et de la résolution spatiale. Comme le montre la figure 2.5, la racine de l'octree est un cube contenant tout l'environnement. Si une partie de l'espace circonscrit par ce cube a une probabilité d'occupation différente de ses voisins, le cube est subdivisé en huit parties ayant potentiellement chacune une valeur d'occupation différente. Ce processus est répété jusqu'à la résolution finale de l'octree. Dans un monde lisse, la capacité de l'octree à représenter de grands volumes par une seule valeur d'occupation mène à une réduction substantielle de la mémoire requise pour représenter de grands environnements en comparaison à une grille tridimensionnelle régulière (Hornung et collab., 2013).

Modélisation de l'environnement

La modélisation de l'espace se fait en incorporant les lectures de capteurs tridimensionnels dans l'octree. Pour une mesure de distance, telle que donnée par un capteur sonar ou laser, la probabilité d'occupation du voxel correspondant à la mesure est mise à jour ainsi que celle de tous les voxels se trouvant entre le capteur et ce voxel. Une illustration de cette opération est présentée à la figure 2.6. Comme le présentent Moravec (1988), Elfes (1989) et Thrun (2003), la probabilité d'occupation $p(x \mid z_{1:t})$ d'une cellule x connaissant toutes les mesures $z_{1:t}$ prises entre le temps 1 et le temps t est donnée par la règle de Bayes, soit

$$p(x \mid z_{1:t}) = \frac{p(z_t \mid x, z_{1:t-1})p(x \mid z_{1:t-1})}{p(z_t \mid z_{1:t-1})}. \tag{2.12}$$

L'hypothèse d'un monde statique est généralement faite afin de réduire la complexité de l'équation précédente. En effet, on pose l'indépendance entre toutes les mesures

$$p(z_t \mid x, z_{1:t-1}) = p(z_t \mid x). \tag{2.13}$$

Par conséquent, l'équation 2.12 se simplifie en

$$p(x \mid z_{1:t}) = \frac{p(z_t \mid x)p(x \mid z_{1:t-1})}{p(z_t \mid z_{1:t-1})}. \tag{2.14}$$

Appliquer à nouveau la règle de Bayes au terme $p(z_t \mid x)$ donne

$$p(x \mid z_{1:t}) = \frac{p(x \mid z_t)p(z_t)p(x \mid z_{1:t-1})}{p(x)p(z_t \mid z_{1:t-1})}, \tag{2.15}$$

la probabilité du voxel x d'être occupé. De manière analogue on obtient la probabilité du voxel x d'être libre

$$p(\neg x \mid z_{1:t}) = \frac{p(\neg x \mid z_t)p(z_t)p(\neg x \mid z_{1:t-1})}{p(\neg x)p(z_t \mid z_{1:t-1})}. \tag{2.16}$$

Diviser l'équation 2.15 par l'équation 2.16, en plus de transformer les probabilités en rapport de chance, permet d'éliminer plusieurs probabilités difficiles à calculer

$$\frac{p(x \mid z_{1:t})}{p(\neg x \mid z_{1:t})} = \frac{p(x \mid z_t)}{p(\neg x \mid z_t)} \frac{p(x \mid z_{1:t-1})}{p(\neg x \mid z_{1:t-1})} \frac{p(\neg x)}{p(x)}. \tag{2.17}$$

De plus, on sait que $p(x) = 1 - p(\neg x)$ et $p(x \mid \cdot) = p(\neg x \mid \cdot)$, et ce, peu importe la variable de conditionnement « $\cdot$ ». On peut donc réécrire l'équation 2.17 comme suit

$$\frac{p(x \mid z_{1:t})}{1 - p(x \mid z_{1:t})} = \frac{p(x \mid z_t)}{1 - p(x \mid z_t)} \frac{p(x \mid z_{1:t-1})}{1 - p(x \mid z_{1:t-1})} \frac{1 - p(x)}{p(x)}. \tag{2.18}$$

Cette forme ne dépend que de trois probabilités, soit

- $p(x \mid z_t)$ la probabilité que le voxel x soit occupé sachant la mesure z_t au temps t,
- $p(x \mid z_{1:t-1})$ la probabilité que le voxel x soit occupé sachant les mesures antérieures $z_{1:t-1}$, c'est-à-dire l'estimation du monde avant d'incorporer la nouvelle mesure, et
- $p(x)$ la probabilité indépendante du voxel x d'être occupé.

En général on pose $p(x) = 0{,}5$, c'est-à-dire que chaque voxel a la même probabilité d'être occupé que libre.

Afin de simplifier encore plus le calcul de la probabilité d'occupation, on transforme le rapport de chance en rapport de chance logarithmique, soit

$$l(\cdot) = \log \frac{p(\cdot)}{1 - p(\cdot)}. \tag{2.19}$$

En utilisant cette notation à l'équation 2.18, on trouve

$$l(x \mid z_{1:t}) = l(x \mid z_t) + l(x \mid z_{1:t-1}) - l(x). \tag{2.20}$$

Cela étant, on transforme trois multiplications en trois additions. Seulement la valeur de $l(x \mid z_{1:t})$ a besoin d'être conservée pour chaque voxel de l'octree. De plus, le logarithme n'a pas non plus besoin d'être calculé à chaque mise à jour puisque $l(x \mid z_t)$ est constant pour un capteur donné. Comme présenté à la figure 2.6, on met donc à jour tous les voxels entre le capteur et celui indiqué par la mesure de distance avec $l(x \mid z_t) < 0$ laquelle indique une probabilité d'occupation plus petite que 0.5 et le voxel indiqué par la mesure avec $l(x \mid z_t) > 0$. On peut facilement retrouver la probabilité d'occupation exacte avec

$$p(x \mid z_{1:t}) = 1 - \frac{1}{1 + e^{l(x|z_{1:t})}} \tag{2.21}$$

ou obtenir l'état estimé du voxel avec $l(x \mid z_{1:t})$ directement. Si $l(x \mid z_{1:t}) > 0$, le voxel est probablement occupé, alors que si $l(x \mid z_{1:t}) < 0$ il est libre. Autrement, il est impossible de le déterminer.

Les chances logarithmiques ont une convergence asymptotique vers $-\infty$ et ∞ respectivement lorsque le nombre de mesures libres et occupées pour un voxel tend vers l'infini. Cette propriété d'accepter une infinité de mise à jour est indésirable lorsque le monde n'est pas entièrement statique, car le nombre de mesures à intégrer afin de changer la valeur d'occupation d'un voxel est égal au nombre de mesures prises dans l'état précédent. Pour faire face à ce problème, Yguel et collab. (2008) présentent une procédure pour borner la valeur que peut prendre le rapport de chance logarithmique entre deux valeurs $l_{\min}$ et $l_{\max}$. Cette technique engendre deux avantages au niveau de l'octree :

1. les probabilités sont bornées, donc le modèle peut s'adapter à un environnement dynamique et
2. de grands volumes atteignent la même probabilité d'occupation, ce qui augmente le niveau de compression même si toutes les cases ne sont pas mises à jour aussi souvent.

La grille d'occupation seuillée encodée dans un octree permet donc de modéliser la géométrie l'environnement en incorporant de manière récursive les mesures prises par un ou plusieurs capteurs de distance de manière efficace tant en quantité de mémoire qu'en temps de calcul. Nous verrons maintenant comment l'information de visibilité et de distance peut aussi être récupérée efficacement de cette représentation

Simulation d'un capteur

La visibilité et la distance nécessaires au calcul de la fonction objectif de la section 2.3 sont disponibles directement dans la grille d'occupation. En effet, l'opération de lancer de rayon permet de simuler la vue d'un capteur dans une direction donnée en cherchant la première

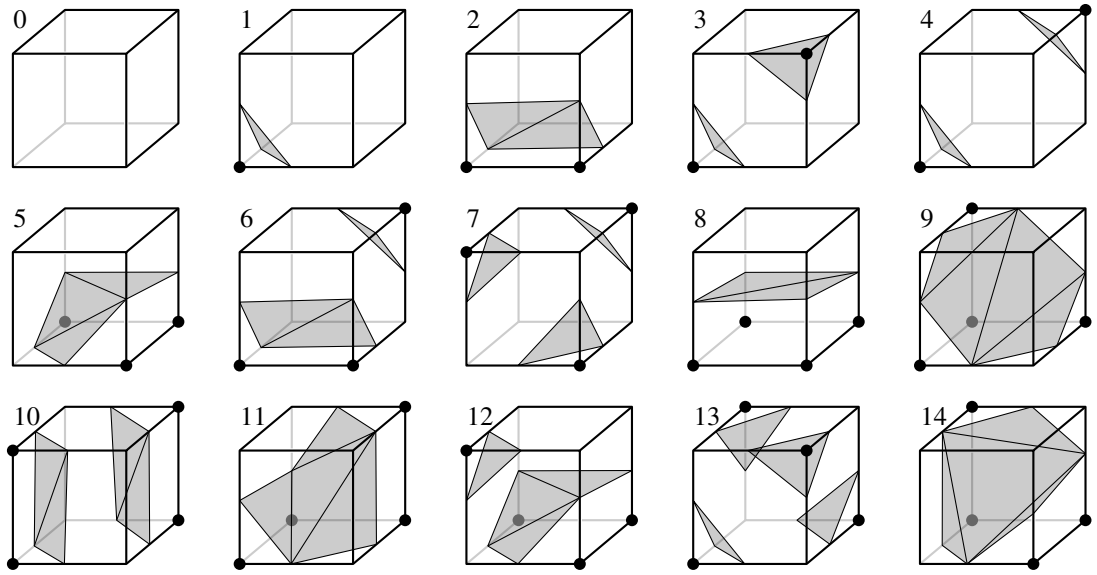


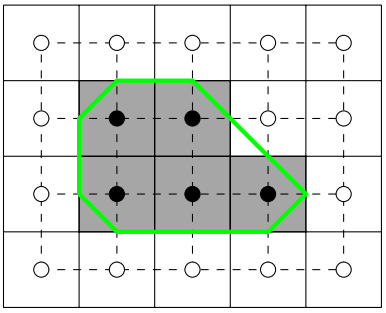
FIGURE 2.7 – Les 15 cas fondamentaux de l'algorithme « marching cubes » adaptés de Lorensen et Cline (1987).

surface occupée le long de ces rayons. Grâce à l'algorithme rapide de lancer de rayons dans une grille d'occupation de Amanatides et Woo (1987), il est aisé de déterminer de manière efficace la visibilité et la distance du point d'intérêt par rapport au capteur. Pour ce faire, un rayon est lancé à partir du capteur en direction du point d'intérêt. Si la première case occupée trouvée le long de ce rayon est différente de celle du point d'intérêt alors le capteur ne voit pas le point d'intérêt. Au cas contraire, la distance est déterminée en trouvant la norme du vecteur entre le capteur et le point d'intérêt.

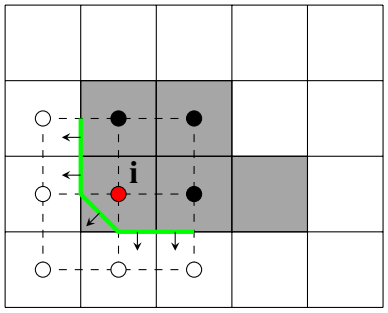
2.2.2 Marching cubes

Un désavantage de la grille d'occupation est la discrétisation des surfaces, c'est-à-dire toutes les surfaces sont représentées par une série de voxels alignés sur une grille. Donc, l'information d'orientation des surfaces de l'environnement est perdue. L'algorithme des « marching cubes », élaboré par Lorensen et Cline (1987), permet de reconstruire un maillage polygonal à partir d'un ensemble de voxels d'occupation. L'algorithme original parcourt les arêtes de la grille d'occupation et considère l'état des huit voxels voisins, qui forment un cube, pour déterminer la surface à l'intérieur du cube. La surface est choisie parmi un ensemble de configurations présentées à la figure 2.7. Une version bidimensionnelle de l'algorithme est présentée à la figure 2.8(a).

Cet algorithme est légèrement modifié afin de trouver toutes les surfaces autour du point d'intérêt $\mathbf{i}$. Pour ce faire, seulement les itérations de l'algorithme des marching cubes pour



(a) Algorithme original des marching cubes en deux dimensions. Chaque carré pointillé centré sur une arête de la grille d'occupation (en ligne pleine) représente une itération de l'algorithme, les cases grises sont occupées et les cases blanches libres. Le polygone produit est montré en vert, Il dépend à chaque itération des quatre arêtes du carré.



(b) Présentation des quatre itérations en deux dimensions nécessaires pour trouver les lignes au point d'intérêt **i** marqué en rouge. Toutes les normales représentées par des flèches sont retournées par l'algorithme.

FIGURE 2.8 – Calcul des surfaces à partir de l'algorithme des marching cubes. En trois dimensions, chaque segment de ligne représente une surface.

lesquelles le point d'intérêt se situe sur l'une ou l'autre des huit arêtes du cube sont effectuées, tel qu'illustré à la figure 2.8(b), soit huit itérations en trois dimensions. Une fois toutes les normales récupérées, la normale $\mathbf{n}$ minimisant l'angle d'incidence est choisie, ce qui a pour effet de maximiser la mesure de pixels par unité d'aire de l'équation 2.11.

2.3 Fonction objectif

La fonction objectif évalue la qualité d'une configuration de capteurs par rapport à la vue du capteur virtuel de manière à ce qu'on puisse comparer les différentes configurations entre elles. De manière générale, nous proposons d'utiliser une fonction objectif dont la forme est

$$f(\mathbf{c}_\mathrm{v}, C) = \int_I f_\mathrm{e}(f_\mathrm{q}(\mathbf{c}_\mathrm{v}, \mathbf{i}), f_\mathrm{c}(C, \mathbf{i}))\, \mathrm{d}\mathbf{i}, \tag{2.22}$$

où la qualité d'acquisition $f_\mathrm{q}(\cdot)$ doit être remplacée par la fonction calculant la densité de pixels par unité d'aire d'un capteur $\hat{d}$, telle que vue à la section 2.1. Nous gardons dans cette section la notation de qualité d'acquisition pour accentuer la possibilité de tirer parti de toute autre mesure qualifiant la vue d'un capteur, dans cette thèse elle est synonyme de densité de pixels.

La fonction objectif proposée intègre sur l'ensemble de la scène d'intérêt $\mathcal{I}$ l'erreur d'acquisition f_e faite par la combinaison f_c des vues des capteurs réels $\mathbf{c}_i \in C$ par rapport au capteur virtuel $\mathbf{c}_v$. La présente section décrit chacune des composantes de la fonction objectif ainsi que sa mise en oeuvre.

2.3.1 Scène d'intérêt

La scène d'intérêt $\mathcal{I}$ est délimitée, comme mentionné dans la définition du problème, par la vue d'un capteur virtuel $\mathbf{c}_v$ placé de manière souhaitée par l'utilisateur dans l'environnement. Le capteur virtuel peut prendre différentes formes, que nous étudierons ci-après, qui influencent chacune la manière dont est déterminée la scène d'intérêt.

Le capteur caméra délimite la scène d'intérêt par son empreinte $\mathcal{I} = \mathcal{E}_v$ telle que présentée à l'équation 2.1. La scène d'intérêt est donc déterminée par la visibilité du capteur virtuel, c'est-à-dire qu'elle contient tous les points en ligne de vue directe et dans le champ de vision du capteur virtuel. La qualité d'acquisition $f_q(\mathbf{c}_v, \mathbf{i})$ est la densité de pixels par unité d'aire $\hat{d}(\mathbf{c}_v, \mathbf{i})$ de l'équation 2.11.

Le capteur omniscient est un capteur permettant d'observer tout ce qui se trouve dans un volume $\mathcal{V}$, soit $\mathcal{I} = \mathcal{Q} \cap \mathcal{V}$. Ce capteur n'est pas limité par le critère de visibilité. En effet, ce capteur n'a pas de point de vue unique. Il observe la totalité des points à l'intérieur d'une région ciblée, même s'il est impossible de tous les voir simultanément à cause d'occlusions. La qualité d'acquisition $f_q(\mathbf{c}_v, \mathbf{i}) = \gamma$ de ce type de capteur virtuel est constante pour toutes les surfaces à l'intérieur du volume.

2.3.2 Fonction d'erreur

La fonction d'erreur $f_e : \mathbb{R}^2 \mapsto \mathbb{R}$ mesure la différence entre la qualité d'acquisition demandée par le capteur virtuel $\mathbf{c}_v$ et la qualité d'acquisition obtenue par le groupe de capteurs C en tous points de l'environnement. Comme décrit à la section 1.2, le groupe de capteurs doit observer l'environnement au moins avec la résolution demandée. Ainsi la fonction d'erreur ne doit ni pénaliser ni encourager les capteurs à avoir une qualité d'acquisition plus grande que celle qui est demandée. À cette fin, nous proposons d'utiliser une différence bornée entre les qualités d'acquisition s'exprimant par la fonction

$$f_e(f_q(\mathbf{c}_v, \mathbf{i}), f_q(C, \mathbf{i})) = \max\left(0, \frac{f_q(\mathbf{c}_v, \mathbf{i}) - f_q(C, \mathbf{i})}{f_q(\mathbf{c}_v, \mathbf{i})}\right), \tag{2.23}$$

où $f_q(C, \mathbf{i})$ est la qualité d'acquisition de l'ensemble de capteurs C au point $\mathbf{i}$ et la division par $f_q(\mathbf{c}_v, \mathbf{i})$ permet de borner l'erreur entre 0 et 1. En remplaçant la qualité d'acquisition $f_q(\cdot)$

par la densité de pixels $\hat{d}(\cdot)$ présentée précédemment, l'erreur produite par cette fonction est nommée erreur normalisée de densité de pixels (ENDP).

2.3.3 Combinaison des qualités d'acquisition

La fonction de combinaison $f_c : \mathbb{R}^n \mapsto \mathbb{R}$ permet de calculer la qualité d'acquisition lorsque plusieurs capteurs $\mathcal{D} \subseteq C$ observent un même point $\mathbf{q} \in \bigcap_{\mathbf{c}_i \in \mathcal{D}} \mathcal{E}_i$. Une telle fonction de combinaison permet d'obtenir une seule mesure de qualité peu importe le nombre de capteurs dans l'ensemble C, simplifiant donc significativement la comparaison de la qualité d'acquisition entre deux ensembles contenant potentiellement un nombre différent de capteurs.

Schwager et collab. (2011b) proposent la fonction de combinaison

$$f_c(q_1, \ldots, q_n) = \left(\sum_{i=1}^{n} q_i^{\alpha} \right)^{1/\alpha} \tag{2.24}$$

où $\alpha \in [-\infty, \infty]$ est un paramètre ajustable et les $q_i \geq 0$ sont les valeurs de qualité de mesure des différents capteurs. Cette fonction de combinaison reproduit la norme-p du vecteur $[q_1 \ \ldots \ q_n]$ lorsque $\alpha \geq 1$. Schwager et collab. (2011b) ont démontré qu'en posant $\alpha = -\infty$, la fonction de coût de leur contrôleur fondé sur le gradient reproduit le déploiement de capteurs suivant la décomposition en cellules de Voronoï (Cortés et collab., 2004). Cette dernière permet de minimiser le temps que prend un groupe de robots pour se rendre en tous points de l'environnement. En effet, si les q_i sont des mesures de distance euclidienne, la décomposition optimale de l'environnement est d'assigner chaque point au robot le plus près, soit une décomposition en cellules de Voronoï. La stratégie de déploiement optimale du groupe de robots est donc d'uniformiser la taille des cellules sur l'ensemble de l'environnement.

Suivant cette même logique, mais en utilisant notre mesure de qualité plutôt que la distance euclidienne, on associe chaque point de l'environnement au capteur l'observant avec la plus grande netteté. C'est pourquoi on utilise $\alpha = \infty$ afin de reproduire la norme-∞ ou une maximisation. Ainsi, en tous points de l'environnement, on retient uniquement la densité de pixels du capteur ayant la meilleure qualité d'acquisition parmi tous les capteurs. La fonction de combinaison utilisée est donc

$$\begin{aligned} f_c(C, \mathbf{i}) &= \left(\sum_{\mathbf{c}_i \in C} f_q(\mathbf{c}_i, \mathbf{i})^{\infty} \right)^{1/\infty} \\ &= \max_{\mathbf{c}_i \in C} f_q(\mathbf{c}_i, \mathbf{i}). \end{aligned} \tag{2.25}$$

Elle reproduit le déploiement de capteurs suivant la décomposition en cellules de Voronoï généralisée.

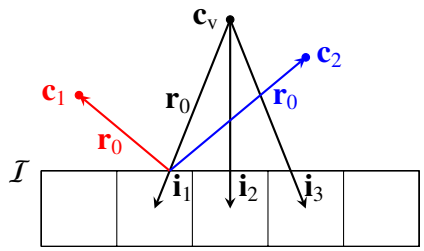


FIGURE 2.9 – Procédure de projection-rétroprojection pour le point d'intérêt $\mathbf{i}_1$.

2.3.4 Résolution numérique

La définition du problème stipule que l'environnement est tridimensionnel et que le système n'en possède qu'un modèle imparfait et incomplet. Ces trois contraintes rendent difficile, voire impossible, la définition de la géométrie de la cible $\mathcal{I}$ requise pour la résolution analytique de l'intégrale de l'équation 2.22. Cependant, la résolution numérique est toujours possible. En effet, nous pouvons estimer la valeur de l'intégrale en moyennant l'erreur de qualité d'acquisition en une série de points $\mathbf{i}_j \in \mathcal{I}$. On transforme donc l'équation 2.22 en

$$\hat{f}(\mathbf{c}_v, C) = \frac{1}{M} \sum_{j=1}^{M} f_e(f_q(\mathbf{c}_v, \mathbf{i}_j), f_c(C, \mathbf{i}_j)). \tag{2.26}$$

Pour résoudre cette équation, nous devons combiner tous les concepts vus jusqu'à maintenant. Tout d'abord, les points d'intérêt $\mathbf{i}_j \in \mathcal{I}$ sont déterminés, pour un capteur virtuel de type caméra, en échantillonnant les angles θ_j et ϕ_j uniformément dans le champ de vue du capteur virtuel $\mathbf{c}_v$. À partir de ces angles, un rayon $\mathbf{r}_0$ est lancé à partir de la position du capteur virtuel $\mathbf{c}_v$ dans le modèle de l'environnement. Le point d'intersection entre $\mathbf{r}_0$ et la première surface touchée constitue le point d'intérêt $\mathbf{i}_j$. La densité de pixels $\hat{d}(\mathbf{c}_v, \mathbf{i}_j)$ du capteur virtuel à ce point est déterminée selon l'équation 2.11 et les informations recueillies à partir du modèle. Ensuite, un nouveau rayon $\mathbf{r}_0$, partant du point d'intérêt, est lancé en direction de chaque capteur $\mathbf{c}_i \in C$ ayant le point d'intérêt $\mathbf{i}_j$ dans leur champ de vue. Si le rayon se rend au capteur $\mathbf{c}_i$, c'est-à-dire si le point $\mathbf{i}_j$ est en ligne de vue directe de $\mathbf{c}_i$, la densité $\hat{d}(\mathbf{c}_i, \mathbf{i}_j)$ est calculée, sinon elle est nulle. Finalement, la densité de pixels de tous les capteurs $\mathbf{c}_i \in C$ est combinée par la fonction $f_c(C, \mathbf{i}_j)$ décrite à l'équation 2.25 et l'erreur normalisée de densité de pixels est calculée par la fonction $f_e(\hat{d}(\mathbf{c}_v, \mathbf{i}_j), f_c(C, \mathbf{i}_j))$, présentée à l'équation 2.23. Cette procédure nommée projection-rétroprojection est présentée à la figure 2.9 pour le point d'intérêt $\mathbf{i}_1$.

2.4 Conclusion

Dans le présent chapitre, nous avons mis de l'avant une fonction objectif permettant de calculer la qualité d'une configuration de capteurs par rapport à une vue désirée par un utilisateur dans un environnement tridimensionnel non convexe. Le calcul se base sur un modèle réaliste de caméra sténopé inverseur avec un plan image discret. C'est en tirant avantage de cette discrétisation qu'on a pu déterminer en tous points de l'environnement la densité de pixels par unité d'aire induite par le capteur. Nous avons aussi présenté une représentation du monde permettant de modéliser efficacement, en terme de calcul et d'occupation mémoire, un monde initialement inconnu. Cette représentation contient toute l'information nécessaire au calcul de la qualité d'acquisition. Finalement, nous avons établi une fonction objectif qui définit la différence entre la qualité d'acquisition requise et obtenue, et combine la qualité d'acquisition d'un ensemble de capteurs sur une scène d'intérêt déterminée.

La fonction objectif décrite dans ce chapitre pose quelques problèmes d'optimisation. Tout d'abord, la géométrie tridimensionnelle non convexe, les discontinuités et les occlusions inhérentes, ainsi que le modèle imparfait et incomplet de l'environnement la rendent non dérivable. Ensuite, elle ne produit pas de contrôleur capable de diriger le robot d'un endroit à l'autre. Le chapitre 3 traite de la manière dont est optimisé le positionnement de l'ensemble de capteurs, alors que le chapitre 4 établit comment sont contrôlés chacun des robots.

Chapitre 3

Optimisation

Comme mentionné à la section 1.2, le problème de placement de capteurs abordé dans cette thèse consiste à trouver la configuration d'un nombre indéterminé de capteurs $\mathbf{c}_i \in C$ afin d'observer avec la résolution désirée une scène d'intérêt délimitée par le capteur virtuel $\mathbf{c}_\mathrm{v}$. Nous formulons le problème d'optimisation comme la minimisation de la fonction objectif $\hat{f}$ de la section 2.3 en contraignant chaque capteur $\mathbf{c}_i \in C$ à fournir une contribution minimale effective ϵ en terme de densité de pixels au réseau de capteurs, soit

$$\min_{C \in \mathcal{P}^n} \hat{f}(\mathbf{c}_\mathrm{v}, C) \tag{3.1}$$

$$\text{sujet à} \qquad \hat{f}(\mathbf{c}_\mathrm{v}, C \backslash \mathbf{c}_i) - \hat{f}(\mathbf{c}_\mathrm{v}, C) \geq \epsilon \quad \forall \mathbf{c}_i \in C, \tag{3.2}$$

où $C \backslash \mathbf{c}_i$ représente l'ensemble de capteurs C excluant $\mathbf{c}_i$.

L'utilisation d'un algorithme d'optimisation classique, par exemple, la programmation linéaire et la descente du gradient, n'est pas possible car

1. la fonction objectif $\hat{f}$ n'est pas convexe ,
2. cette dernière n'est pas non plus dérivable, car la géométrie exacte de l'environnement est inconnue et
3. le nombre de capteurs nécessaires doit être trouvé automatiquement.

Nous proposons donc de combiner deux algorithmes d'optimisation spécialisés pour réaliser le placement de capteurs décrit dans cette thèse. Ces deux algorithmes seront présentés et intégrés dans ce chapitre. Tout d'abord, à la section 3.1, nous introduisons les stratégies d'évolution (Beyer et Schwefel, 2002; Hansen et Ostermeier, 2001) permettant d'optimiser une fonction dont on ne peut calculer la matrice jacobienne. Les stratégies d'évolution constituent un type d'algorithme évolutionnaire (Mitchell, 1998) très efficace pour l'optimisation de problèmes à nombres réels (Auger et Hansen, 2009). Elles sont donc adaptées à notre

problème de placement de capteurs. Ensuite, nous décrivons, à la section 3.2, la coévolution coopérative (Potter et De Jong, 2001) permettant de décomposer automatiquement un problème en un nombre adéquat de sous-problèmes pour lesquels il est plus facile de trouver une solution. Lorsque combinées, ces solutions permettent de résoudre le problème original. La décomposition automatique offerte par la coévolution répond naturellement au besoin de trouver le nombre approprié de capteurs pour accomplir la tâche d'observation de la scène d'intérêt. Cette propriété en fait un algorithme de choix pour le problème que nous tentons de résoudre. Par la suite, à la section 3.3, nous suggérons une nouvelle forme de coévolution coopérative qui rend possible l'optimisation efficiente de problèmes plus difficiles que ceux à la portée de l'algorithme original. Nous montrons que cette addition permet notamment la décomposition de problèmes ayant un plus grand nombre de sous-problèmes naturels. Cette propriété s'avère nécessaire lorsque le nombre de capteurs pour observer une scène est grand. Finalement, à la section 3.4, nous combinons les stratégies d'évolution et la coévolution coopérative efficiente pour former l'algorithme d'optimisation qui servira à résoudre le problème de placement de capteurs.

3.1 Stratégies d'évolution

Les stratégies d'évolution (Beyer et Schwefel, 2002; Hansen et Ostermeier, 2001) font partie de la grande famille des algorithmes évolutionnaires. Elles se révèlent particulièrement efficaces pour l'optimisation de problèmes à nombres réels. Les stratégies d'évolution, divisée en deux sous ensembles soit les stratégies « + » et les statégies « , », sont des algorithmes d'optimisation sans dérivée qui maintiennent en tout temps une population $\mathfrak{P}_g$ contenant μ solutions candidates. À chaque itération g, aussi appelée génération, une population $\mathfrak{P}'_g$ de λ nouveaux candidats est générée en modifiant, selon un pas de mutation défini, les solutions de la population $\mathfrak{P}_g$. Les μ meilleurs candidats, parmi ceux de $\mathfrak{P}_g$ et $\mathfrak{P}'_g$ dans le cas des stratégies « + » ou uniquement ceux de $\mathfrak{P}'_g$ dans le cas des stratégies « , », forment la population $\mathfrak{P}_{g+1}$. L'aspect le plus intéressant des stratégies d'évolution est leur habileté à adapter le pas de mutation pour maximiser la génération de candidats de qualité supérieure. En effet, les stratégies d'évolution emploient divers mécanismes pour apprendre la densité de probabilité optimale. Parmi les mécanismes les plus populaires, on compte

— en raison de sa simplicité, l'apprentissage de la déviation standard isotropique par la règle du $1/5$ (Rechenberg, 1973) et,
— pour sa performance, l'apprentissage d'une matrice de covariance complète permettant une modification anisotropique des candidats (Hansen et Ostermeier, 2001).

La performance de ce second mécanisme sur de nombreux problèmes réels et théoriques (Auger et Hansen, 2009; Friedrichs et Igel, 2005; Tettamanzi et collab., 2011; Akbarzadeh et collab., 2013) et son début de preuve de convergence (Auger et Hansen, 2013) en font un algorithme de choix pour le placement de capteurs. Elle sera donc davantage détaillée dans le reste de cette section.

La stratégie d'évolution à matrice de covariance adaptée (CMA-ES, de l'anglais Covariance Matrix Adaptation Evolution Strategy) (Hansen et Ostermeier, 2001) est l'une des stratégies d'évolution les plus performantes tant à l'égard des problèmes synthétiques que réels (Auger et Hansen, 2009). Cette performance est principalement due à deux principes généraux. Premièrement, CMA-ES utilise une matrice de covariance complète déterminant la modification, suivant une loi normale, à apporter à un candidat afin de maximiser la probabilité de l'améliorer. De manière comparable à l'estimation de la matrice hessienne des méthodes quasi-Newton, la matrice de covariance est construite à partir des statistiques provenant des modifications antérieures fructueuses. Deuxièmement, CMA-ES conserve des statistiques après chaque itération afin d'ajuster la taille du pas d'optimisation. Cet ajustement permet d'éviter la convergence prématurée tout en rendant possible une convergence rapide sur un optimum.

CMA-ES est présenté à l'algorithme 3.1. Tout d'abord, λ solutions potentielles sont générées à partir d'une distribution normale multivariée $\mathcal{N}\left(\mathbf{x}^{(g)}_{\text{parent}}, (\sigma^{(g)})^2\mathbf{C}^{(g)}\right)$. Puis, la taille du pas d'optimisation $\sigma^{(g)}$ est ajustée en fonction du niveau de succès des solutions. Ensuite, si la meilleure des λ solutions $\mathbf{x}^{(g)}_{1:\lambda}$, évaluées par la fonction objectif f, est préférable ou égale ($\preceq$) à la solution parente $\mathbf{x}^{(g)}_{\text{parent}}$, le centre de la distribution d'échantillonnage est déplacé à cette solution et la matrice de covariance $\mathbf{C}^{(g)}$ est mise à jour pour maximiser la génération de solutions de qualité supérieure. Pour davantage de détails sur la mise à jour de la matrice de covariance, le lecteur intéressé est référé aux travaux de Hansen et Ostermeier (2001).

Pour l'optimisation du placement de capteurs, telle que décrite à la définition du problème abordé de la section 1.2, une solution candidate $\mathbf{x}^{(g)}_k$ contient tous les paramètres d'un capteur potentiel à la génération g. En tout temps, la solution parent $\mathbf{x}_{\text{parent}}$ de l'algorithme CMA-ES représente la configuration actuelle d'un capteur $\mathbf{c}_i \in C$. N stratégies CMA-ES sont donc nécessaires afin d'optimiser la position de N capteurs. Les prochaines sections décrivent un méta-algorithme faisant coopérer des stratégies CMA-ES pour trouver la position d'un ensemble de capteurs pouvant varier en nombre. On note ici que CMA-ES pourrait être remplacé par n'importe quel autre algorithme d'optimisation numérique n'utilisant pas le calcul direct de la dérivé, tel que le recuit simulé (Kirkpatrick et collab., 1983) ou l'algorithme

Algorithme 3.1 : $(1 + \lambda)$-CMA-ES

$g = 0$, initialiser $\mathbf{x}_{\text{parent}}^{(g)}$, $\sigma^{(g)}$ et $\mathbf{C}^{(g)}$;
répéter
  pour $k = 1, \ldots, \lambda$ **faire**
    $\mathbf{x}_k^{(g+1)} \sim \mathcal{N}\left(\mathbf{x}_{\text{parent}}^{(g)}, (\sigma^{(g)})^2 \mathbf{C}^{(g)}\right)$;
  fin
  actualiser la taille du pas $\sigma^{(g+1)}$;
  si $f\left(\mathbf{x}_{1:\lambda}^{(g+1)}\right) \preceq f\left(\mathbf{x}_{\text{parent}}^{(g)}\right)$ **alors**
    $\mathbf{x}_{\text{parent}}^{(g+1)} \leftarrow \mathbf{x}_{1:\lambda}^{(g+1)}$;
    actualiser la matrice de covariance $\mathbf{C}^{(g+1)}$;
  fin
  $g \leftarrow g + 1$;
jusqu'au *critère d'arrêt*;

de Broyden-Fletcher-Goldfarb-Shanno (BFGS) (Broyden, 1970; Fletcher, 1970; Goldfarb, 1970; Shanno, 1970). Toutefois, CMA-ES est choisi car sa supériorité a été démontré à mainte reprises pour des problèmes fortement non convexes et possédant une myriade d'optimums locaux (Auger et collab., 2010).

3.2 Coévolution coopérative

Les algorithmes coévolutionnaires (Axelrod, 1987; Hillis, 1990) représentent une extension des algorithmes évolutionnaires standards. Ils font appel à plusieurs populations d'individus compétitionnant ou coopérant en vue de solutionner un problème. En effet, les populations, appelées *espèces*, participent à une évaluation conjointe, c'est-à-dire que la qualité d'un individu d'une population est évaluée conjointement avec celle des individus des autres populations. De cette manière, l'interaction entre les populations mène à une compétition ou à une collaboration permettant générer des solutions répondant de mieux en mieux aux critères du problème.

Proposée par Potter et De Jong (2001), la coévolution coopérative utilise la modularité d'un problème afin de le décomposer en sous-problèmes plus simples à résoudre. Dans ce paradigme, chaque espèce représente une partie de la solution globale ; ce n'est qu'en combinant un représentant de chaque espèce qu'on obtient une solution complète. De plus, le nombre d'espèces peut varier au cours du processus d'optimisation permettant de trouver automatiquement le nombre de sous-problèmes composant le problème initial. La coévolution coopérative a été appliquée à plusieurs reprises à des problèmes réels. Parmi ceux-ci,

Algorithme 3.2 : Coévolution coopérative

1 initialiser $\mathfrak{P} \leftarrow \{\mathfrak{S}_i \mid i = 1, \ldots, n\}$;
2 $\mathfrak{H} \leftarrow \{\text{choisir\_aléatoirement}(1, \mathfrak{S}_i) \mid i = 1, \ldots, n\}$;
3 **répéter**
4 **pour toutes** *espèces* $\mathfrak{S}_i \in \mathfrak{P}$ **faire**
5 $\mathfrak{S}_i \leftarrow \text{générer}(\mathfrak{S}_i)$;
6 évaluer($\mathfrak{S}_i, \mathfrak{H} \backslash \mathbf{h}_i$);
7 mettre_à_jour($\mathfrak{S}_i$);
8 **fin**
9 $\mathfrak{H} \leftarrow \{\text{choisir\_meilleur}(1, \mathfrak{S}_i), i = 1, \ldots, n\}$;
10 **si** *amélioration* $< T_{\text{i}}$ **alors**
11 $\mathcal{J} = \{j \mid \text{contribution}(\mathfrak{S}_j) < T_{\text{c}}, j = 1, \ldots, n\}$;
12 créer une nouvelle espèce $\mathfrak{S}'$;
13 $\mathfrak{P} \leftarrow \left(\mathfrak{P} \backslash \bigcup_{j \in \mathcal{J}} \mathfrak{S}_j\right) \cup \{\mathfrak{S}'\}$;
14 $\mathfrak{H} \leftarrow \left(\mathfrak{H} \backslash \bigcup_{j \in \mathcal{J}} \mathfrak{H}_j\right) \cup \{\text{choisir\_aléatoirement}(1, \mathfrak{S}')\}$;
15 **fin**
16 **jusqu'au** *critère d'arrêt*;

on compte la classification par ensembles (García-Pedrajas et collab., 2005), l'apprentissage multi-agents (Panait et Luke, 2005) et le placement de capteurs (De Rainville et collab., 2012b).

Plus particulièrement, nous utilisons la coévolution coopérative pour trouver le nombre de capteurs nécessaire à la couverture de la scène d'intérêt délimitée par la caméra virtuelle. La position de chaque capteur est optimisée par une espèce $\mathfrak{S}_i$ et le nombre de capteurs est déterminé par le nombre d'espèces présentes à tout moment durant l'optimisation. La coopération entre les espèces est réalisée au moment de l'évaluation des individus. En l'occurrence, l'évaluation d'un individu se fait toujours en collaboration avec un représentant $\mathbf{h}_k$ de chacune des n autres espèces. Cette collaboration forme la solution multicapteurs complète $C = \{\mathbf{x}_j\} \cup \{\mathbf{h}_k \mid k = 1, \ldots, n \text{ et } k \neq j\}$ qui est évaluée par l'équation 2.26 avec $\mathbf{c}_{\text{v}}$ le capteur virtuel courant.

L'algorithme 3.2 présente le pseudo-code de la coévolution coopérative telle que décrite par Potter et De Jong (2001). Tout d'abord, une population $\mathfrak{P}$ composée de n espèces $\mathfrak{S}_i$ est initialisée avec des solutions $\mathbf{s}_j$ choisies aléatoirement dans le domaine de recherche du problème. Ensuite, pour chaque espèce, on sélectionne aléatoirement un représentant $\mathbf{h}$ de chaque population pour le placer dans un ensemble de représentants $\mathfrak{H} = \{\mathbf{h}_1, \ldots, \mathbf{h}_n\}$. Subséquemment, chaque espèce est optimisée indépendamment des autres selon les étapes d'un algorithme évolutionnaire. Pour chaque espèce, aux lignes 5 à 7, on génère de nouvelles so-

lutions potentielles à partir des paramètres de l'espèce i. Puis, on évalue ces solutions en collaboration avec les représentants des autres espèces pour mettre à jour les paramètres de l'espèce en fonction du résultat des évaluations. Ces étapes peuvent être remplacées par n'importe quel algorithme d'optimisation [1]. Lorsqu'une génération s'est écoulée pour chaque espèce, on choisit la meilleure solution candidate de chaque espèce pour former l'ensemble des représentants $\mathfrak{H}$.

Ensuite, à la ligne 10, le progrès de l'algorithme est évalué. Pour ce faire, on vérifie si les représentants se sont améliorés plus qu'un seuil T_a par rapport à ceux d'il y a T_l générations. Si l'amélioration n'est pas suffisante, les espèces ne contribuant pas suffisamment à la solution finale, soit celles ayant une contribution inférieure à un seuil T_c, sont retirées de la population $\mathfrak{P}$ et leur représentant est supprimé de l'ensemble des représentants $\mathfrak{H}$. La contribution d'une espèce est généralement mesurée en calculant la différence de qualité de la solution globale avec et sans le représentant de cette espèce. Finalement, une nouvelle espèce générée aléatoirement est ajoutée à la population et un individu de cette espèce, aussi choisi aléatoirement, est inclus à l'ensemble des représentants. Le processus d'optimisation se poursuit jusqu'à ce qu'un critère d'arrêt soit atteint.

Potter et De Jong (2001) mettent de l'avant que les espèces peuvent évoluer de manière asynchrone à l'intérieur de l'algorithme (lignes 4 à 8). En effet, un nombre arbitraire de générations peut s'écouler pour chaque espèce avant que ces dernières ne fournissent un nouveau représentant. Il est donc possible d'allouer judicieusement le temps d'optimisation aux espèces offrant un plus fort potentiel d'amélioration. La prochaine section suggère une nouvelle méthode pour distribuer les ressources computationnelles afin de permettre une optimisation plus efficace.

3.3 Coévolution coopérative efficiente

L'égalité de la distribution des ressources computationnelles entre les espèces est un aspect introduit dans l'algorithme original de Potter et De Jong (2001). On imagine aisément que la décomposition automatique d'un problème puisse produire des sous-problèmes de difficulté inégale. Par conséquent, l'allocation des ressources doit se faire en fonction du rendement des espèces en terme d'amélioration de la solution. Cette section décrit les travaux présentés par De Rainville et collab. (2013) dans le but d'allouer les générations à chacune des espèces pour obtenir une coévolution efficiente et un rendement maximal.

1. Dans le cadre du placement de capteurs, nous utilisons l'algorithme CMA-ES décrit à la section précédente. La combinaison des deux algorithmes est présentée à la section 3.4.

Le problème de la distribution des ressources se formule en problème de décision où l'on doit choisir à chaque génération quelle population doit être optimisée afin de maximiser le rendement global de l'algorithme coévolutionnaire. Le problème du bandit manchot (Gittins, 1979; Lai et Robbins, 1985) étudie le choix que doit faire un joueur cherchant à maximiser ses gains lorsqu'il se retrouve face à une rangée de machines à sous ayant chacune une probabilité de gain différente et inconnue. Dans le contexte de la coévolution coopérative, nous étudierons la sélection de l'espèce la plus prometteuse. Cette section sera donc consacrée à la description et à l'incorporation de l'algorithme du bandit manchot à la coévolution coopérative.

3.3.1 Bandit manchot

Le problème du bandit manchot (MAB, de l'anglais Multi-Armed Bandit), introduit par Robbins (1952), est utilisé pour gérer le compromis exploration-exploitation d'un agent devant choisir une action a_i (une machine à sous) parmi un ensemble de k actions $\mathcal{A} = \{a_1, \ldots, a_k\}$ chacune associée à une récompense donnée selon une distribution de probabilité inconnue $\{R_1, \ldots, R_k\}$. L'agent fait face à deux buts opposés :

1. acquérir de l'information sur les distributions de probabilité de la récompense (exploration), et
2. profiter de sa connaissance afin de maximiser ses gains immédiats (exploitation).

La notion de regret est centrale aux algorithmes résolvant le problème du MAB. Elle permet de quantifier la performance des algorithmes de sélection d'action. Le regret consiste en la somme de la différence entre le gain obtenu par le joueur et le gain optimal obtenu en jouant l'action optimale. Par exemple, un algorithme parfait, soit un oracle, obtiendrait un regret nul, alors qu'un algorithme aléatoire obtiendrait un regret linéaire en fonction du nombre d'actions tirées.

Auer et collab. (2002) ont développé un algorithme nommé « borne supérieure de confiance » (UCB, de l'anglais Upper Confidence Bound) lequel permet d'atteindre le regret optimal de manière asymptotique dans les cas classiques de MAB avec récompenses indépendantes et stationnaires. Cet algorithme tente de trouver la meilleure action aussi prestement que possible tout en exécutant des actions d'exploration à une fréquence exponentielle décroissante. Malheureusement, dans le contexte de la sélection d'une espèce durant la coévolution coopérative, aucune des hypothèses d'indépendance et de stationnarité n'est respectée. En effet, la qualité des solutions d'une espèce dépend des individus des autres espèces et lorsque ceux-ci changent, la distribution de probabilité des récompenses est également modifiée.

Algorithme 3.3 : Assignation du crédit selon l'aire sous la courbe (AUC).

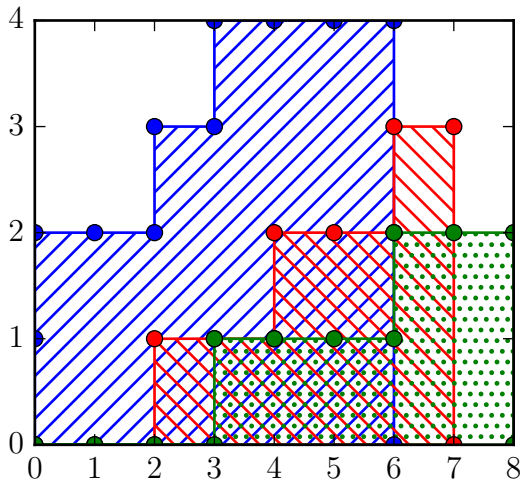
Entrées : indice d'une action i, fenêtre des récompenses $\mathbf{w} = [w_1, \ldots, w_W]$
Résultat : crédit g

trier lexicographiquement $\mathbf{w}$ selon les récompenses (meilleur d'abord), et l'âge en cas d'égalité (plus récent d'abord);
ajouter un élément nul à la fin de la fenêtre $\mathbf{w} \leftarrow [\mathbf{w}\ (-1, -1)]$;
$g \leftarrow y \leftarrow 0$;
pour chaque *rang* $r = 1, \ldots, W + 1$ **faire**
    $j \leftarrow \text{indice}(w_r)$;
    $\rho \leftarrow D^{(r-1)}(W - r - 1)$;
    si $i = j$ **alors** $y \leftarrow y + \rho$;
    sinon $g \leftarrow g + y\rho$;
fin

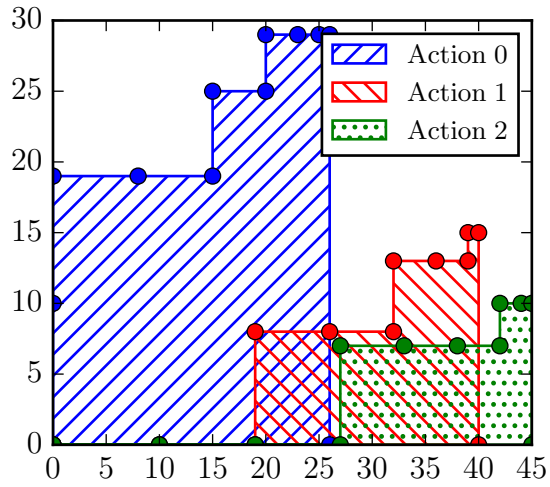
Da Costa et collab. (2008) présentent un MAB dynamique combinant les principes de UCB et les statistiques de Page-Hinkley afin de redémarrer le bandit lorsque les distributions de probabilité changent. Ce bandit est utilisé pour la sélection automatique d'un opérateur de variation dans un algorithme évolutionnaire. Il a par la suite été raffiné par Fialho et collab. (2010) afin d'augmenter sa robustesse et son invariance aux variations monotoniques ; pour ce faire, ils y ont ajouté une méthode d'attribution des récompenses établie sur le rang similaire à la méthode d'aire sous la courbe (AUC, de l'anglais Area Under the Curve) fréquemment utilisée en apprentissage machine.

L'algorithme AUC de Fialho et collab. (2010) sera utilisé dans la suite de la présente thèse pour l'attribution du crédit à chaque action. Une modification à l'algorithme original a été apportée afin de ne pas tenir compte des égalités qui sont nombreuses lorsque la récompense est binaire ($récompense \in \{0, 1\}$) et non continue. De plus, l'ordre d'apparition des récompenses permet de donner plus d'importance à une récompense plus récente. L'algorithme 3.3 présente l'attribution du crédit AUC à chaque action en fonction de la fenêtre des W dernières récompenses $\mathbf{w} = [w_1, \ldots, w_W]$ où w_i contient la récompense et l'indice i représente l'action jouée. La valeur $D \in [0, 1]$ est une valeur d'amortissement permettant aux récompenses classées en premier d'exercer une plus grande influence sur le crédit attribué. La valeur nulle ajoutée à la fin de la fenêtre $\mathbf{w}$ triée sert à donner une aire non nulle (mais faible) au dernier élément de $\mathbf{w}$. La figure 3.1 présente le crédit AUC pour une série d'actions ordonnées par valeur décroissante de gain, et ce pour trois valeurs d'amortissement. On note que plus l'amortissement est près de 0, moins les actions ayant un faible gain sont récompensées.

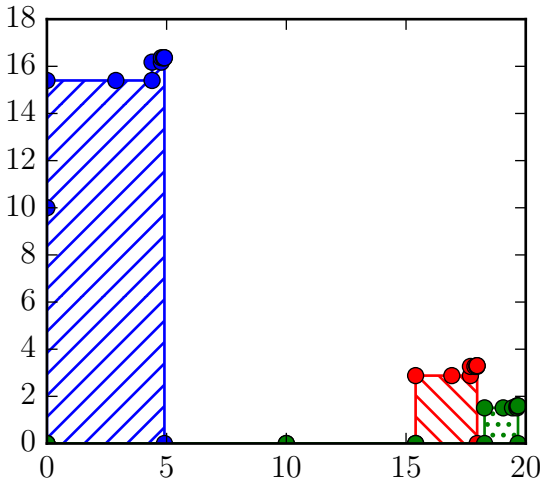
L'algorithme de sélection du MAB est similaire à celle de Fialho et collab. (2010) : un amalgame de UCB (Auer et collab., 2002) et d'une fenêtre glissante sur le crédit attribué par



(a) AUC sans amortissement $\rho = 1$, l'aire relative pour chacune des action est respectivement 54.3%, 25.7% et 20.0%.



(b) AUC avec amortissement $D = 1.0$, l'aire relative pour chacune des action est respectivement 69.2%, 22.6% et 14.5%.



(c) AUC avec amortissement $D = 0.6$, l'aire relative pour chacune des action est respectivement 88.8%, 8.8% et 2.5%.

FIGURE 3.1 – Aires sous la courbe pour une série d'actions $[0, 0, 1, 2, 0, 1, 0, 2, 1]$ ordonnées par valeur de gain.

l'algorithme 3.3. La fenêtre glissante permet à UCB d'oublier le crédit attribué à une action après W itérations. Ainsi, l'algorithme s'adapte mieux à une récompense dynamique. L'algorithme 3.4 présente la sélection d'une action. Initialement, le nombre de sélections $\mathbf{m} = [m_1, \ldots, m_k]$ et le crédit $\mathbf{g} = [g_1, \ldots, g_k]$ pour chacune des k actions sont à 0, alors que la fenêtre des récompenses $\mathbf{w}$ est vide. Les lignes 3 à 6 sont identiques à l'algorithme UCB standard. S'il existe une action qui n'ait jamais été jouée, on la choisit, sinon, on opte pour

Algorithme 3.4 : Sélection d'une action.

$\mathbf{m} \leftarrow [0, \ldots, 0], \mathbf{g} \leftarrow [0, \ldots, 0], \mathbf{w} \leftarrow \emptyset$;
répéter
  si $\exists m_i \in \mathbf{m} : m_i = 0$ **alors**
    $action \leftarrow$ choisir_aléatoirement$(1, \{i \mid m_i = 0, i = 1, \ldots, n\})$;
  sinon
    $action \leftarrow \arg\max_{i=1,\ldots,k} \left(g_i + C \sqrt{\frac{2 \log \sum_{j=1}^{n} m_j}{m_i}} \right)$;
  fin
  jouer l'*action* sélectionnée et recevoir la *récompense*;
  $\mathbf{w} \leftarrow [(récompense, action)\ \mathbf{w}]$;
  si $|\mathbf{w}| > W$ **alors** $\mathbf{w} \leftarrow [w_1, \ldots, w_W]$;
  ;
  $\mathbf{m} \leftarrow [\text{compter}(i, \mathbf{w}) \mid i = 1, \ldots, W]$;
  $\mathbf{g} \leftarrow \left[\frac{\text{AUC}(i,\mathbf{w})}{\sum_{j=1}^{W} \text{AUC}(j,\mathbf{w})} \,\middle|\, i = 1, \ldots, W \right]$;
jusqu'au *critère d'arrêt*;

l'action ayant la qualité maximale selon le crédit lui étant attribué et le nombre de fois qu'elle fut choisie. Le paramètre $C \in \mathbb{R}^+$, quant à lui, règle le poids de l'exploration par rapport à l'exploitation. Ensuite, l'action sélectionnée est jouée et la récompense obtenue est ajoutée à la fenêtre des récompenses $\mathbf{w}$. Si la taille de $\mathbf{w}$ est plus grande que la taille prescrite de la fenêtre W, on retire les éléments les plus vieux. On met ensuite à jour le nombre de sélections et le crédit de chaque action.

3.3.2 Sélection d'une espèce en coévolution coopérative

L'idée derrière l'algorithme proposé est d'accorder plus de temps d'optimisation aux espèces les plus prometteuses qu'aux autres espèces. Cette sélection des espèces est faite par l'algorithme MAB dynamique décrit à la section précédente. Cette stratégie possède deux avantages majeurs :

1. une accélération de la convergence des espèces vers une niche, et
2. une distribution adéquate des ressources aux nouvelles espèces.

En effet, une bonne gestion de l'allocation des ressources permet aux espèces ajoutées récemment de rattraper le retard qu'elles ont accumulé par rapport aux plus anciennes.

La coévolution coopérative efficiente résultante est présentée à l'algorithme 3.5. La procédure débute de manière similaire à la coévolution originale. La population d'espèces $\mathfrak{P}$ est initialisée avec des espèces générées aléatoirement et un représentant $\mathbf{h}$ de chaque espèce

Algorithme 3.5 : Coévolution coopérative efficiente avec un bandit manchot

```
1  initialiser 𝔓 ← {𝔖_i | i = 1, ..., n};
2  𝔊 ← {choisir_aléatoirement(1, 𝔖_i) | i = 1, ..., n};
3  répéter
4      choisir l'espèce i;
5      𝔖_i ← générer(𝔖_i);
6      évaluer(𝔖_i, 𝔊\h_i);
7      mettre_à_jour(𝔖_i);
8      h_i ← choisir_meilleur(1, 𝔖_i);
9      𝔊 ← {h_0, ..., h_i, ..., h_n};
10     assigner la récompense ϱ à l'espèce i ;
11     si amélioration < T_i alors
12         𝒥 = {j | contribution(𝔖_j) < T_c, j = 1, ..., n};
13         créer une nouvelle espèce 𝔖′;
14         𝔓 ← (𝔓 \ ⋃_{j∈𝒥} 𝔖_j) ∪ {𝔖′};
15         𝔊 ← (𝔊 \ ⋃_{j∈𝒥} 𝔊_j) ∪ {choisir_aléatoirement(1, 𝔖′)};
16         ajuster le nombre d'actions du bandit;
17     fin
18 jusqu'au critère d'arrêt;
```

est choisi au hasard et copié vers l'ensemble des représentants $\mathfrak{H}$. Le premier changement à l'algorithme de Potter et De Jong (2001) se retrouve à la ligne 4 où on choisit, en utilisant les lignes 3 à 6 de l'algorithme 3.4, l'espèce à faire évoluer (l'action à prendre). Une fois l'espèce choisie, les étapes normales d'évolution sont appliquées et le meilleur candidat remplace le représentant de son espèce i dans $\mathfrak{H}$. Ensuite, on attribue une récompense ϱ à l'espèce i selon la règle définie à la section suivante ; cette étape correspond aux lignes 9 à 13 de l'algorithme 3.4. Finalement, une itération de l'algorithme se conclut par le test de progrès suffisant tel que décrit précédemment à la section 3.2, et l'ajustement du nombre d'actions du bandit manchot (ligne 16).

Récompense

Récompenser correctement l'action sélectionnée est une phase critique devant être mise au point soigneusement. Le choix de la récompense doit encourager à la fois l'exploitation des meilleures actions, et permettre l'exploration des actions moins bien connues. À l'instar de ce qui est présenté par Fialho et collab. (2010), nous utilisons une récompense binaire de manière similaire à l'algorithme UCB original (Auer et collab., 2002). En effet, la récompense

ϱ (ligne 10, algorithme 3.5) est donnée par

$$\varrho = \begin{cases} 1 & \text{si } f(\mathfrak{H}^{(g)}) \prec f(\mathfrak{H}^{(g-1)}) \\ 0 & \text{sinon,} \end{cases} \tag{3.3}$$

où $a \prec b$ indique que l'on préfère a à b et $f(\mathfrak{H}^{(g)})$ (respectivement $f(\mathfrak{H}^{(g-1)})$) est le résultat de la fonction objectif pour l'ensemble des représentants de la génération présente (précédente).

La récompense binaire fondée sur l'amélioration est la récompense la plus adaptée au choix d'une espèce pour la coévolution coopérative pour deux principales raisons. Premièrement, considérer l'amélioration, soit la différence entre la valeur actuelle de la fonction objectif et cette valeur à l'itération précédente, plutôt qu'uniquement la valeur actuelle de la fonction objectif, tel que présenté par Fialho et collab. (2010), permet d'avoir une récompense entièrement découplée de cette dernière. En effet, dans le cas contraire, si un choix d'espèce entraîne une forte dégradation de la fonction objectif, pour un certain temps, les améliorations subséquentes ne seront pas récompensées même si elles constituent le choix optimal. Deuxièmement, l'utilisation d'un critère binaire est analogue à l'utilisation du signe de la dérivée. Puisque l'assignation du crédit de l'algorithme 3.3 est fait selon le rang des gains et que l'optimisation tend généralement à avoir un gain décroissant asymptotiquement vers 0 en fonction du temps, le signe de la dérivé assure de récompenser plus amplement un gain récent qu'un gain passé lorsqu'un second critère de classement basé sur l'ancienneté est utilisé.

Ajout et retrait d'actions

L'algorithme de coévolution coopérative exige également que l'on puisse ajouter et retirer des espèces, ce qui nécessite que le bandit puisse aussi ajouter et retirer des actions. Plutôt que de redémarrer entièrement le bandit, on modifie seulement son état interne pour ne pas perdre entièrement l'apprentissage fait avant l'ajout ou le retrait d'une action.

Pour ajouter une espèce, on ajoute une entrée à la fois au nombre de sélections et au crédit de chaque action, ainsi

$$\mathbf{m} \leftarrow [m_1, \ldots, m_k, 0]$$
$$\mathbf{g} \leftarrow [g_1, \ldots, g_k, 0].$$

Cette procédure force le bandit à sélectionner la nouvelle action un certain nombre de fois pour équilibrer le nombre de sélections de chaque action. La nouvelle espèce a donc la possibilité de se trouver une niche plus facilement. Le retrait d'une espèce de déroule de manière

pratiquement similaire. Les entrées correspondantes à l'action retirée i sont extraites des vecteurs $\mathbf{m}$, $\mathbf{g}$ et $\mathbf{w}$

$$\begin{aligned}\mathbf{m} &\leftarrow [m_j \mid j \neq i, j = 1, \ldots, k] \\ \mathbf{g} &\leftarrow [g_j \mid j \neq i, j = 1, \ldots, k] \\ \mathbf{w} &\leftarrow [w_j \mid arm \neq i, j = 1, \ldots, W].\end{aligned}$$

3.3.3 Résultats préliminaires

Nous comparons maintenant l'algorithme de coévolution coopérative efficiente (ACC-MAB) à l'algorithme original de Potter et De Jong (2001) (ACC) sur le problème de l'appariement de chaînes (Forrest et collab., 1993), tel qu'utilisé par Potter et De Jong (2001), Smith et collab. (1993) et Wu et Banzhaf (2010) pour tester la coévolution coopérative.

Le problème consiste à trouver un groupe de chaînes binaires d'appariement $\mathcal{A} = \{\mathbf{a}_1, \ldots, \mathbf{a}_n\}$ reproduisant le plus possible les bits d'un groupe de chaînes binaires cibles $C = \{\mathbf{c}_1, \ldots, \mathbf{c}_m\}$, où $n \ll m$ et n inconnu. Les chaînes du groupe d'appariement doivent donc généraliser les patrons du groupe cible. La force d'appariement $f_a(\mathbf{a}_i, \mathbf{c}_j)$ entre les chaînes $\mathbf{a}_i = [a_{i,1}, \ldots, a_{i,k}]$ et $\mathbf{c}_j = [c_{j,1}, \ldots, c_{j,k}]$ est donnée par le nombre d'éléments similaire à chaque position, soit

$$f_a(\mathbf{a}_i, \mathbf{c}_j) = \sum_{l=1}^{k} e(a_{i,l}, c_{j,l}) \quad \text{avec} \tag{3.4}$$

$$e(a_{i,l}, c_{j,l}) = \begin{cases} 1 & \text{si } a_{i,l} = c_{j,l} \\ 0 & \text{sinon.} \end{cases} \tag{3.5}$$

Ensuite, la force d'appariement $F_a(\mathcal{A}, C)$ entre deux groupes de chaînes $\mathcal{A}$ et C est définie par la moyenne de l'appariement maximal des chaînes de $\mathcal{A}$ sur chaque chaîne de C, soit

$$F_a(\mathcal{A}, C) = \frac{1}{m} \sum_{\mathbf{c}_i \in C} \max_{\mathbf{a}_j \in \mathcal{A}} f_a(\mathbf{a}_j, \mathbf{c}_i). \tag{3.6}$$

Par exemple, un groupe d'appariement $\mathcal{A} = \{$`1001`$\}$ apparie le groupe cible $C = \{$`0001`, `1000`, `1001`$\}$ avec un force de 3,33.

Les chaînes du groupe cible sont générées à partir de p patrons ayant une partie fixe et une partie variable, par exemple le groupe cible du paragraphe précédent aurait pu être généré à partir du patron `#00#`, où les `#` représentent la partie variable. Le but du problème est d'identifier la partie fixe des patrons ayant produit les chaînes cibles. Le mécanisme de décomposition du problème de l'algorithme coévolutionnaire est parfaitement adapté à ce type

de problème. En effet, ce dernier se divise naturellement en p sous-problèmes, pour lesquels on tente de trouver une chaîne maximisant leur force d'appariement sur l'ensemble cible. En combinant les sous-solutions, on parvient à un ensemble d'appariement contenant $n \approx p$ chaînes binaires identifiant les patrons.

Potter et De Jong (2001) définissent quatre scénarios servant à tester les différentes propriétés de leur algorithme. Les deux premiers scénarios jaugent respectivement la capacité de l'algorithme à identifier et à généraliser la couverture de multiples sous-problèmes lors de l'optimisation. Dans ces scénarios, le nombre d'espèces est fixe et la difficulté du problème se situe au niveau de l'appariement des sous-solutions. Nous nous intéressons plutôt aux deux autres scénarios qui impliquent une coadaptation des espèces et la découverte du nombre approprié d'espèces. Ces scénarios seront décrits plus en détail dans les prochaines sections.

Pour chaque scénario, la longueur k des chaînes binaires est fixée à 64 bits. Dans les algorithmes ACC et ACC-MAB, la génération de nouveaux individus d'une espèce à chaque itération est réalisée par un croisement en deux points, suivi d'une mutation inversant la valeur d'un bit. Ensuite, la qualité d'un individu est évaluée par la force d'appariement du groupe qu'il forme avec les représentants des autres espèces. Puis, la mise à jour d'une espèce est réalisée en sélectionnant les individus de qualité supérieure par tournoi. Finalement, la contribution d'une espèce est donnée par le nombre de fois où son représentant est utilisé dans la force d'appariement de l'ensemble des représentants. La table 3.1 présente les paramètres des deux algorithmes. Les paramètres communs sont choisis tels qu'indiqués par Potter et De Jong (2001), alors que les paramètres de ACC-MAB sont ceux proposés par Fialho et collab. (2010).

Nous définissons deux configurations de l'ensemble cible C présentant un degré de difficulté différent.

Configuration 1 De manière similaire à ce qui est utilisé par Potter et De Jong (2001), l'ensemble cible C de la première configuration contient $m = 30$ éléments initialisés à partir des $p = 3$ patrons suivants, où les 32 bits fixes sont marqués par des `0` et des `1`, et les 32 bits variables sont marqués par des `#`

```
1##1###1###11111##1##1111#1##1###1#1111##111111##1#11#1#11######
1##1###1###11111##1##1000#0###0###0#0000##000000##0#00#0#00######
0##0###0###00000##0##0000#0###0###0#0000##001111##1#11#1#11######
```

Configuration 2 La seconde configuration contient aussi $m = 30$ éléments dans l'ensemble cible C, mais ceux-ci sont générés à partir de $p = 5$ patrons. Les cinq patrons partagent les mêmes bits variables que la configuration précédente, soit les `#` dans la chaîne

```
*##*###*###*******##*##****#*##*###*#*****##*******##*#**#*#**######
```

Paramètres	ACC	ACC-MAB
Taille des individus	64	64
Taille des espèces	50	50
Nombre initial d'espèces	1	1
Probabilité de croisement	0.6	0.6
Probabilité de mutation	1.0	1.0
Probabilité d'inversion	1/64	1/64
Taille des tournois	3	3
Délai d'amélioration (T_l)	5	5
Seuil d'amélioration (T_i)	0.5	0.5
Seuil d'extinction (T_c)	5.0	5.0
Taille de la fenêtre (W)	–	50
Facteur d'amortissement (D)	–	1.0
Facteur d'exploration (C)	–	1.0

TABLE 3.1 – Paramètres utilisés pour l'appariement de chaînes.

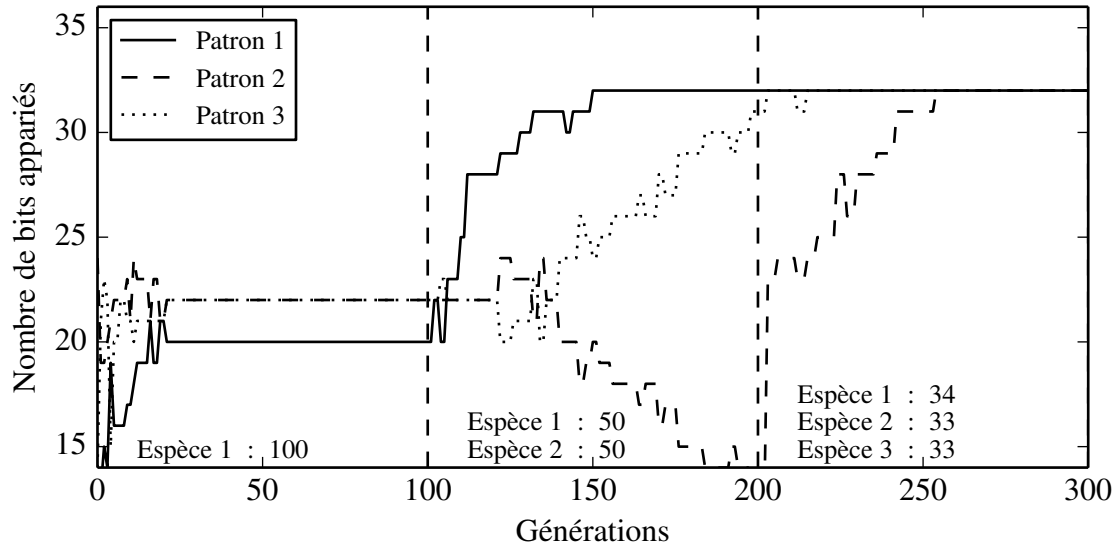
Les bits fixes de chacun des patrons, marqués ici par des *, sont choisis en tirant aléatoirement une combinaison avec répétition de 32 valeurs parmi $\{0, 1\}$. Le plus grand nombre de patrons complique la tâche de l'algorithme d'optimisation, car plus le nombre d'espèces est grand, moins l'effort par espèce sera grand pour une distribution équitable.

Coadaptation des espèces

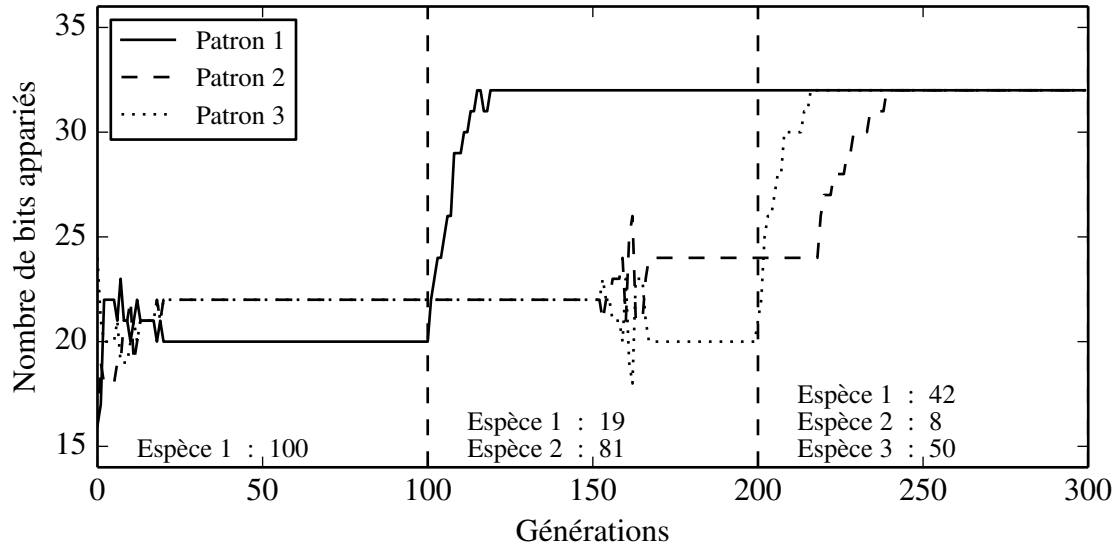
Ce test évalue la capacité d'adaptation de l'algorithme lorsque de nouvelles espèces sont introduites dans l'environnement. Pour ce test, l'algorithme d'optimisation n'est pas responsable d'ajouter automatiquement des espèces ; il débute plutôt avec une seule espèce et une nouvelle espèce est ajoutée à chaque 100 itérations jusqu'à ce qu'il y ait le même nombre d'espèces que de patrons. Ainsi, les résultats devraient montrer qu'au départ, alors qu'une seule espèce doit apparier tous les patrons, la solution tente de généraliser le plus possible et que plus des espèces sont ajoutées, plus les solutions se spécialisent sur un patron en particulier.

La sélection de l'espèce à évoluer par un MAB doit permettre d'atteindre un appariement idéal des patrons plus rapidement que pour l'algorithme original. En effet, lorsqu'une espèce cesse de faire progresser la qualité du groupe de représentant, le MAB cesse de la sélectionner. Ainsi, plus de ressources sont allouées à optimiser les autres espèces.

Les essais avec la première configuration ont été répétés 100 fois. ACC a réussi à trouver l'appariement optimal 99 fois alors que ACC-MAB y est parvenu à tout coup. Un essai typique



(a) ACC



(b) ACC-MAB

FIGURE 3.2 – Nombre de bits appariés de chaque patron par l'ensemble des représentants au cours de l'optimisation de la configuration 1. Le nombre de sélections de chaque espèce à chaque phase est présenté pour les deux algorithmes.

pour chaque algorithme est présenté à la figure 3.2, où on montre la force d'appariement sur chaque patron de l'ensemble des représentants. La figure 3.2(b) démontre l'allocation efficace des ressources à une espèce prometteuse. En effet, on voit qu'après les générations 100 et 200, là où sont introduites de nouvelles espèces, le nombre de bits appariés augmente plus rapidement que dans l'algorithme original à la figure 3.2(a). De plus, on remarque qu'une espèce

ayant convergé vers un patron ne reçoit qu'un nombre limité de ressources pour continuer son évolution. D'ailleurs, à la figure 3.2(b), on observe que lors de la dernière phase, l'espèce 2 ayant déjà convergé vers le patron 1 ne se voit sélectionnée qu'à 8 des 100 dernières générations, alors que les deux autres espèces, pour lesquelles un progrès est toujours possible, obtiennent 44 et 49 générations chacune. ACC-MAB réussit à éliminer l'effort inutile alloué à l'espèce 2 et l'investit de manière efficace sur les deux autres espèces.

La seconde configuration démontre de manière encore plus éloquente l'importance de l'allocation des ressources. Avec 5 patrons à identifier et donc 5 espèces, une allocation uniforme des ressources ne laisse que 25 générations à la dernière espèce introduite pour trouver son patron. Parmi les 100 répétitions réalisées avec la configuration 2, ACC n'a réussi à trouver que 22 fois un appariement idéal des patrons, alors que ACC-MAB a trouvé l'appariement parfait 71 fois.

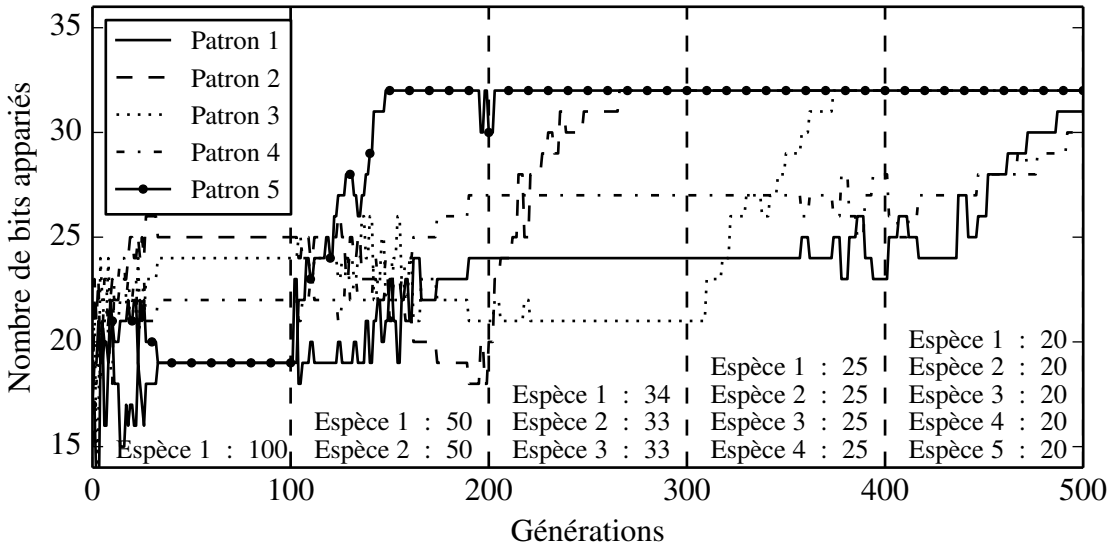
La figure 3.3 présente un essai typique pour chaque algorithme. On y constate plus aisément que la vitesse d'augmentation du nombre de bits appariés par patron ne diminue pas en fonction de l'augmentation du nombre d'espèces pour ACC-MAB alors que la pente décroît rapidement pour ACC. Conséquemment, ACC arrive rarement à trouver le cinquième patron dans le temps accordé. À la figure 3.3(b), on note que les générations sont distribuées de manière irrégulière parmi les espèces favorisant celles n'ayant pas atteint leur configuration optimale.

Ces deux tests montrent bien que l'utilisation d'un MAB pour la sélection de l'espèce à évoluer aide à l'adaptation des espèces et augmente leur habilité à trouver parfaitement tous les patrons. Qui plus est, l'algorithme utilisant le MAB est moins dépendant du nombre d'espèces présentes dans la population et conserve plus facilement son efficacité lorsque ce nombre augmente.

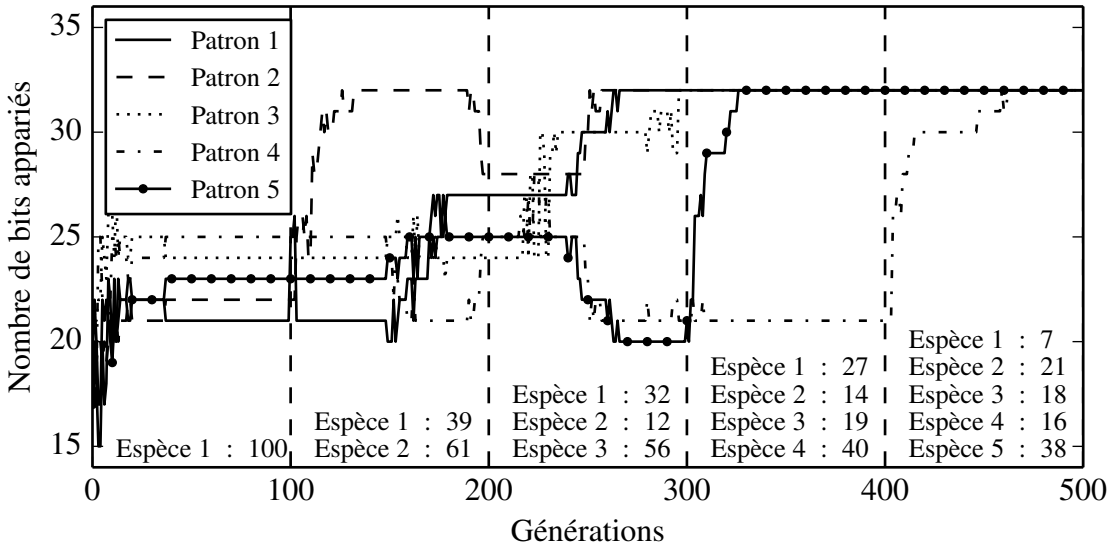
Découverte du nombre approprié d'espèces

Ce test utilise intégralement les algorithmes 3.2 et 3.5. Les patrons et le nombre de patrons doivent donc être trouvés automatiquement. La contribution d'une espèce est déterminée en comptant le nombre de chaînes pour lesquelles cette espèce obtient la couverture maximale parmi toutes les autres espèces. Cette expérience est la plus proche d'un problème réel : les algorithmes débutent avec une seule espèce et doivent ajuster ce nombre pour découvrir tous les patrons dans le temps accordé, soit 500 générations.

En plus de ce qui a été vu au test précédent, l'avantage du MAB se manifeste au niveau du nombre de générations accordées à une nouvelle espèce. Concrètement, toute nouvelle



(a) ACC



(b) ACC-MAB

FIGURE 3.3 – Nombre de bits appariés de chaque patron par l'ensemble des représentants au cours de l'optimisation de la configuration 2. Le nombre de sélections de chaque espèce à chaque phase est présenté pour les deux algorithmes.

espèce se voit accorder de nombreuses sélections successives de par la nature exploratoire du MAB. Cette sélection successive devrait permettre à la nouvelle espèce de se trouver une niche plus facilement.

La figure 3.4 présente la génération à laquelle tous les patrons ont été trouvés par chacun des deux algorithmes sur 100 essais de la première configuration. En moyenne, ACC a trouvé les

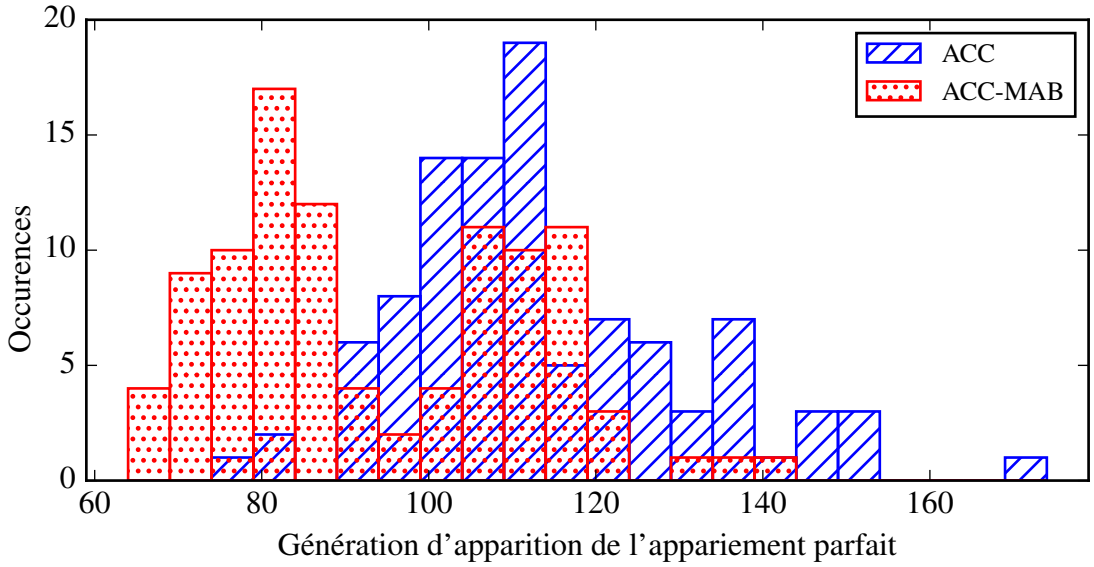


FIGURE 3.4 – Histogramme du nombre de générations requis pour atteindre l'appariement parfait pour la configuration 1.

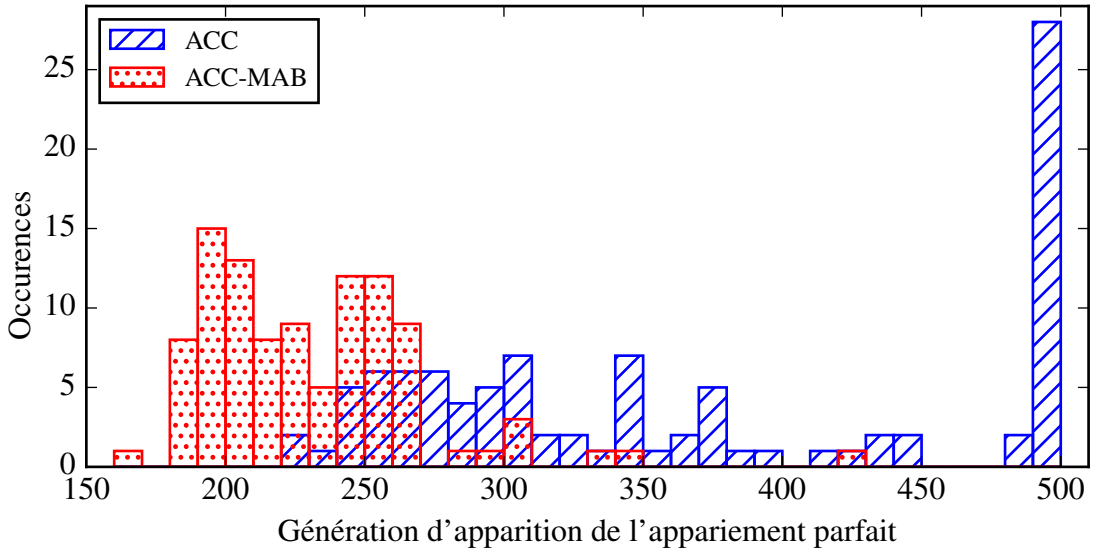


FIGURE 3.5 – Histogramme du nombre de générations requis pour atteindre l'appariement parfait pour la configuration 2.

trois patrons à la génération 112 alors que ACC-MAB est généralement plus rapide par 19 générations. Un test de Wilcoxon signé révèle que ACC-MAB est supérieur à ACC avec une valeur significative de 99.9%.

La génération de premier appariement parfait pour la seconde configuration est présentée à la figure 3.5. En moyenne, en excluant tous les essais qui n'ont pas convergé en moins de

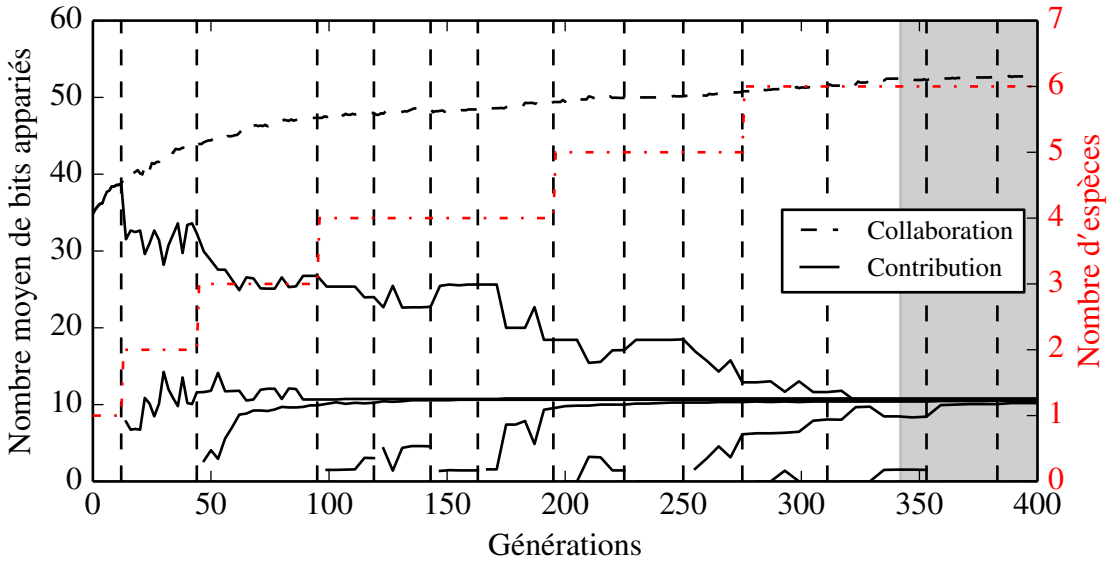
500 générations, ACC et ACC-MAB trouvent les patrons en 316 et 225 générations respectivement. Encore une fois, le test de Wilcoxon signé révèle que ACC-MAB est supérieur à ACC avec une valeur significative de 99.9%. En outre, on remarque que ACC-MAB trouve les cinq patrons dans 98% des essais alors ACC n'y parvient pas dans 27% des cas.

La figure 3.6 présente l'essai avec le nombre médian de générations pour atteindre l'appariement parfait. Chaque ligne pointillée verticale présente une génération où une stagnation a été détectée. On remarque à la figure 3.6(a) qu'après la centième génération les lignes verticales marquant la stagnation sont plus fréquentes pour ACC que pour ACC-MAB. En effet, il faut quatre essais à ACC avant de trouver une niche convenable pour la quatrième espèce. Ceci allonge le processus d'optimisation car le temps alloué à une espèce retirée est perdu. On constate à la figure 3.6(b) que ce phénomène ne se produit pas pour ACC-MAB. L'effort supplémentaire alloué aux nouvelles espèces par ACC-MAB permet de trouver une niche adéquate plus facilement et plus rapidement, augmentant la probabilité d'un essai fructueux. On note cette différence significative en comparant le nombre total d'essais effectués par les deux algorithmes ; ACC requiert 12 essais et un peu moins de 350 générations pour trouver les cinq patrons alors que ACC-MAB nécessite seulement six essais et un peu plus de 200 générations pour effectuer le même travail.

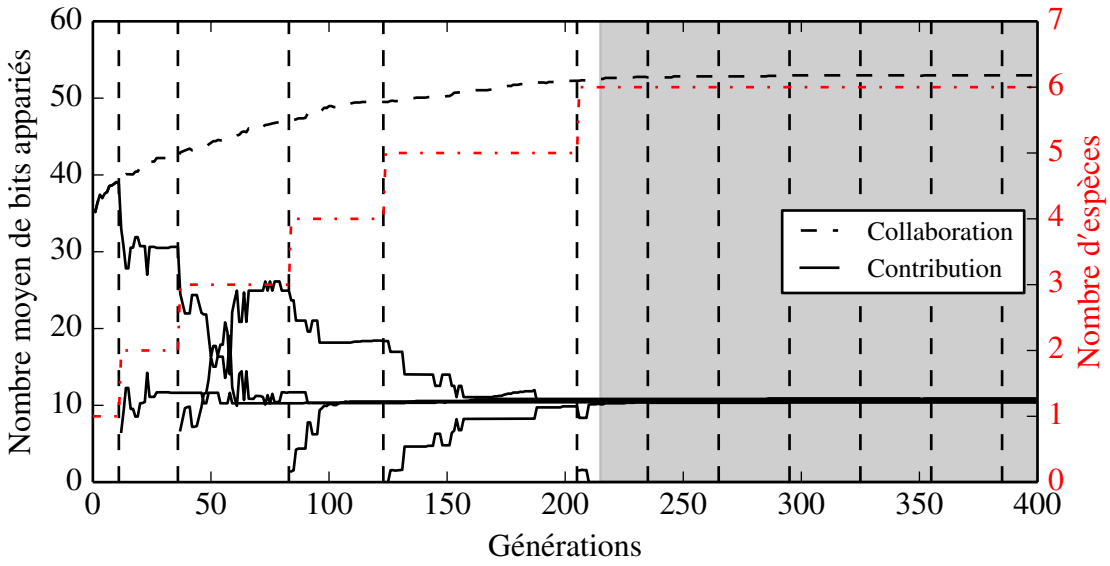
3.3.4 Conclusions préliminaires

Cette section a présenté un moyen efficace de distribuer les ressources entre les espèces lors de la coévolution coopérative afin d'augmenter les chances de trouver une solution optimale au problème d'optimisation. La méthode proposée exploite le plein potentiel des espèces donnant un rendement supérieur et alloue plus de temps d'optimisation aux espèces nouvellement introduites pour déterminer leur potentiel. Ensuite, notre approche permet à l'algorithme d'ajuster l'effort alloué à chacune des espèces lorsque celles-ci travaillent sur des problèmes de difficultés différentes. Les essais sur les différentes configurations du problème d'appariement de chaînes ont illustré que l'algorithme proposé dans ce chapitre permet l'obtention d'une solution optimale plus rapidement pour des problèmes où l'algorithme original n'arrive généralement pas à trouver une solution. Ces résultats sont plus amplement présentés aux chapitres 5 et 6.

L'algorithme présenté dans cette section peut être néanmoins amélioré de plusieurs façons. Tout d'abord, UCB étant un algorithme déterministe, il est impossible de tirer plus d'une action à la fois. Ainsi, l'algorithme perd sa capacité à être exécuté en parallèle pour chaque espèce. L'utilisation d'un bandit non-déterministe comme l'échantillonnage de Thompson



(a) ACC



(b) ACC-MAB

FIGURE 3.6 – Évolution du résultat médian pour la configuration 2 du problème d'appariement de chaînes. La ligne pointillée présente le nombre de bits appariés par l'ensemble des représentants de l'algorithme coopératif. Les lignes continues montrent la contribution de chaque espèce, soit la force moyenne d'appariement de son représentant lorsque celui-ci a la force d'appariement la plus élevée. Les lignes verticales indiquent la génération où le mécanisme d'ajout et de retrait d'espèces est mis en branle par la détection de stagnation. La ligne marquée de traits et de points présente le nombre d'espèces dans l'évolution, elle se rapporte à l'axe de droite. La zone ombragée présente la fin de l'évolution où les patrons sont tous trouvés.

Agrawal et Goyal (2013) permettrait toutefois de retirer cette limitation en tirant plus d'une action à la fois et en les exécutant en parallèle. Ensuite, un bandit combinatoire (Chen et collab., 2013) permettrait de ne fait aucune hypothèse sur l'indépendance entre les actions, permettant de choisir un groupe d'actions optimales plutôt qu'une seule. Les bandits sans repos (Whittle, 1988; Dai et collab., 2011) méritent aussi d'être étudiés puisqu'ils peuvent modéliser des récompenses non stationnaires, caractéristique présente dans la coévolution. Par la suite, la sélection de l'espèce s'applique naturellement à la coévolution compétitive. En effet, l'utilisation d'un bandit pourrait aider à équilibrer la coévolution entre les espèces afin d'éviter qu'une espèce n'en domine totalement une autre. Finalement, sur une autre note, l'emploi du bandit s'applique tout aussi naturellement aux algorithmes multipopulations pour lesquels il peut allouer les ressources aux populations les plus prometteuses.

3.4 Coévolution efficiente avec stratégie d'évolution à matrice de covariance adaptée

Cette section propose l'intégration de CMA-ES à l'algorithme ACC-MAB présenté à la section précédente. Cette intégration se fait de manière presque directe tel que montré dans l'algorithme 3.6.

Les premières étapes de la coévolution coopérative sont identiques à l'algorithme original. Ce n'est qu'aux lignes 6 à 10 qu'apparaissent les premiers changements. Ces lignes emploient la génération de nouveaux candidats à partir des paramètres de la stratégie CMA-ES i associée à l'espèce $\Im_i$. Contrairement à l'algorithme 3.1, le calcul de la qualité des solutions candidates est fait explicitement à la ligne 11. Ces qualités sont utilisées lors de la mise à jour des paramètres $\sigma_i^{(g+1)}$ et $\mathbf{C}_i^{(g+1)}$de la stratégie associée faite entre les lignes 12 et 22. Finalement, l'algorithme de coévolution se termine de manière similaire avec l'ajout et le retrait d'espèces en cas de stagnation.

Afin de favoriser la coadaptation des espèces, deux modifications simples ont été apportées à CMA-ES. La première, se situant entre les lignes 32 et 34 de l'algorithme 3.6, permet la diversification des espèces en doublant la taille du pas d'optimisation $\sigma^{(g)}$ chaque fois qu'une nouvelle espèce est introduite. Cette stratégie permet d'échapper aux minimums locaux qui auraient pu être atteints avec la configuration de capteurs précédente. La seconde modification intervient au niveau de la variance minimale que peut avoir la distribution multivariée $(\sigma^{(g)})^2\mathbf{C}^{(g)}$, soit entre les lignes 18 et 22. En effet, CMA-ES est connue pour sa convergence log-linéaire sur un optimum grâce à la réduction de la distribution d'échantillonnage multi-

Algorithme 3.6 : Intégration de CMA-ES à la coévolution coopérative efficiente

initialiser $\mathfrak{P} \leftarrow \{\mathfrak{S}_i \mid i = 1, \ldots, n\}$;
$\mathfrak{H} \leftarrow \{\text{choisir\_aléatoirement}(1, \mathfrak{S}_i) \mid i = 1, \ldots, n\}$;
$g \leftarrow 0$;
répéter
  choisir l'espèce i;
  $\mathfrak{S}_i \leftarrow \emptyset$;
  pour $k = 1, \ldots, \lambda$ **faire**
    $\mathbf{x}_{i,k}^{(g+1)} \sim \mathcal{N}\left(\mathbf{x}_{i,\text{parent}}^{(g)}, (\sigma_i^{(g)})^2 \mathbf{C}_i^{(g)}\right)$; // Génération d'individus
    $\mathfrak{S}_i \leftarrow \mathfrak{S}_i \cup \{\mathbf{x}_{i,k}^{(g+1)}\}$;
  fin
  évaluer($\mathfrak{S}_i, \mathfrak{H} \backslash \mathbf{h}_i$);
  $\sigma' \leftarrow \sigma_i^{(g)}, \mathbf{C}' \leftarrow \mathbf{C}_i^{(g)}$;
  actualiser la taille du pas σ'; // Mise à jour de la stratégie
  si $f\left(\mathbf{x}_{i,1:\lambda}^{(g+1)}\right) \preceq f\left(\mathbf{x}_{i,\text{parent}}^{(g)}\right)$ **alors**
    $\mathbf{x}_{i,\text{parent}}^{(g+1)} \leftarrow \mathbf{x}_{i,1:\lambda}^{(g+1)}$;
    actualiser la matrice de covariance $\mathbf{C}'$;
  fin
  si $(\sigma')^2 \mathbf{C}'(j,j) > \gamma \; \forall j \in [1, \ldots, s_i]$ **alors**
    $\sigma_i^{(g+1)} \leftarrow \sigma', \mathbf{C}_i^{(g+1)} \leftarrow \mathbf{C}'$;
  sinon
    $\sigma_i^{(g+1)} \leftarrow \sigma_i^{(g)}, \mathbf{C}_i^{(g+1)} \leftarrow \mathbf{C}_i^{(g)}$; // Prévention de la réduction
  fin
  $\mathbf{h}_i \leftarrow \text{choisir\_meilleur}(1, \mathfrak{S}_i)$;
  $\mathfrak{H} \leftarrow \{\mathbf{h}_0, \ldots, \mathbf{h}_i, \ldots, \mathbf{h}_n\}$;
  assigner la récompense ϱ à l'espèce i;
  si *amélioration* $< T_\text{i}$ **alors**
    $\mathcal{J} = \{j \mid \text{contribution}(\mathfrak{S}_j) < T_\text{c}, j = 1, \ldots, n\}$;
    créer une nouvelle espèce $\mathfrak{S}'$;
    $\mathfrak{P} \leftarrow \left(\mathfrak{P} \backslash \bigcup_{j \in \mathcal{J}} \mathfrak{S}_j\right) \cup \{\mathfrak{S}'\}$;
    $\mathfrak{H} \leftarrow \left(\mathfrak{H} \backslash \bigcup_{j \in \mathcal{J}} \mathfrak{H}_j\right) \cup \{\text{choisir\_aléatoirement}(1, \mathfrak{S}')\}$;
    ajuster le nombre d'actions du bandit;
    pour $j \in \mathcal{J}$ **faire**
      $\sigma_j^{(g+1)} \leftarrow 2\sigma_j^{(g+1)}$; // Diversification
    fin
  fin
  $g \leftarrow g + 1$;
jusqu'au *critère d'arrêt*;

variée. Toutefois, en situation de coadaptation, une réduction trop significative de l'espace de recherche n'est pas souhaitable car lorsque la diversité des solutions candidates d'une espèce devient trop faible cette dernière ne pourra pas s'adapter à un changement de configuration des autres espèces. Ainsi, pour l'optimisation coopérative, la mise à jour du pas d'optimisation et de la matrice de covariance ne s'effectuent que si le résultat donne un espace de recherche $(\sigma^{(g)})^2\mathbf{C}^{(g)}$ dont tous les axes ont une dimension supérieure à un certain seuil γ.

3.5 Conclusion

Dans ce chapitre, nous avons introduit deux algorithmes d'optimisation, soit la stratégie d'évolution et la coévolution coopérative. Ces algorithmes permettent respectivement d'optimiser une fonction objectif non dérivable et de décomposer automatiquement un problème en sous-problèmes. En combinant ces deux algorithmes, nous obtenons une procédure d'optimisation capable de trouver à la fois le nombre de sous-problèmes et leur solution optimale à un problème complexe. De plus, nous avons développé une modification à l'algorithme de coévolution coopérative améliorant ses performances sur des problèmes de plus grande difficulté. Nous avons proposé de distribuer les ressources computationnelles entre les espèce à l'aide d'un bandit manchot. Ainsi, les espèces plus prometteuses, c'est-à-dire celles permettant d'améliorer la qualité de la solution actuelle, se voient allouer plus de temps d'optimisation que les autres.

L'algorithme d'optimisation présenté dans ce chapitre sera utilisé, conjointement à la fonction objectif présentée au chapitre 2, pour l'optimisation du placement de capteurs. Les détails de cette intégration à l'intérieur d'un système complet de placement de capteurs sont présentés au chapitre suivant.

Chapitre 4

Intégration du système

Les deux derniers chapitres présentaient, d'une part, une méthode pour estimer la qualité d'une configuration de capteurs en fonction d'une scène d'intérêt à observer et, d'autre part, un algorithme d'optimisation pouvant diviser un problème en plusieurs morceaux pour en faciliter l'optimisation. Ce chapitre est consacré à la mise en oeuvre du système dans son ensemble, soit l'intégration de ces deux dernières parties dans un système complet permettant le placement de capteurs mobiles dans un environnement tridimensionnel initialement inconnu.

Le système entier est présenté à la figure 4.1. Chaque robot exécute localement la localisation et la planification de sa trajectoire alors que la représentation tridimensionnelle du monde, l'optimisation des positions de capteurs et la reconstruction de la vue virtuelle se font sur une unité centrale. Cette division permet de maintenir un seul modèle synchrone de l'environnement tout en ne surchargeant pas l'unité centrale ni le canal de communication avec les calculs de localisation et de planification. De plus, les robots sont ainsi indépendants de l'unité centrale ; si un robot ne reçoit aucune commande de l'unité centrale, il peut continuer sa tâche actuelle sans nuire au système.

Au cours de l'utilisation du système, chaque robot mesure continuellement la géométrie de l'environnement par ses capteurs tridimensionnels. Cette information est ensuite filtrée et envoyée à la représentation centralisée du monde. Chaque robot est responsable de transformer ses propres données dans le repère du monde à partir de sa position estimée par la localisation. Ainsi, l'unité centrale n'a pas le fardeau de faire concorder les données à l'intérieur du modèle. De plus, l'intégration constante des données des capteurs permet de construire un modèle complet de l'environnement au fur et à mesure que les robots l'explorent. Ensuite, l'information contenue dans la représentation est utilisée par l'algorithme d'optimi-

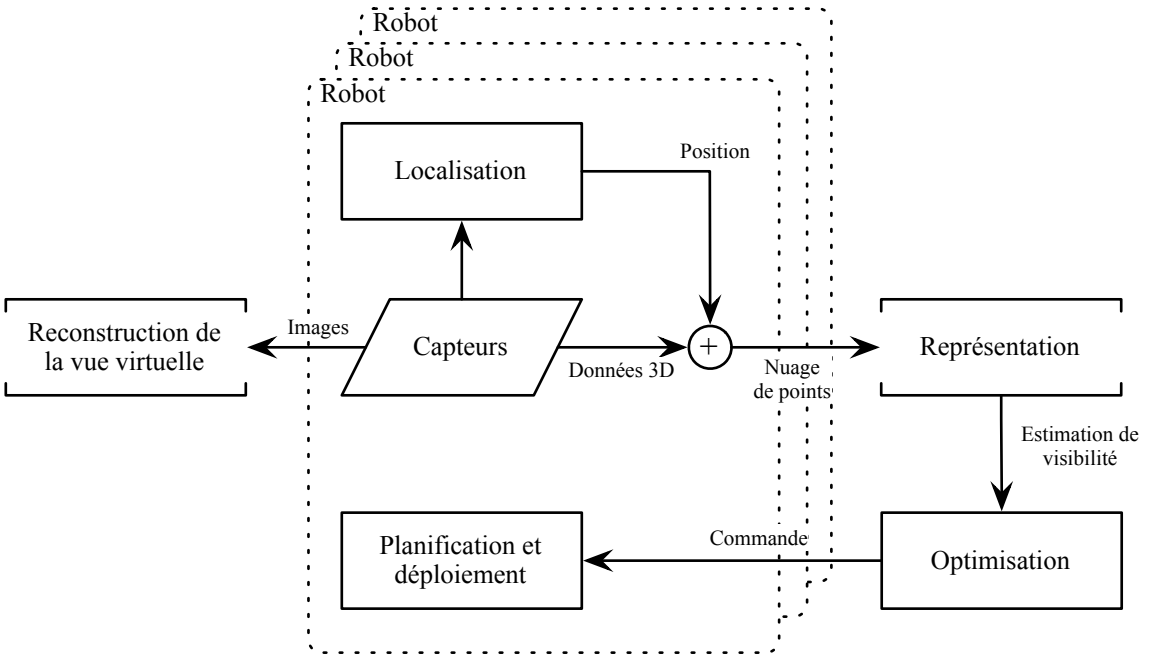


FIGURE 4.1 – Intégrations du système proposé. Les algorithmes à l'intérieur de l'entité « Robot » sont exécutés localement sur chaque robot, tandis que les autres algorithmes sont réalisés sur une unité centrale.

sation pour trouver une configuration de capteurs permettant d'observer la scène d'intérêt avec la densité de pixels désirée. La configuration de capteurs trouvée est envoyée aux robots pour déploiement. Une fois la commande exécutée, l'optimisation se poursuit jusqu'à ce qu'une couverture satisfaisante de la scène d'intérêt soit atteinte. Finalement, lorsque le déploiement est convenable, les images des capteurs sont utilisées pour reconstruire la vue virtuelle demandée par l'utilisateur. Chacune de ces étapes est maintenant détaillée dans ce qui suit.

4.1 Localisation

La première étape de notre système consiste à situer les robots dans leur environnement. Ceci leur permet de transformer les données de leurs capteurs dans le repère du monde et de planifier leurs déplacements pour atteindre la commande reçue. Il est assez rare qu'un système robotisé possède un dispositif lui fournissant sa position précise et ce type de dispositif est encore plus exceptionnel lorsque le système robotisé doit naviguer dans un environnement dangereux ou distant. Lorsqu'un tel appareil n'est pas disponible, la localisation doit être faite par les données des capteurs proprioceptifs et extéroceptifs. En général, ce type de localisation nécessite une carte de l'environnement qui permet d'estimer la position du robot

en recalant ce qu'il observe sur ce qu'il connaît du monde, ce mécanisme est présenté à la section 4.1.1. Malheureusement, une carte n'est pas toujours disponible. Le robot doit alors à la fois cartographier l'environnement et estimer sa position. Ce processus sera décrit à la section 4.1.2.

4.1.1 Localisation avec carte

Lorsqu'une carte de l'environnement est disponible, le problème de localisation consiste à trouver où se situe le robot dans cette carte à partir des mesures de ses capteurs. Il existe deux versions du problème de localisation : la localisation globale et locale. Dans la localisation globale, le robot ne connaît pas sa position initiale. Il doit donc estimer sa position à partir de zéro ; toutes les positions dans la carte sont donc préalablement équiprobables. Ce problème est aussi nommé le problème du robot kidnappé, où, à tout moment, le robot peut être déposé à un nouvel endroit dans le monde. Le second problème, la localisation locale, est sensiblement plus simple. Le robot a une idée générale de sa position et n'a qu'à corriger de manière incrémentale la mesure de ses déplacements. On réfère souvent à ce deuxième problème comme celui du suivi de robot. Proposée par Fox et collab. (1999) et Fox (2003), la localisation Monte-Carlo adaptative (AMCL, de l'anglais *Adaptive Monte Carlo Localization*) permet de résoudre à la fois ces deux versions du problème de localisation.

AMCL utilise un filtre à particules à échantillonnage-importance-rééchantillonnage [1] pour estimer la position du robot dans la carte m. Le filtre maintient, par un ensemble de particules $x_t^{(i)}$, une estimation de la densité de probabilité $p(x_t \mid z_{1:t}, u_{1:t-1})$, soit la probabilité du robot de se trouver à la position x au temps t en fonction des mesures des capteurs $z_{1:t} = z_1, \ldots, z_t$ et de l'odométrie $u_{1:t-1} = u_1, \ldots, u_{t-1}$. L'estimation de la position du robot se fait au gré de ses déplacements et de ses observations.

1. À chaque mouvement du robot, les particules sont **échantillonnées** à partir d'une distribution proposée $\pi(x_t \mid x_{t-1}, u_t)$, soit un modèle probabiliste de l'odométrie du robot. Essentiellement, chaque particule suit le mouvement du robot exprimé par l'odométrie u_t plus un certain bruit υ
$$x_{t+1}^{(i)} = x_t^{(i)} + u_t + \upsilon.$$
2. À chaque mesure des capteurs, l'**importance** $w^{(i)}$ de chaque particule $x_t^{(i)}$ est calculée selon la probabilité de mesurer z_t au point $x_t^{(i)}$
$$w^{(i)} = \eta p(z_t \mid x_t^{(i)}, m),$$
où η est une constante de normalisation de sorte que $\sum w^{(i)} = 1$.

1. Traduction de *sampling-importance-resampling*

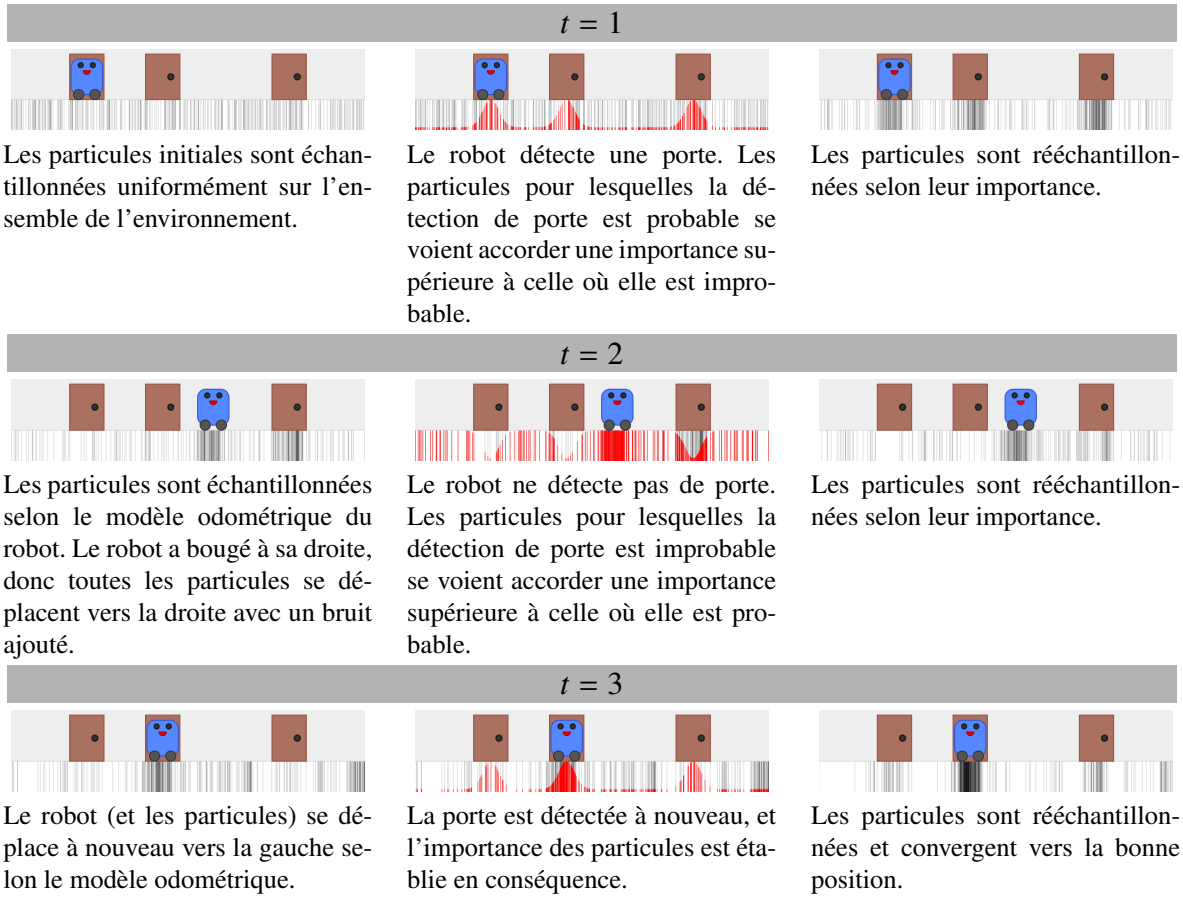


FIGURE 4.2 – Localisation Monte-Carlo adaptative [2].

3. Finalement, les particules participant à l'estimation de $p(x_t \mid z_{1:t}, u_{1:t-1})$ sont **rééchantillonnées** avec remplacement selon leur importance w. Le rééchantillonnage permet d'avoir une distribution cible p différente de la distribution proposée π.

La figure 4.2 présente trois itérations de AMCL avec un robot se déplaçant en une seule dimension et avec un détecteur de porte.

Pour le reste de cette thèse, l'implémentation d'AMCL utilisée est celle de Fox (2003) [3]. Elle utilise l'échantillonnage adaptatif Kullback-Leibler qui permet de minimiser l'erreur entre la vraie distribution p et son estimation par échantillonnage en faisant varier le nombre de particules automatiquement.

2. Figures tirées de http://en.wikipedia.org/wiki/Monte_Carlo_localization et distribuées sous la licence Creative Commons BY-SA.
3. L'implémentation est disponible dans ROS http://wiki.ros.org/amcl.

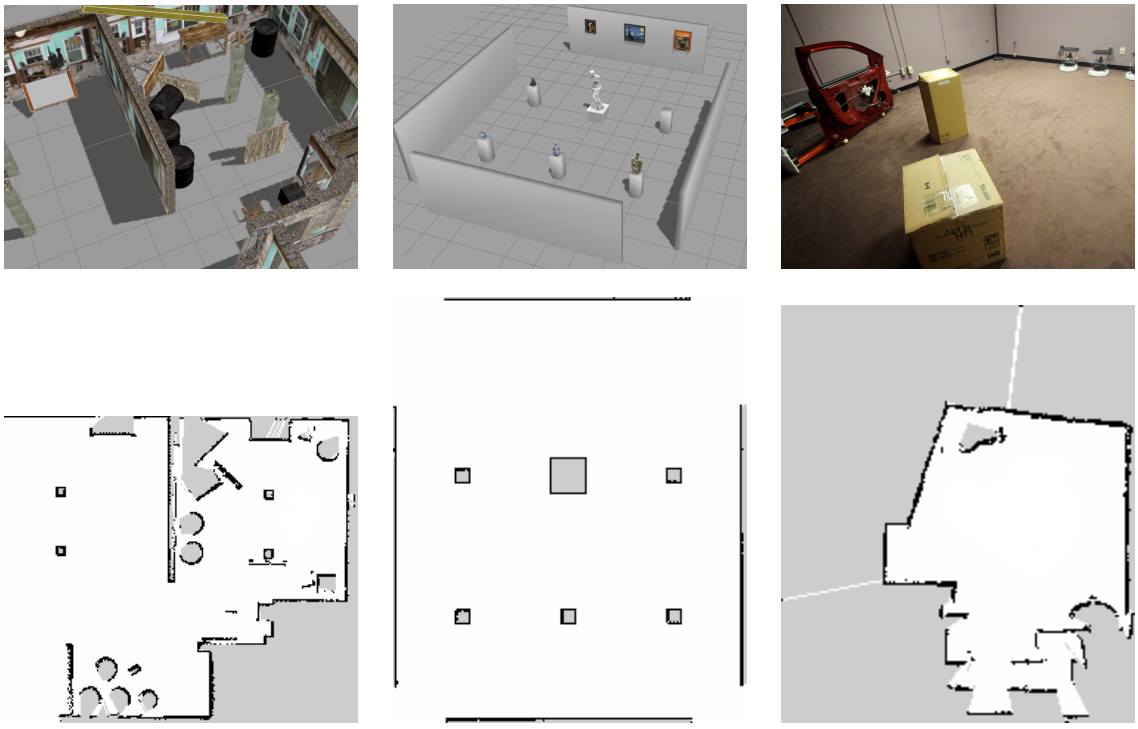
4.1.2 Localisation et cartographie simultanées

La localisation et la cartographie simultanées (SLAM, de l'anglais *Simultaneous Localisation and Mapping*) consistent à cartographier l'environnement tout en maintenant activement la position du robot. Ces deux problèmes sont intrinsèquement liés ; une carte fidèle de l'environnement est nécessaire pour la localisation et une localisation précise est nécessaire pour la cartographie. Ceci fait de la SLAM l'un des problèmes fondamentaux de la robotique mobile. Deux versions du problème de SLAM existent : la SLAM *complète* se fait une fois toutes les données recueillies et la SLAM *en ligne* se fait récursivement et permet d'estimer la position actuelle d'un robot. L'un des algorithmes de SLAM les plus répandus est l'algorithme *FastSLAM* notamment pour sa rapidité d'exécution et sa capacité à résoudre les deux versions du problème.

Dans le FastSLAM, Murphy (1999) et Grisetti et collab. (2007) utilisent un filtre à particules *Rao-Blackwellisé* pour représenter la probabilité conjointe a posteriori $p(m, x_{1:t} \,|\, z_{1:t}, u_{1:t-1})$ de la carte m et de la trajectoire $x_{1:t} = x_1, \dots x_t$ en fonction des mesures des capteurs $z_{1:t} = z_1, \dots z_t$ et de l'odométrie $u_{1:t-1} = u_1, \dots u_{t-1}$ du robot. Le processus de *Rao-Blackwellisation* leur permet de réduire considérablement la dimension de l'espace d'états. Cette opération est nécessaire car les filtres à particules standards sont affectés par la malédiction de la dimensionnalité : leur complexité dépend exponentiellement de la dimension de l'espace d'état. Le théorème de Rao-Blackwell permet de maintenir des particules uniquement pour estimer la probabilité *a posteriori* de la trajectoire du robot $p(x_{1:t} \,|\, z_{1:t}, u_{1:t-1})$ et de construire une carte pour chacune de ces particules puisqu'une description complète de la carte $p(m \,|\, x_{1:t}, z_{1:t})$ peut être calculée analytiquement par cartographie avec position connue (Moravec, 1988). Le reste du filtre à particules estimant la position du robot est similaire à celui utilisé par AMCL. La SLAM n'étant qu'un outil permettant au système décrit dans cette thèse de se localiser, le lecteur est référé au chapitre de Thrun et Leonard (2008) du manuel de robotique pour plus amples détails. Il est aussi important de noter qu'il existe des versions multirobots de la SLAM (Thrun et Liu, 2003; Burgard et collab., 2005).

Dans la présente thèse, la SLAM est utilisée préalablement à toutes les expériences pour créer une carte de l'environnement. Cette cartographie permet de réduire la charge computationnelle sur les unités de calculs. L'implémentation utilisée dans cette thèse est celle de Grisetti et collab. (2007), aussi nommée *GMapping* [4]. Trois cartes construites par l'algorithme sont présentées à la figure 4.3.

4. L'implémentation est disponible dans ROS http://wiki.ros.org/gmapping et sur OpenSLAM http://openslam.org.



(a) Bâtiment endommagé (b) Musée (c) Porte de voiture

FIGURE 4.3 – Cartes d'occupation construites lors de simulations et expérimentations réelles par l'algorithme de SLAM GMapping. Les zones blanches, grises et noires sont respectivement libres, inconnues et occupées.

4.2 Navigation

La seconde phase du système consiste à déplacer un ou plusieurs robots pour acquérir le modèle tridimensionnel de l'environnement ainsi que l'information nécessaire à la reconstruction du point de vue virtuel. Le premier déplacement ordonné à un robot lors de l'initialisation du système est de se rendre le plus près possible d'où se trouve la caméra virtuelle. Ce premier déplacement, qui précède toute forme d'optimisation, permet d'initialiser le modèle tridimensionnel de la zone d'intérêt. Les déplacements subséquents sont effectués suite à l'optimisation des points de vue des capteurs. Ils servent à raffiner le modèle et à acquérir les points de vue réels nécessaires à la reconstruction du point de vue virtuel.

Pour se déplacer, les robots doivent être localisés et avoir une connaissance de leur entourage immédiat. Dans notre système, la première condition est remplie par l'un ou l'autre des algorithmes de localisation présentés à la section précédente et la seconde est satisfaite par l'utilisation d'un capteur de télémétrie placé sur chaque robot. Avec l'aide de ces deux outils, l'itinéraire que doit parcourir un robot pour se rendre de sa position actuelle à la position

demandée est calculée par le module de planification de trajectoire. Dans cette section, nous décrirons, à la section 4.2.1, l'algorithme de planification de trajectoire, à la section 4.2.2, le mécanisme permettant aux robots d'éviter d'entrer en collision entre eux lors des déplacements et, à la section 4.2.3, la navigation des robots aériens.

4.2.1 Planification de la trajectoire

La planification de la trajectoire s'effectue localement sur chaque robot à chaque fois que ce dernier reçoit une nouvelle commande. Conformément à Marder-Eppstein et collab. (2010), elle se fait à deux niveaux, soit une planification grossière globale et un raffinement local. Tout d'abord, la planification globale emploie la carte de l'environnement pour produire un plan de haut niveau du trajet du robot. Pour améliorer sa performance en termes de temps de calcul, cette planification ne tient pas compte de la forme exacte du robot. Elle fait plutôt l'hypothèse d'un robot circulaire et emploie l'algorithme de Dijkstra dans la carte de l'environnement où les obstacles ont été dilatés du rayon du robot suivant la somme de Minkowski. Pour ces raisons, le plan de haut niveau est optimiste et potentiellement invalide dû à des obstacles qui ne sont pas représentés sur la carte ou une trajectoire empruntant un chemin trop serré.

La planification locale raffine le plan global dans une fenêtre restreinte autour de la position actuelle du robot tout en essayant de rester le plus fidèle possible au plan original. L'algorithme de planification locale utilise l'approche de fenêtre dynamique (DWA, de l'anglais *Dynamic Window Approach*) introduite par Fox et collab. (1997) et illustrée à la figure 4.4. L'algorithme va comme suit :

1. Échantillonner l'espace de contrôle du robot, soit les vitesse selon ses degrés de liberté (par exemple, dx et dθ pour un robot à conduite différentielle terrestre).
2. Simuler sur une courte période la trajectoire du robot pour chaque échantillon.
3. Évaluer chaque trajectoire en fonction de sa distance au plan original, de sa distance au but et de sa distance aux obstacles.
4. Appliquer les commandes ayant obtenu le plus haut score pondéré des objectifs évalués.
5. Répéter.

Contrairement à la planification globale, le comportement de la planification locale peut être modifié par de nombreux paramètres. Ainsi, à titre d'exemple, les propensions du robot à suivre le plan original, à s'approcher du but et à s'éloigner des obstacles sont modifiées par les poids associés à ces mesures.

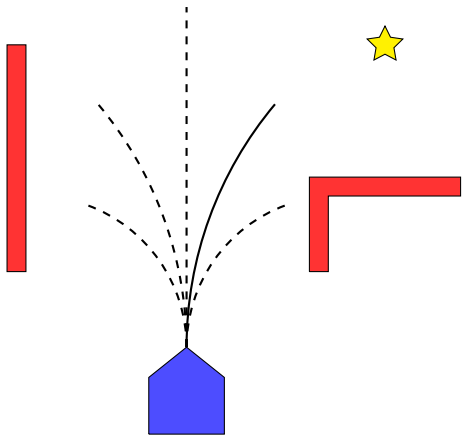
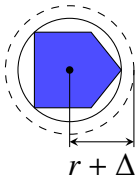
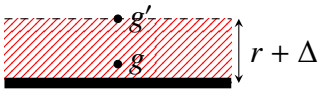


FIGURE 4.4 – Trajectoires simulées par l'algorithme DWA pour atteindre le but indiqué par une étoile tout en évitant les obstacles. La trajectoire représentée par une ligne pleine est celle obtenant l'évaluation la plus favorable, ses paramètres de commandes sont appliqués au robot.



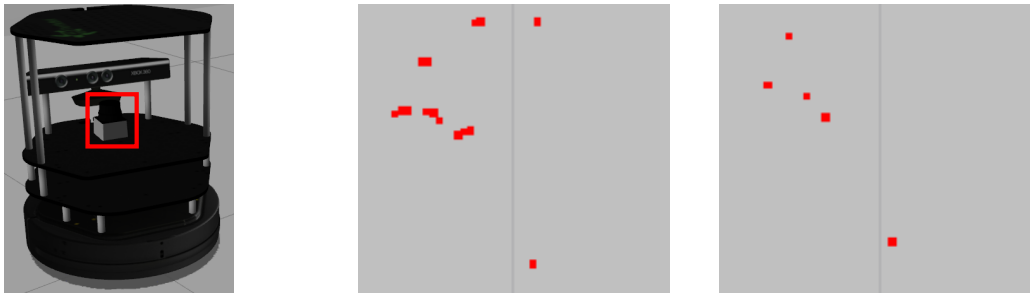
(a) Rayon r du cercle englobant l'empreinte d'un robot augmenté d'une zone de sûreté Δ.



(b) Gonflement des obstacles de $r+\Delta$ et déplacement du but original invalide g vers le but valide g'.

FIGURE 4.5 – Placement d'un capteur lors de l'envoi d'une position trop près d'un obstacle.

À l'intérieur du système proposé, aucun mécanisme n'est nécessaire pour vérifier la validité des positions envoyées aux robots. En effet, toutes les positions envoyées par l'algorithme d'optimisation sont nécessairement à l'extérieur des obstacles présents dans l'environnement. De par la nature de la fonction objectif et du critère de visibilité, une position de capteur à l'intérieur d'un obstacle se voit automatiquement attribuer une erreur maximale, car, de ce point de vue, le capteur ne peut voir aucune surface de la cible désignée $\mathcal{I}$. Ainsi, ces positions sont automatiquement écartées en faveur de positions valides. Toutefois, l'optimisation peut choisir une position de capteur arbitrairement près d'un obstacle. Puisque les robots ont une taille non nulle, le placement physique du capteur à cet endroit est impossible. Pour pallier à ce problème, la planification tente de placer le capteur le plus près possible de la position demandée. La planification locale gonfle les obstacles du rayon du cercle circonscrivant l'empreinte au sol du robot plus une zone de sûreté et elle utilise la position valide la plus près en dehors de ces obstacles dilatés, tel qu'illustré à la figure 4.5.



(a) Emplacement du capteur de télémétrie.

(b) Donnée captée lorsqu'un second robot est placé à 50 cm du premier robot.

(c) Donnée captée lorsqu'un second robot est placé à 150 cm du premier robot.

FIGURE 4.6 – Problème engendré par le dégagement du capteur de télémétrie.

4.2.2 Évitement de collisions interrobots

L'utilisation d'un capteur de télémétrie à grand-angle, tel un lidar Hokuyo, requiert une large zone de dégagement à la hauteur du capteur. Afin de protéger ce dernier des impacts, le capteur est généralement placé sur le dessus ou sur un étage complètement dégagé du robot. Tel que le présente la figure 4.6, si tous les robots sont similaires, seulement une très faible zone du robot peut être détectée par le capteur et la probabilité de détection de cette mince zone diminue avec la distance entre les robots. En ajoutant à cela la vitesse de déplacement des robots, il devient presque impossible à deux robots de se percevoir.

En simulation, puisque nous n'avons pas à nous soucier du risque de briser le capteur, nous réglons ce problème en plaçant le lidar directement devant la base du robot. Il peut ainsi détecter facilement la base des autres robots et permettre l'évitement de collisions. Par contre, lors des expérimentations réelles, cette configuration n'est pas possible, car en cas de défectuosité, nous risquons d'endommager le capteur. Pour pallier au problème de visibilité des robots, chaque unité publie un nuage de points ayant la forme de son empreinte à sa position actuelle. Ce nuage de points est utilisé par tous les autres robots lors de la mise à jour de leur carte des obstacles. Ainsi, cette technique nous a permis d'éviter de nombreuses collisions lors des expérimentations réelles. Cependant, cette méthode demeure imparfaite. Parmi ses défauts de cette méthode, on compte sa déficience à représenter le mouvement des robots, sa dépendance au système de localisation et son inextensibilité pour de grands réseaux de capteurs. Cette thèse ne s'attardera pas plus longuement à ces problèmes puisqu'ils sont hors de notre sujet principal, soit l'optimisation du placement des capteurs et non leur navigation en essaim.

4.2.3 Navigation des robots aériens

Jusqu'à maintenant, la navigation n'a été décrite que pour des robots se déplaçant au sol. Cependant, lors de nos simulations, nous employons également des robots aériens. Ceux-ci, en plus de se déplacer dans le plan horizontal, peuvent ajuster leur altitude, ce qui ajoute une difficulté supplémentaire à la planification de trajectoire et l'évitement de collisions. Malgré l'existence de plusieurs approches spécialisées pour surmonter ces difficultés (Bortoff, 2000; Sasiadek et Dulęba, 2000; De Filippis et collab., 2012; Lin et Saripalli, 2014), nous avons opté pour l'ajout d'un régulateur proportionnel-intégral-dérivée (PID) à l'approche DWA pour contrôler l'altitude des véhicules aériens.

Lorsqu'un robot aérien reçoit une position tridimensionnelle, le PID ajuste l'altitude du robot le plus tôt possible durant son trajet. L'ajustement est fait alors que deux capteurs mesurent le dégagement au-dessus et au-dessous du véhicule. Dès qu'aucun obstacle vertical n'est détecté, le robot règle son élévation et poursuit sa course dans le plan horizontal tel que lui indique l'algorithme DWA. Bien entendu, ce contrôleur ne peut planifier une trajectoire faisant passer un robot par une fenêtre. Néanmoins, il a l'avantage de ne requérir aucune représentation du monde en trois dimensions, il garde la même rapidité d'exécution que DWA et il s'avère suffisant pour effectuer le placement de capteurs hétérogènes dans de nombreux environnements tridimensionnels [5].

4.3 Optimisation

La troisième étape du système consiste à trouver le nombre et la position des capteurs. Cette phase est réalisée par l'algorithme d'optimisation décrit au chapitre 3. Tout au long d'un déploiement de capteurs typique, l'algorithme d'optimisation est appelé à fournir les configurations intermédiaires et finales de capteurs. En effet, l'optimisation s'effectue en alternance avec la phase de déplacement des capteurs présentée à la section précédente. Après un nombre fixe d'itérations, l'optimisation envoie la solution trouvée aux robots avant de continuer sa recherche lorsque ceux-ci arrivent à destination.

Cette alternance des phases de déplacements et d'optimisation permet d'explorer et de construire le modèle de l'environnement aux abords de la zone d'intérêt $\mathcal{I}$. Comme mentionné à la section 2.2, le calcul de la qualité du placement des capteurs s'effectue à l'aide d'une grille d'occupation tridimensionnelle modélisant l'environnement. Ce type de représentation permet de modéliser l'état du monde par des cellules pouvant prendre trois états : occupé,

5. L'implémentation de la couche PID contrôlant l'altitude des robots aériens se trouve à http://github.com/fmder/base_local_planner_3d.

libre ou inconnu. Ce dernier état est d'une importance capitale, car il permet de différentier, sans coût supplémentaire, les sections du monde explorées de celles restant à prospecter. En considérant les zones inconnues de la même manière que celles occupées, soit en requérant leur observation, l'algorithme d'optimisation est porté à explorer l'environnement en plaçant les capteurs de telle sorte qu'ils observent ces zones inconnues. Incidemment, l'exploration de la zone d'intérêt $\mathcal{I}$ se fait naturellement lors de l'optimisation.

Dans le reste de cette section, nous intégrerons le contenu des chapitres 2 et 3 pour composer l'algorithme d'optimisation responsable du placement de capteurs. Tout d'abord, à la section 4.3.1, nous introduirons la manière dont est représentée une solution candidate. Puis, nous verrons à la section 4.3.2 les différents aspects de la fonction objectif et son utilisation pour éviter l'empilement des robots. Ensuite, à la section 4.3.3, nous présenterons les spécificités des algorithmes coopératifs quant au calcul de l'amélioration de la solution et de la contribution de chaque capteur. Enfin, nous mettrons de l'avant les spécificités de l'initialisation des capteurs et leur ordre d'ajout lors de l'emploi d'un groupe hétérogène à la section 4.3.4.

4.3.1 Représentation d'une solution

Un capteur caméra $\mathbf{c}$ est entièrement configuré par ses paramètres intrinsèques et extrinsèques. Une partie de ces paramètres est fixée lors de la définition des types de capteurs tandis que l'autre partie est laissée à l'optimisation pour trouver sa valeur optimale. Dans le langage des algorithmes évolutionnaires, une solution candidate $\mathbf{x}$, telle que vue au chapitre 3, est le génotype d'un capteur alors que $\mathbf{c}$, tel que présenté au chapitre 2, représente son phénotype. Le génotype $\mathbf{x}$ à lui seul ne constitue pas un capteur complet. En effet, s'il est associé à différents paramètres fixes, il produira des phénotypes $\mathbf{c}$ différents. La distinction entre le génotype et le phénotype permet d'utiliser une représentation des solutions plus facile à manipuler par l'algorithme d'optimisation. Cependant, cette représentation n'est pas directement évaluable par la fonction objectif. Donc, préalablement à chaque évaluation de la qualité d'une solution candidate $\mathbf{x}$, le phénotype $\mathbf{c}$ doit être construit à partir des valeurs du génotype et des valeurs fixées par le type du capteur.

L'algorithme CMA-ES requiert une représentation de ses solutions candidates sous forme de vecteur de nombres réels. La séparation du génotype et du phénotype nous permet d'utiliser cette représentation même si ce ne sont pas tous les paramètres qui admettent une configuration valide dans $\mathbb{R}$. En effet, alors que la position $[x, y, z]$ et l'orientation $[\rho, \theta, \phi]$ peuvent naturellement être représentées chacune par trois nombres réels, le facteur de zoom ζ, ayant

un effet à la fois sur la focale f et le champ de vue Θ et Φ du capteur, n'est pas défini hors des spécifications du capteur. Autrement dit, une caméra permettant un zoom maximal de 4:1 n'admettra pas de solution pour une valeur ζ à l'extérieur de l'intervalle $[1, 4]$. Dans ce cas, lors de la transformation du génotype en phénotype, on obtient le facteur de zoom ζ à partir de la variable sous forme de nombre réel non borné $\zeta_{\mathbb{R}}$ par une opération du type

$$\zeta = |\zeta_{\mathbb{R}}| \bmod (\zeta_{\max} - 1) + 1. \tag{4.1}$$

Une transformation similaire peut être effectuée pour tout autre paramètre n'étant pas naturellement défini dans $\mathbb{R}$.

Ensuite, l'ensemble de capteurs C formant une solution complète au problème à optimiser est composé d'un capteur (phénotype) de chaque espèce. Lors de l'optimisation, deux approches sont employées pour choisir ces capteurs. La première méthode sert lors de l'évaluation du progrès et de la contribution des espèces de l'algorithme, soit aux lignes 26 et 27 de l'algorithme 3.6, ainsi que lorsqu'une solution est requise pour le déploiement des capteurs. Dans ce cas, l'ensemble C est directement formé par les phénotypes construits à partir de l'ensemble des représentants $\mathfrak{H}$. La seconde option est appliquée lors de l'évaluation des solutions candidates $f(\mathbf{x})$ d'une espèce i, soit à la ligne 11 de l'algorithme 3.6. Pour cette sélection, la solution candidate $\mathbf{x}$ est jointe aux représentants des autres espèces $\{\mathbf{h}_j \mid \mathbf{h}_j \in \mathfrak{H} \wedge j \neq i\}$ et l'union de leur phénotype forme l'ensemble C. Les ensembles C construits par les deux approches sont équivalents. Ils forment tous deux un réseau de capteurs complet répondant plus ou moins bien à la fonction objectif formulée à la section 2.3. La prochaine section présentera les spécificités de cette fonction objectif pour s'assurer de la validité des réseaux de capteurs et la manière dont ceux-ci sont comparés entre eux. Dans cette thèse, nous utiliserons de cette notation en employant $f(\mathbf{c}_{\mathrm{v}}, C)$ pour désigner la qualité d'une solution candidate et $f(\mathbf{c}_{\mathrm{v}}, \mathfrak{H})$ pour désigner la qualité du meilleur réseau de capteurs trouvé jusqu'à maintenant.

4.3.2 Fonction biobjectif et classement

L'évaluation de la qualité d'une configuration de capteurs C se fait par l'équation 2.26. Un algorithme d'optimisation idéal trouvera, avec cette fonction objectif, une ou plusieurs configurations de capteurs minimisant l'écart entre la densité de pixels demandée et celle obtenue. Toutefois, certaines de ces configurations optimales sont invalides dans le monde réel. En effet, les capteurs possèdent des dimensions finies et par conséquent ils ne peuvent occuper le même volume. Afin de contraindre les capteurs à occuper un volume distinct, nous ajoutons un second objectif à la fonction de classement ($\preceq$) employée à la ligne 14 de l'algorithme 3.6. Cette nouvelle fonction objectif $g(C)$, qui servira ultimement de contrainte,

vérifie si les volumes occupés par deux capteurs de C se chevauchent. Afin d'accélérer le calcul, nous posons l'hypothèse de capteurs sphériques. La fonction objectif $g(C)$ teste donc si la distance entre deux capteurs est supérieure à la somme des rayons de leur sphère englobante. La qualité de placement des capteurs d'un ensemble C est alors définie par le tuple biobjectif $\langle g(C), \hat{f}(\mathbf{c}_v, C)\rangle$. L'optimisation biobjective a l'avantage sur l'application d'une contrainte dure d'aider le système à atteindre des solutions valides à partir de solutions invalides.

L'ordonnancement des configurations se fait selon les deux objectifs définis précédemment. Tout d'abord, les configurations sont classées selon le premier objectif $g(C)$, soit le chevauchement entre deux capteurs. Toute configuration n'ayant pas de chevauchement est préférable à une autre en présentant. Ensuite, les configurations comportant une superposition égale sont ordonnées en minimisant la différence entre la densité de pixels désirée et celle obtenue, soit le second objectif $\hat{f}(\mathbf{c}_v, C)$. Ce classement est généralement appelé tri lexicographique. Il permet à l'optimisation de mettre l'accent sur un premier objectif, puis lorsque l'optimum est atteint pour cet objectif, il axe sa recherche sur le second objectif. Dans le présent cas, puisque la grande majorité des solutions ne présente pas de recouvrement entre les capteurs, cette technique ne fait qu'écarter d'emblée toute solution impossible physiquement et assure la validité de l'ensemble C tout au long de l'optimisation.

4.3.3 Stagnation et contribution

La détection de la stagnation et le calcul de la contribution jouent un rôle déterminant dans le succès des algorithmes d'optimisation coopératifs. La combinaison de ces deux processus entraîne la sélection d'un nombre adéquat d'espèces pour résoudre le problème. La détection tardive de la stagnation fait perdre d'importantes générations d'optimisation. Et une contribution minimale trop laxiste permet à des capteurs presque inutiles de joindre le réseau alors que lorsqu'elle est trop stricte, elle écarte prématurément des capteurs avec un fort potentiel. Toutefois, comme mentionné par Potter et De Jong (2001), ces deux mécanismes dépendent du problème et aucune méthode ne garantit leur sélection adéquate.

Tout d'abord, la détection de la stagnation s'effectue en surveillant l'amélioration de la solution proposée par l'algorithme d'optimisation, soit celle formée par les phénotypes de l'ensemble des représentants $\mathfrak{H}$. Nous remarquons que la valeur de ce critère n'est pas très importante, mais qu'il suffit plutôt de détecter le moment où le système s'essouffle avec les capteurs présents. De plus, nous notons que l'amélioration est beaucoup plus facile en début d'évolution car de grandes zones ne sont pas observées, alors qu'à la fin, lorsque de nombreux capteurs sont présents, plusieurs d'entre eux doivent coopérer pour faire des gains.

Ainsi, nous introduisons un critère adaptatif dépendant de la valeur de la solution. L'amélioration au cours des l dernières générations, calculée comme étant la différence entre l'erreur normalisée de densité de pixels de l'ensemble $\mathfrak{H}$ à la génération présente g et celle d'il y a l générations, doit être supérieure ou égale à une fraction de la valeur de la solution d'il y a l générations, sinon le système détecte une stagnation, soit

$$\hat{f}^{(g-l)}(\mathbf{c}_\mathrm{v}, \mathfrak{H}) - \hat{f}^{(g)}(\mathbf{c}_\mathrm{v}, \mathfrak{H}) \geq T_\mathrm{a} \hat{f}^{(g-l)}(\mathbf{c}_\mathrm{v}, \mathfrak{H}). \tag{4.2}$$

Ensuite, la contribution des espèces permet d'éliminer celles ne contribuant pas suffisamment à la solution globale après un certain laps de temps, tout en gardant celles ayant un potentiel élevé. Dans le contexte du placement de capteurs avec des environnements et des zones d'intérêt changeants, il est difficile de trouver un seuil de contribution fixe pour chaque espèce. En effet, plus la solution requiert un nombre élevé de capteurs, moins chaque capteur abaisse l'erreur normalisée de densité de pixels individuellement. Par conséquent, nous introduisons un critère de contribution variable en fonction du nombre d'espèces présentes dans le réseau de capteurs. Pour être significative, la contribution d'une espèce i, calculée comme la différence entre l'erreur normalisée de densité de pixels de l'ensemble $\mathfrak{H}$ avec et sans le représentant de l'espèce i, doit être supérieure ou égale à une fraction, dépendante du nombre de capteurs, de la contribution de toutes les espèces,

$$\hat{f}(\mathbf{c}_\mathrm{v}, \mathfrak{H} \backslash \mathbf{h}^i) - \hat{f}(\mathbf{c}_\mathrm{v}, \mathfrak{H}) \geq \frac{T_\mathrm{c}}{\|\mathfrak{H}\|} \left(1 - \hat{f}(\mathbf{c}_\mathrm{v}, \mathfrak{H})\right), \tag{4.3}$$

où $\hat{f}(\cdot) \in [0, 1]$, $\hat{f}(\mathbf{c}_\mathrm{v}, \mathfrak{H} \backslash \mathbf{h}^i) \geq \hat{f}(\mathbf{c}_\mathrm{v}, \mathfrak{H})$ et $\|\mathfrak{H}\|$ est la taille du réseau de capteurs. Ce critère encourage donc les espèces à contribuer de manière plus ou moins égale à la solution trouvée, ce qui représente intuitivement la solution optimale. Lorsque l'environnement est morcelé en sections de taille inégale, le paramètre T_c devra prendre une valeur plus faible pour permettre d'observer les plus petites sections.

4.3.4 Banque de capteurs et initialisation

L'ajout d'un capteur lors de la stagnation de l'algorithme 3.6 est directe lorsqu'un seul type de capteurs est employé. Le système décrit dans cette thèse propose l'utilisation d'un groupe de capteurs hétérogènes pour lequel le nombre de capteurs de chaque type peut varier. Afin de trouver la bonne combinaison de capteurs, l'algorithme doit accorder la même chance à tous les types de capteurs, peu importe leur nombre. Pour ce faire, chaque fois qu'un nouveau capteur est demandé, son type est choisi à la ronde. Si un type de capteur n'est plus disponible, car tous les capteurs de ce type sont utilisés, l'algorithme passe au suivant. Lorsqu'une

espèce est retirée de la population, une instance de son type est remise dans la banque de capteurs pour utilisation ultérieure. Enfin, lorsqu'il n'y a plus de capteurs d'aucun type, l'algorithme n'ajoute tout simplement plus de nouvelles espèces à la population et l'optimisation se poursuit avec les capteurs présents.

Lors de l'introduction d'une nouvelle espèce, le processus d'optimisation du nouveau capteur doit être initialisé judicieusement. Plutôt que de laisser l'algorithme d'optimisation globale trouver lui-même un bon point de départ, chaque nouvelle stratégie CMA-ES est introduite avec sa moyenne $\mathbf{x}_{\text{parent}}$ centrée sur la position du capteur virtuel $\mathbf{c}_{\text{v}}$ et son pas d'optimisation σ initialisé à 0,3. Cette méthode permet d'avoir une bonne diversité initiale, soit 99,6% des solutions candidates initiales se trouveront dans un rayon d'environ 1 m du capteur virtuel avec une orientation de ±51°. De plus, lorsqu'une bonne diversité de solutions candidates est présente dans la population, CMA-ES ajuste automatiquement le pas d'optimisation aux besoins de la fonction objectif. Par conséquent, le pas d'optimisation initial σ n'a que très peu d'importance tant qu'il est choisi de manière proportionnelle à la taille de la scène d'intérêt.

Pour le reste de cette thèse, l'implémentation de CMA-ES utilisée est celle de l'outil d'algorithmes évolutionnaires DEAP (Fortin et collab., 2012) (de l'anglais *Distributed Evolutionary Algorithms in Python*). Ce logiciel a été développé dans le cadre des recherches doctorales, en collaboration avec Félix-Antoine Fortin, afin de fournir à la communauté un cadre logiciel d'algorithmes évolutionnaires facile à modifier. Sans ce logiciel, les expérimentations présentées dans la seconde partie de cette thèse n'auraient pas été possibles.

4.4 Reconstruction de la vue virtuelle

La quatrième et dernière étape du processus implanté par la système est de générer la vue virtuelle demandée par l'utilisateur. Cette étape peut s'effectuer en parallèle au placement des capteurs ou lorsque l'optimisation a atteint son optimum. La construction de la vue virtuelle requiert la combinaison de l'ensemble de l'information acquise par tous les capteurs en une ou plusieurs vues compréhensibles par l'utilisateur. La présente section revoit deux techniques couramment rencontrées pour assembler l'information contenue dans de multiples images. Nous décrirons donc brièvement l'assemblage d'images [6] dans la suite de cette section.

L'assemblage d'images est une technique très répandue et couramment étudiée permettant de coudre ensemble plusieurs images prises de différent points de vue d'une même scène

6. Traduction de *image stitching*

(Pérez et collab., 2003; Agarwala et collab., 2004; Eden et collab., 2006; Brown et Low, 2007). L'assemblage d'images consiste à trouver une matrice d'homographie $\mathbf{H}$ transformant les points $\mathbf{x} = [x\ y]^T$ de l'image I en $x' = [x'\ y']^T$ de l'image I' telle que $\tilde{\mathbf{x}}' \sim \mathbf{H}\tilde{\mathbf{x}}$, avec $\tilde{\mathbf{x}}$ représentant $\mathbf{x}$ en coordonnées homogènes. D'entrée de jeu, la matrice de transformation $\mathbf{H}$, de taille 3×3, est estimée par la transformation linéaire directe (Zhang, 1997) d'un ensemble de points associés $\{\mathbf{x}_i, \mathbf{x}'_i\}$ des images I et I'. Puis, l'homographie estimée $\hat{\mathbf{H}}$ est appliquée à tous les pixels de I afin de les transformer dans le repère de I'. Cette technique pose deux restrictions majeures :

1. les vues ne doivent différer que par pure rotation, ou
2. la scène doit être plane.

Ces deux limitations restreignent son utilisation dans notre système. En effet, dans le système décrit jusqu'à maintenant, la taille finie des robots et l'environnement tridimensionnel non convexe dans lequel ils évoluent brisent respectivement l'une et l'autre de ces restrictions. Ceci crée des erreurs d'alignement et de parallaxe dans l'image reconstruite. Pour corriger ces erreurs, Zaragosa et collab. (2014) proposent de diviser l'image I en cellules uniformes et d'effectuer la transformation linéaire directe pour chaque cellule en utilisant une décomposition en valeurs singulières pondérée (WSVD, de l'anglais *Weighted Singular Value Decomposition*). Les poids de chaque WSVD sont choisis en fonction de la distance entre les points $\mathbf{x}_i$ et le centre de la cellule. L'homographie ainsi trouvée s'applique uniquement aux points $\mathbf{x}_i$ à l'intérieur de cette cellule. Cet algorithme permet d'assouplir les restrictions posées par l'homographie appliquée à l'image entière et donne de meilleurs résultats lorsque ces contraintes ne sont pas entièrement respectées. La figure 4.7 présente le résultat de l'assemblage de trois images de la mosaïque de Jordi Bonet du pavillon Adrien-Pouliot de l'Université Laval avec un appareil Canon Rebel SL1 et redimensionnées à 2048 par 1365 pixels. Dans le reste de cette thèse, cette technique est utilisée lors de l'assemblage d'images [7].

4.5 Conclusion

Dans ce chapitre, nous avons présenté le fonctionnement général du système ainsi que détaillé chacune de ses parties. Plus spécifiquement, nous avons vu que le déroulement d'une expérimentation se fait en quatre phases. La première phase consiste à localiser les robots dans leur environnement. Celle-ci peut se faire par simple localisation lorsqu'une carte est

7. L'implémentation de l'assemblage d'images utilisée est disponible à http://cs.adelaide.edu.au/~jzaragoza.

FIGURE 4.7 – Résultant de l'assemblage de trois images prises de la mosaïque de Jordi Bonet du pavillon Adrien-Pouliot de l'Université Laval avec un appareil Canon Rebel SL1 par l'algorithme de Zaragosa et collab. (2014).

disponible ou par localisation et cartographie simultanées dans le cas contraire. La localisation permet de transformer les données de capteurs du repère du robot à celui du monde et d'amorcer la navigation lorsqu'une commande est reçue. La seconde phase est de déplacer les robots aux positions commandées. Ces déplacements requièrent l'élaboration d'une trajectoire qui évite les obstacles de l'environnement et les collisions entre robots. Cette stratégie a aussi été ajustée à la navigation de robots aériens pouvant modifier leur altitude. La troisième phase, s'effectuant en alternance avec la seconde, permet d'optimiser les points de vue desquels observer la scène d'intérêt de manière à amasser toute l'information nécessaire à sa reconstruction à partir du point de vue choisi par l'utilisateur. Les différents aspects de cette phase ont été présentés en détail, soit la représentation par vecteur de nombres réels d'une solution, l'évaluation bi-objectif des solutions candidates, les critères de stagnation et de contribution des espèces et la sélection des types de capteurs et leur initialisation. Nous avons aussi vu que cette phase joue un rôle essentiel dans l'exploration de l'environnement aux abords de la scène d'intérêt. Finalement, la quatrième et dernière phase présentée permet de visionner l'information recueillie par les capteurs de manière centralisée. La méthode présentée est l'assemblage d'images permettant de coudre plusieurs images les unes aux autres. Le système présenté sera mis à l'épreuve dans les prochains chapitres, tant lors de simulations que d'expérimentations réelles, pour illustrer les solutions proposées par le placement de capteurs que cette thèse vise à solutionner.

Deuxième partie

Expérimentations et résultats

Chapitre 5

Simulations

Nous présentons maintenant les résultats de la mise en œuvre du système de déploiement de capteurs décrit dans les chapitres précédents dans un contexte de simulation. Deux types de simulations sont exposées. Dans les premières simulations, discutées à la section 5.1, on demande que des robots aériens munis d'une caméra regardant vers le sol observent entièrement un polygone non convexe représentant la scène d'intérêt $\mathcal{I}$. Cette première simulation sert à comparer les performances du système proposé face à l'état de l'art en matière de placement de capteurs observant une zone d'intérêt bidimensionnelle. Quant aux secondes simulations, présentées à la section 5.2, elles généralisent les résultats obtenus à des environnements pleinement tridimensionnels de différentes complexités géométriques et employant des robots terrestres et aériens.

5.1 Simulations bidimensionnelles

Afin de montrer le bon fonctionnement de notre approche et de comparer sa performance à celle d'autres méthodes, nous débutons avec des simulations où les capteurs, placés dans un environnement en trois dimensions, observent une scène d'intérêt bidimensionnelle sans occlusion. En outre, dans ce scénario, la géométrie de la scène d'intérêt $\mathcal{I}$ est connue de l'algorithme. Ainsi, les contrôleurs basés sur un gradient sont applicables. Cinq algorithmes seront comparés au cours de ces simulations, qui reproduisent celles de Schwager et collab. (2011a). Le premier représente l'état de l'art en matière de placement de capteurs, soit le contrôleur à gradient décentralisé (CGD) tel que présenté par Schwager et collab. (2011a). Les deux algorithmes suivants sont les algorithmes de coévolution coopérative (ACC) et de coévolution coopérative efficiente (ACC-MAB), tout deux employant CMA-ES pour l'optimisation intraespèce tels que présentés au chapitre 3. Les deux derniers algorithmes servent

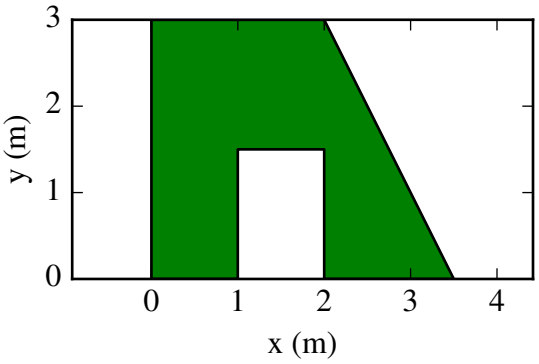


FIGURE 5.1 – Polygone à observer lors des simulations bidimensionnelles.

de contrôle. Le premier est un algorithme procédant au placement de capteurs de manière globale (Global) utilisant une seule stratégie CMA-ES pour tous les capteurs à la fois. Le second procède au placement de manière vorace (Vorace). Il utilise une stratégie CMA-ES par capteur mais effectue le placement un capteur à la fois sans déplacer les capteurs déjà optimisés. Ces deux derniers algorithmes permettent de déterminer si la complexité ajoutée par les algorithmes coopératifs permet d'obtenir des résultats significativement meilleurs tel qu'anticipé. Deux types de simulations bidimensionnelles sont effectuées. Le premier teste la capacité des algorithmes à trouver une solution satisfaisante alors que le nombre de capteurs est fixé tandis que le second vérifie plutôt leur capacité à trouver un nombre adéquat de capteurs.

L'environnement utilisé pour les deux types de simulations avec scène d'intérêt bidimensionnelle est présenté à la figure 5.1. La zone verte représente la cible à observer. Les capteurs sont des caméras orientées vers le bas (axe optique perpendiculaire au plan xy), pouvant se déplacer selon les trois axes x, y et z et tourner au tour de l'axe z. Les capteurs ont une longueur focale $f = 10\,\text{mm}$, une largeur de pixel $u = v = 10\,\mu\text{m}$ et un champ de vue de $\Theta = 70°$ par $\Phi = 40°$ produisant une caméra ayant environ 1,02 Mpx telle que celle utilisée par Schwager et collab. (2011a).

Les paramètres utilisés pour ces simulations sont les suivants. Tout d'abord, pour les cinq algorithmes, la position initiale de chaque capteur est choisie aléatoirement dans l'espace à observer à une altitude entre 30 et 40 cm et une rotation entre 0 et 360°. Ensuite, la connaissance *a priori* de l'environnement du contrôleur CGD est fixée à $w = 2^{16}$ et les gains à $k = 10^{-6} \times [1\ 1\ 0{,}1\ 10^{-9}]$ de manière similaire à ce qui est suggéré par Schwager et collab. (2011a) pour un environnement comparable. Puis, pour les algorithmes ACC et ACC-MAB, le nombre de solutions candidates générées à chaque itération est de $\lambda = 20$ et dans le cas de

Algorithme	ENDP	Aire par pixel	Nb. gén. sol. min.
AAC	**0,0137 (0,0351)**	**2,615e+09 (6,740e+09)**	301,2 (152,7)
AAC-MAB	**0,0137 (0,0179)**	**3,008e+09 (4,988e+09)**	**113,6 (41,36)**
CGD	–	13,60e+09 (17,55e+09)	387,0 (153,2)
Global	0,0237 (0,0158)	**3,318e+09 (2,477e+09)**	187,6 (12,37)
Vorace	0,0339 (0,0244)	5,570e+09 (8,552e+09)	152,6 (21,98)

TABLE 5.1 – Moyenne (déviation standard) des résultats des simulations bidimensionnelles avec nombre de capteurs fixe. La dernière colonne présente le nombre de générations requises pour trouver la solution avec une erreur normalisée de densité de pixels (aire par pixel) minimale. Les valeurs en gras sont significativement meilleures selon le test Mann-Whitney U avec un niveau de confiance supérieur à 99,9%.

ACC-MAB, la taille de la fenêtre du bandit est fixée à $W = 50$, le facteur d'amortissement $L = 1$ et le facteur d'exploration $C = 1$, conformément aux expérimentations précédentes lors de l'appariement de chaînes (section 3.3). Par la suite, le nombre de solutions candidates des algorithmes Global et Vorace est fixé respectivement à $\lambda = 20N$ et $\lambda = 20$, où N est le nombre de capteurs disponibles. Pour sa part, l'algorithme Vorace ajoute une nouvelle espèce, jusqu'à concurrence de N, si la solution globale ne s'est pas améliorée de 0,1% au cours des 10 dernières itérations. Finalement, tous les algorithmes sont arrêtés si leur solution ne s'est pas améliorée au cours des $10n$ dernières itérations, où $n \in [1, N]$ est le nombre de capteurs utilisés par l'algorithme.

5.1.1 Nombre de capteurs fixe

Les premières simulations bidimensionnelles utilisent un nombre fixe de capteurs $n = N = 5$. Ainsi, ACC et ACC-MAB n'ont pas à ajuster le nombre d'espèces durant l'optimisation. L'aire de la surface à observer, présentée à la figure 5.1, est de 6,75 m^2. Si cette surface était parfaitement divisible par la géométrie de l'empreinte des caméras, une densité de pixels de 0,75 Mpx/m^2 serait atteignable avec cinq caméras ayant 1,02 Mpx et observant chacune 1,35 m^2. Cependant, puisque la géométrie n'est pas parfaitement divisible par l'empreinte des capteurs, il y aura nécessairement des pertes dans la couverture des capteurs. Ainsi, pour rendre possible une couverture complète, la densité de pixels désirée est fixée à 0,3 Mpx/m^2 et est définie comme étant uniforme sur l'entièreté de la surface à observer.

La table 5.1 et la figure 5.2 présentent les résultats de 25 simulations indépendantes des quatre algorithmes utilisant l'erreur normalisée de densité de pixels (ENDP) comme fonction objectif. On y remarque que les deux algorithmes coopératifs performent mieux que leurs concurrents Global et Vorace. En effet, selon un test de Mann-Whitney U, les solutions finales

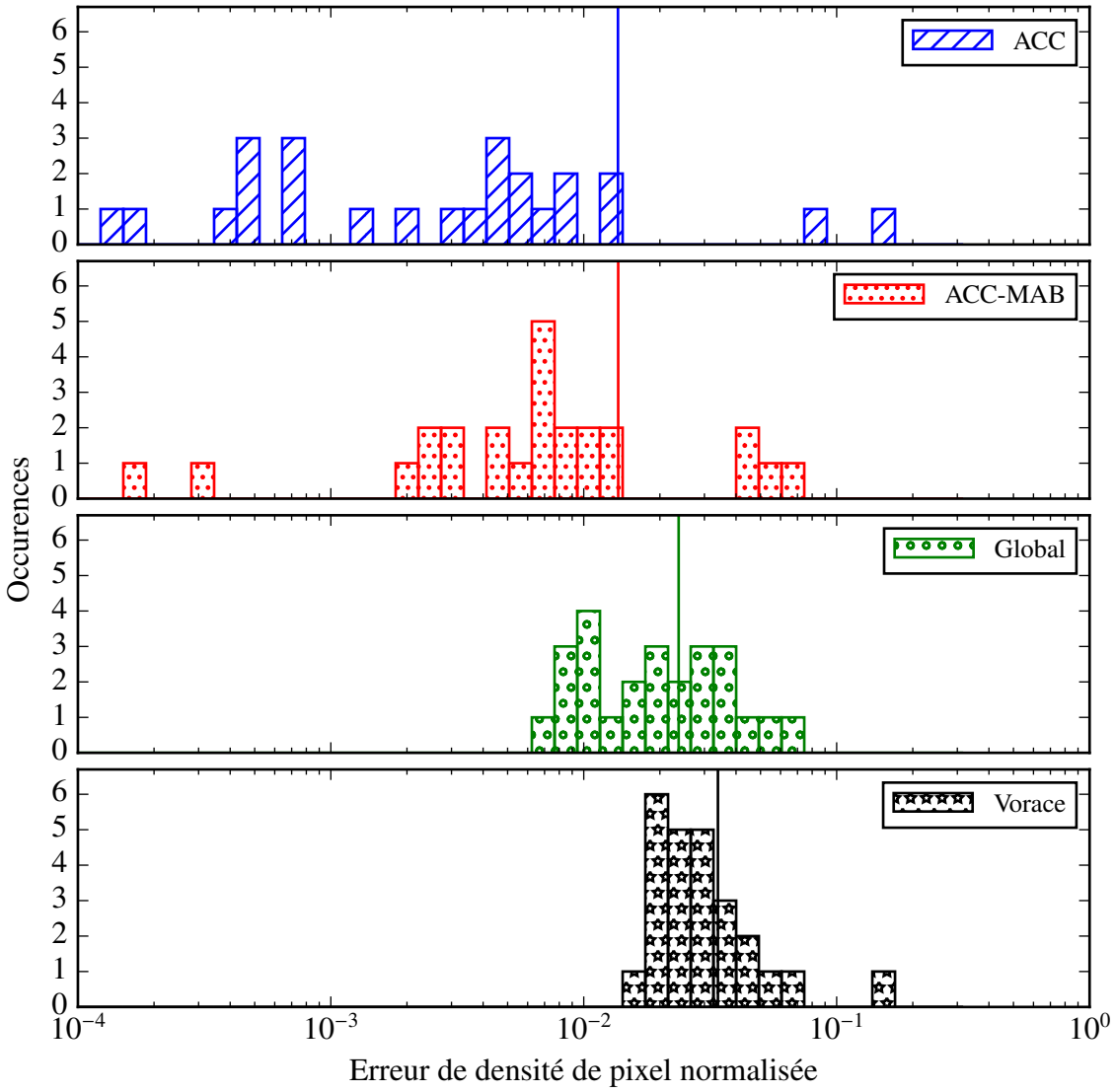


FIGURE 5.2 – Erreur de densité finale pour chaque algorithme, avec la barre verticale indiquant la moyenne des résultats.

de AAC et AAC-MAB surpassent significativement celles de Global et Vorace avec un niveau de confiance supérieur à 99,9%. Par ailleurs, selon ce même test, la différence entre l'ENDP finale pour AAC et AAC-MAB n'est pas significative. Toutefois, on note que le nombre de générations requises à ACC-MAB pour atteindre une configuration avec une ENDP minimale est significativement inférieure celui nécessaire à tous les autres algorithmes. Une itération de CGD prend sensiblement le même temps CPU qu'une génération des autres algorithmes, rendant possible la comparaison.

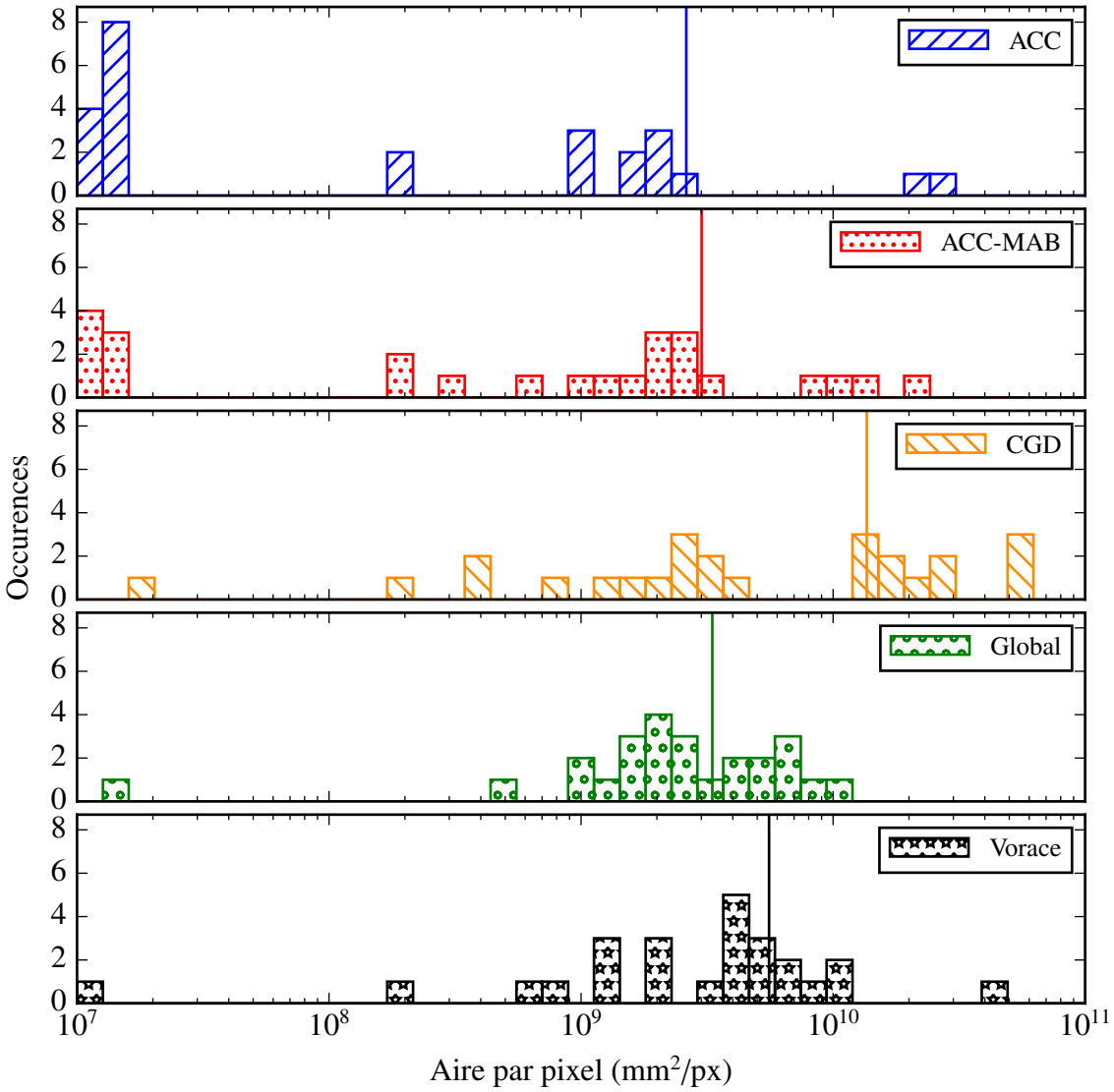


FIGURE 5.3 – Aire par pixel, avec la barre verticale indiquant la moyenne des résultats.

Les résultats de CGD ne sont pas inclus à la figure 5.2 puisque ce dernier algorithme utilise une métrique différente pour le placement de ses capteurs et qu'une comparaison sur cette métrique lui serait défavorable. Afin de comparer les résultats des quatre algorithmes utilisant l'ENDP à ceux de CGD, l'aire par pixel (APP) [1] est calculée pour chacune de leurs 25 configurations présentant l'ENDP minimale, et ces résultats sont présentés à la figure 5.3 et dans la seconde colonne de la table 5.1. Malgré la différence de fonction objectif, on constate que les algorithmes utilisant une optimisation globale sont significativement supérieurs à CGD utilisant une simple montée du gradient. On remarque aussi que les distributions des solu-

1. Métrique présentée par Schwager et collab. (2011a).

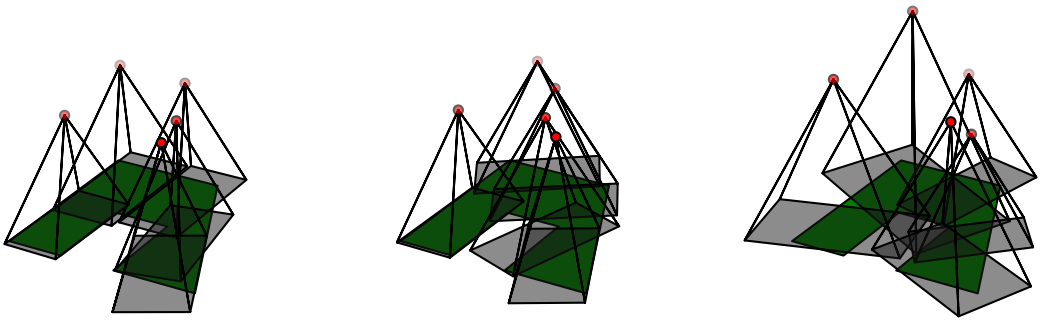
tions finales pour les deux métriques sont sensiblement différentes. Cette différence vient entre autres de l'intégration des points d'intérêts non observés, qui dans le cas de l'ENDP ajoutent une erreur de 1 alors que dans le cas de l'APP ils ont un impact plus important en ajoutant 2^{16} à la valeur de la fonction objectif.

La figure 5.4 montre la configuration finale médiane pour chacun des algorithmes. On y observe que l'avantage des algorithmes utilisant une optimisation globale provient de leur habileté à faire concorder les bornes de l'empreinte des capteurs avec les bornes de l'environnement. En outre, on voit que les capteurs de CGD observent de grandes zones à l'extérieur du polygone d'intérêt alors que ceux de ACC, ACC-MAB, Global et Vorace sont concentrés sur les zones importantes en suivant de près les bornes du polygone. Puisque CGD ne fait que monter le gradient de la fonction objectif, il ne peut se sortir d'une « mauvaise » initialisation, alors que les autres algorithmes en sont moins dépendants. On relève aussi un important empilement des capteurs par l'algorithme Vorace. Cette empilement est dû au fractionnement de l'espace à observer, tel que présent dans les approches meilleure prochaine vue, et de l'impossibilité de réoptimiser un capteur lorsqu'il est placé.

5.1.2 Adaptation du nombre de capteurs

Les secondes simulations bidimensionnelles font varier le nombre de capteurs en plus d'optimiser leur position. Cette section laisse donc les algorithmes ACC, ACC-MAB et Vorace trouver le nombre approprié de capteurs pour la couverture du polygone d'intérêt. Les paramètres relatifs à l'ajout et au retrait d'espèces des algorithmes ACC et ACC-MAB sont un seuil d'amélioration minimale $T_a = 2.5\%$, un délai d'amélioration $T_l = 5n$ itérations et un seuil d'extinction $T_c = \frac{1}{4}$. Les paramètres de Vorace restent quant à eux inchangés. ACC et ACC-MAB débutent l'optimisation avec $n = 2$ espèces et peuvent utiliser un maximum de $N = 10$ espèces, alors que l'algorithme Vorace débute avec $n = 1$ espèce et est aussi limité à dix. Pour vérifier l'adaptation du nombre de capteurs en fonction de la tâche demandée, nous faisons varier la densité de pixels requise, ce qui devrait entraîner un changement du nombre de capteurs utilisés. Le nombre de capteurs N pour les algorithmes CGD et Global est fixé à la valeur moyenne trouvée par ACC, arrondie vers le haut.

La table 5.2 présente la moyenne de l'ENDP, le nombre moyen de capteurs utilisés et le nombre d'évaluations requises pour trouver la solution pour laquelle la fonction objectif est minimale pour trois densités de pixels différentes. Tout d'abord, on remarque que l'algorithme Vorace produit des solutions substantiellement meilleures aux autres algorithmes pour les trois densités requises. Cependant, il utilise un nombre significativement plus élevé



ACC
ENDP = 0,00382
APP = 0,01268e+09 mm^2/px

ACC-MAB
ENDP = 0,00704
APP = 0,1771e+09 mm^2/px

CGD
–
APP = 3,298e+09 mm^2/px

Global
ENDP = 0,01956
APP = 2,145e+09 mm^2/px

Vorace
ENDP = 0,02665
APP = 6,565e+09 mm^2/px

FIGURE 5.4 – Configurations finales obtenues par chaque algorithme pour l'observation de la scène d'intérêt bidimensionnelle avec un nombre fixe de capteurs.

de capteurs, soit environ deux fois plus que ACC et ACC-MAB. En effet, l'algorithme Vorace bénéficie d'un avantage par rapport aux autres algorithmes faisant varier le nombre de capteurs. Il ne prend pas en compte de la contribution de chaque capteur au réseau. Ainsi, comme mentionné précédemment, il peut ajouter des capteurs sans ne jamais en retirer et continuer à faire progresser la fonction d'erreur. Ceci lui évite donc d'être freiné par le critère de stagnation. Pour leur part, les algorithmes coopératifs ne gardent que des capteurs utiles ce qui bloque l'amélioration de la fonction d'erreur lorsque qu'il n'est plus possible de placer un capteur satisfaisant le critère d'utilité. Cette observation se confirme par la réduction de l'avantage de Vorace sur ACC et ACC-MAB lorsque la densité désirée augmente mais que le nombre de capteurs disponibles reste le même.

Ensuite, le nombre d'évaluations pour parvenir à la solution présentant la valeur de fonction objectif minimale, soit le temps de convergence, est plus faible pour Vorace que pour ACC et ACC-MAB pour un nombre de capteurs relativement similaire. De nouveau, puisque Vorace

Densité requise d	Algo.	ENDP	Nb. capteurs	Nb. éval. sol. min.
0.15 Mpx/m^2	ACC	0,0046 (0,0142)	**3,6 (0,49)**	**2508 (767,6)**†
	ACC-MAB	0,0225 (0,0224)	**3,36 (0,48)**	**1895,2 (674,45)**
	CGD	*0,132 (0,0852)*	4 (–)	–
	Global	0,00402 (0,00977)	4 (–)	6611,2 (4778)
	Vorace	**3,14e-06 (1,54e-05)**	7,4 (1,67)	2912,8 (550,2)
0.3 Mpx/m^2	ACC	0,0178 (0,0323)	**5,28 (0,665)**	**5126,4 (1621)**
	ACC-MAB	0,0128 (0,0216)	**5,32 (0,676)**	**5144 (1542,4)**
	CGD	*0,157 (0,0798)*	6 (–)	–
	Global	0,013 (0,0114)	6 (–)	21 494 (4294,2)
	Vorace	**0,000365 (0,000416)**	9,92 (0,271)	**4367,2 (474,06)**
0.5 Mpx/m^2	ACC	0,0235 (0,0274)	**7,56 (1,02)**	10600 (3161,4)
	ACC-MAB	0,0223 (0,041)	**8,08 (1,47)**	12670 (5940,7)
	CGD	*0,209 (0,0637)*	8 (–)	–
	Global	0,0626 (0,0304)	8 (–)	34 899 (1478,5)
	Vorace	**0,0201 (0,00927)**	10 (0)	**5228,8 (515,24)**

TABLE 5.2 – Moyenne (déviation standard) de l'erreur normalisée de densité pixels, du nombre de capteurs requis et du nombre d'évaluations nécessaires pour trouver la solution pour les expérimentations avec un nombre variable de capteurs et trois densités de pixels désirées. Les valeurs en italique notent la méthode où l'optimisation n'a pas été faite sur l'erreur normalisée de densité de pixels et les valeurs en gras sont significativement meilleures selon le test Mann-Whitney U avec un niveau de confiance supérieur à 99,9%.
†Le nombre d'évaluations requises à ACC n'est pas significativement inférieur à Vorace, mais n'est pas non plus significativement supérieur à ACC-MAB.

ne retire jamais de capteurs, il parvient à sa solution présentant une erreur minimale plus rapidement que les algorithmes recherchant des capteurs utiles. Cette rapidité est bien entendue obtenue au prix de configurations nécessitant une plus grande quantité de capteurs pour une erreur relativement similaire. L'étude du nombre d'évaluations requises pour CGD n'est pas possible dû à sont fonctionnement interne.

Puis, l'algorithme Global arrive à des configurations ayant une erreur similaire à celles de ACC et ACC-MAB lorsque le nombre de capteurs requis est relativement faible. Toutefois, cette équivalence s'efface lorsque le nombre de capteurs augmente. On peut en conclure que l'algorithme Global n'arrive pas à tenir le rythme lorsque le nombre de capteurs à optimiser augmente.

Finalement, l'ENDP, le nombre de capteurs trouvés et le nombre d'évaluations nécessaire pour atteindre l'ENDP minimale de ACC et ACC-MAB pour les trois densités requises ne

sont pas significativement différents selon un test de Mann-Whitney U avec un niveau de confiance fixé à 99,9%.

La figure 5.5 présente les configurations médianes produites par les cinq algorithmes et les trois densités de pixels. On observe à nouveau que les algorithmes coopératifs arrivent

— à faire concorder le champ de vue des capteurs avec les bords de l'environnement,
— à conserver uniquement les capteurs utiles et
— à éviter l'empilement des champs de vue des capteurs.

Ces trois caractéristiques leur permettent d'atteindre des configurations de capteurs produisant de près la densité de pixel désirée tout en trouvant le nombre de capteurs utiles nécessaires à cette tâche.

5.2 Simulations tridimensionnelles

Cette section présente les simulations où l'environnement et la scène d'intérêt sont tridimensionnels. Un total de cinq scénarios, impliquant autant de types de robots, sont explorés. Les deux premiers scénarios se déroulent respectivement sur la planète Mars, où un robot explorateur s'est enlisé et autour de la Station spatiale internationale, où l'on veut inspecter l'état d'un panneau solaire. Ces deux premières simulations sont réalisées alors que le modèle tridimensionnel de l'environnement est initialement connu du système. Ainsi, seules les composantes de l'optimisation sont utilisées. En contrepartie, au cours des deux simulations suivantes, se déroulant respectivement dans un bâtiment endommagé, où l'on veut inspecter l'état d'une poutre de soutien et dans un musée, où l'on veut regarder une pièce de collection, le système débute dans un monde qui lui est inconnu, à l'exception d'une carte bidimensionnelle utilisée uniquement pour la navigation. Pour ces quatre simulations, les quatre algorithmes pouvant oeuvrer dans un environnement tridimensionnel sont comparés, soit ACC, ACC-MAB, Global et Vorace. Les paramètres des algorithmes utilisés dans toutes les simulations sont présentés à la table 5.3.

Les simulations décrites dans cette section font usage du système présenté au chapitre 4. Elles sont réalisées grâce au simulateur robotique Gazebo (Koenig et Howard, 2004) en combinaison avec le système d'exploitation pour robot ROS (Quigley et collab., 2009). Le modèle de l'environnement, bâti à partir des données acquises des capteurs tridimensionnels, est entreposé grâce au logiciel OctoMap (Wurm et collab., 2010; Hornung et collab., 2013) implémentant la grille d'occupation encodée dans un octree présentée à la section 2.2. Afin de faciliter les simulations et réduire la charge de calcul, la localisation des robots à l'intérieur du simulateur est sans erreur.

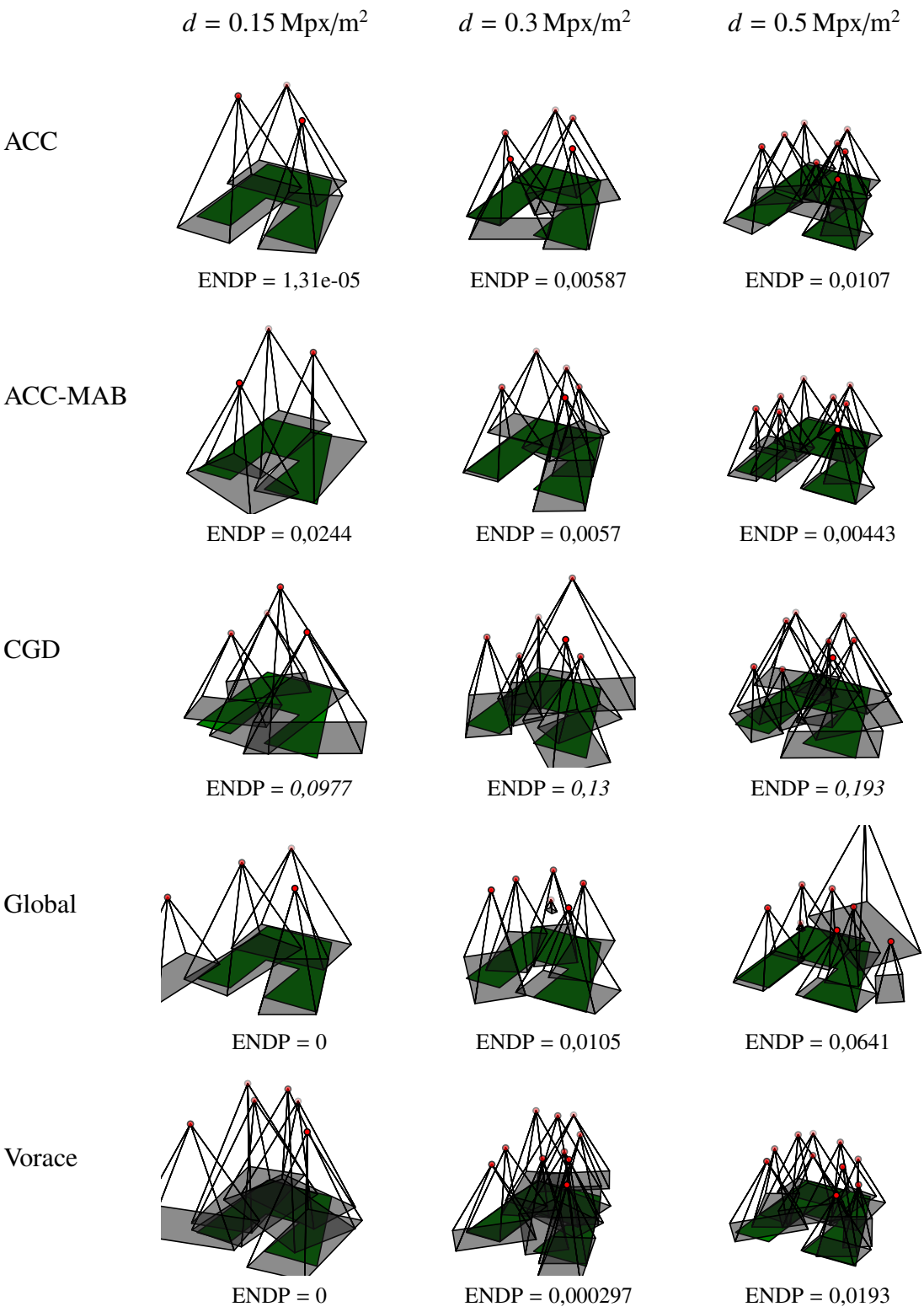


FIGURE 5.5 – Configurations finales médianes et ENDP lors de l'observation de la scène d'intérêt bidimensionnelle par un nombre de capteurs et une densité désirée variables et leur erreur associée. Les valeurs en italique notent la méthode où l'optimisation n'a pas été faite sur l'erreur normalisée de densité de pixels.

Algorithme	Global	Vorace	ACC	ACC-MAB
Nombre de capteurs initial (n)	N	1	2	2
Taille des populations (λ)	200	50	50	50
Déviation standard initiale (σ)	0,3	0,3	0,3	0,3
Délai d'amélioration (T_1)	–	10	$15n$	$15n$
Seuil d'amélioration (T_a)	–	0.001	10%	10%
Seuil d'extinction (T_e)	–	–	1/4	1/4
Taille de la fenêtre (W)	–	–	–	50
Facteur d'amortissement (D)	–	–	–	1
Facteur d'exploration (C)	–	–	–	1

TABLE 5.3 – Paramètres des algorithmes d'optimisation pour les simulations tridimensionnelles.

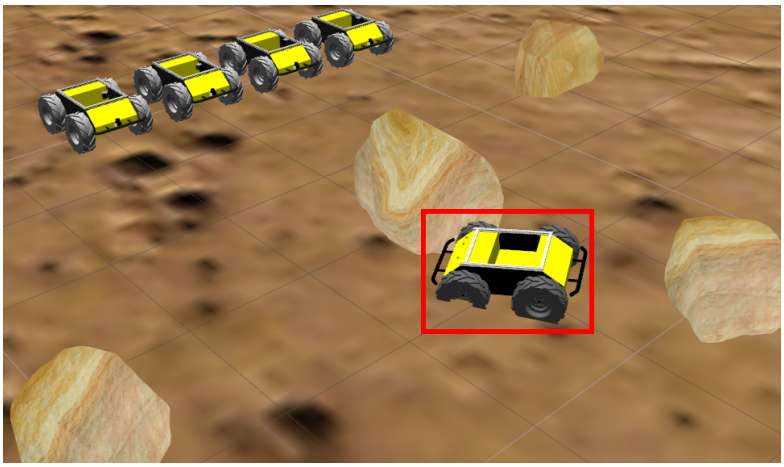


FIGURE 5.6 – Scénario d'inspection d'un astromobile enlisé.

Lors des simulations où le modèle de l'environnement est déjà connu, Gazebo n'est utilisé que lors de la phase de modélisation de l'environnement, c'est-à-dire que pour ces simulations, les données des capteurs et les mouvements des robots ne sont pas simulés. Ce préchargement de l'environnement nous permet d'effectuer beaucoup plus de répétitions des simulations, car la simulation physique du système robotique complet, comprenant plusieurs robots donc plusieurs capteurs, est une tâche très exigeante pour l'ordinateur. En effet, notre station de travail propulsée par un processeur Intel Core i7 920 avec 8 GB de mémoire RAM ne réussit à simuler le système complet qu'à environ 20% de la vitesse réelle, transformant une expérimentation requérant un peu plus de 10 minutes en simulation de presque 1 heure.

5.2.1 Rover enlisé

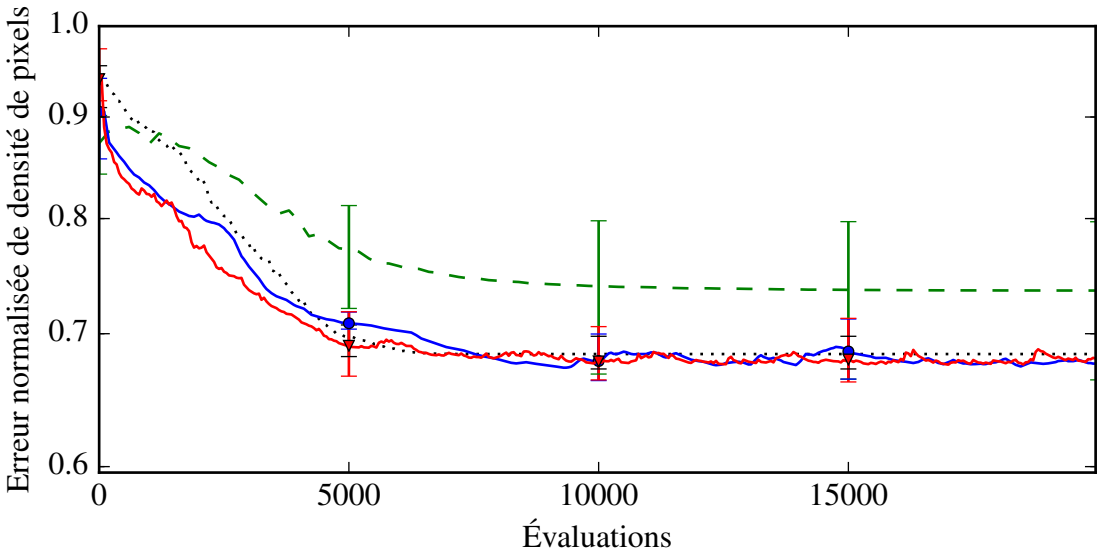
Le premier scénario, illustré à la figure 5.6, se déroule dans un environnement semblable à ce que l'ont peut retrouver sur la planète Mars. L'utilisateur veut observer toutes les faces d'un robot s'étant enlisé, dans le but d'élaborer une stratégie pour l'extirper de sa fâcheuse position. La tâche demandée aux quatre autres robots est d'observer toutes les surfaces de l'environnement comprises à l'intérieur d'un parallélépipède rectangle, soit un capteur omniscient tel que présenté à la section 2.3.1, centré sur le robot enlisé et possédant une densité de pixels constante de $d = 1\,\mathrm{Mpx/m^2}$.

Pour ce scénario, quatre robots de type Husky, commercialisés par Clearpath Robotics [2], sont utilisés. Chaque robot est équipé d'une caméra haute définition dont les paramètres sont une longueur focale $f = 1597{,}71\,\mathrm{px}$, une largeur de pixel $u = v = 1\,\mathrm{px}$ et un champ de vue de $\Theta = 62°$ par $\Phi = 48{,}6°$ produisant une caméra ayant environ 2,7 Mpx. La caméra est fixée à 32 cm du sol sur le pare-choc avant du robot. Elle a donc trois degrés de liberté, soit sa position en x et y et son orientation dans le plan horizontal. En plus de la caméra, un capteur laser bidimensionnel de type Hokuyo URG-04LX-UG01 [3] et un capteur laser tridimensionnel, similaire au Velodyne HDL-32e [4], sont aussi employés respectivement pour la navigation et la perception de l'environnement en trois dimensions.

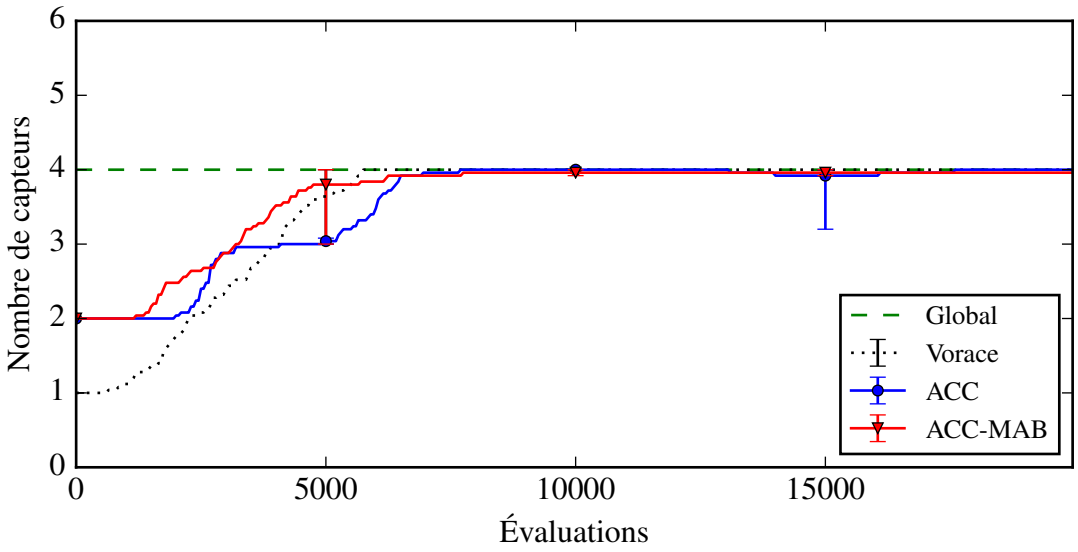
Les simulations de cette section permettent de tester le système lorsqu'un nombre limité de capteurs sont déployés pour accomplir une tâche d'observation complexe. En effet, des obstacles obstruent la visibilité aux abords du robot enlisé. Les robots doivent donc se placer stratégiquement pour observer toutes ses faces. Afin d'augmenter le nombre de répétitions des expériences pour ce scénario, le modèle de l'environnement est acquis lors de la première simulation avec l'algorithme ACC et réutilisé lors des simulations subséquentes.

Les figures 5.7 et 5.8 présentent les résultats de 25 répétitions indépendantes de cette simulation. Tout d'abord, on y remarque que les algorithmes ACC, ACC-MAB et Vorace sont sensiblement identiques. La figure 5.7(a) montre que leur rapidité à trouver une solution ayant une erreur minimale n'est pas significativement différente. Ceci est confirmé par un test de Mann-Whitney U avec niveau de confiance de 99,9% comparant la génération à laquelle est trouvée la solution minimale de chaque répétition. De plus, malgré une augmentation relativement plus rapide du nombre de capteurs pour ACC-MAB, l'ENDP minimale par nombre de capteurs est similaire pour les trois algorithmes. On remarque aussi que l'optimisation

2. http://www.clearpathrobotics.com
3. http://www.hokuyo-aut.jp
4. http://velodynelidar.com



(a) Moyenne de l'erreur normalisée de densité de pixels



(b) Moyenne du nombre de capteurs

FIGURE 5.7 – Optimisation du positionnement des robots dans le scénario du rover enlisé. Les barres d'erreur présentent les percentiles 5 et 95 des 25 répétitions.

individuelle des capteurs mène à une solution de meilleure qualité que par optimisation globale.

Ensuite, on note que que l'erreur normalisée de densité de pixels ne descend jamais sous les 66%. La cause de cette erreur constante vient de l'impossibilité de capter l'intérieur du véhicule enlisé avec la caméra située à seulement 32 cm du sol alors que le capteur tridimensionnel modélisant l'environnement est placé sur le dessus du robot, soit à 46 cm du sol. En

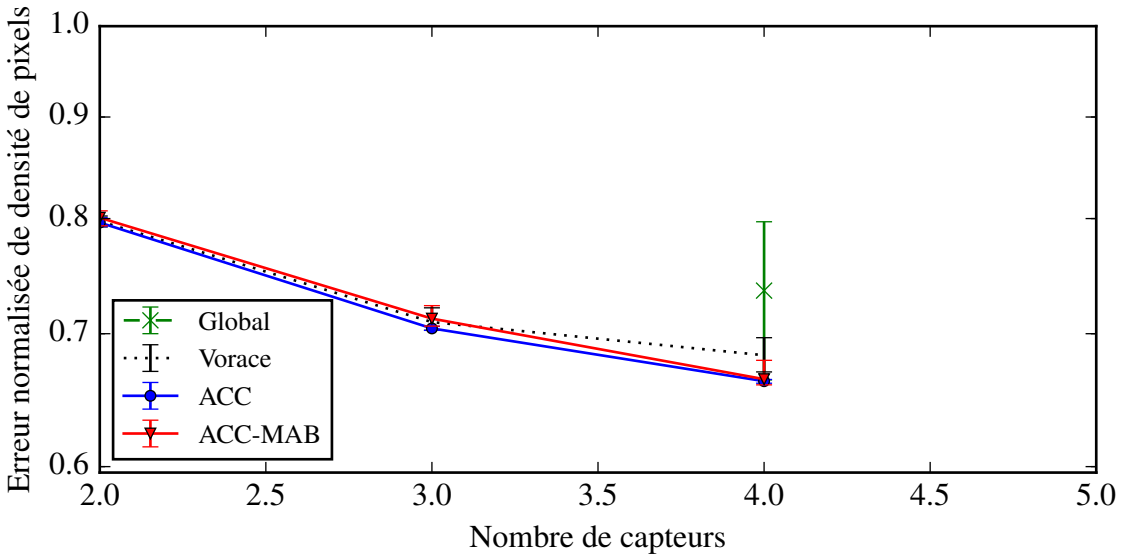


FIGURE 5.8 – Moyenne de l'erreur normalisée de densité de pixels minimale par nombre de capteurs pour le scénario du rover enlisé. Les barres d'erreur présentent les percentiles 5 et 95 des 25 répétitions.

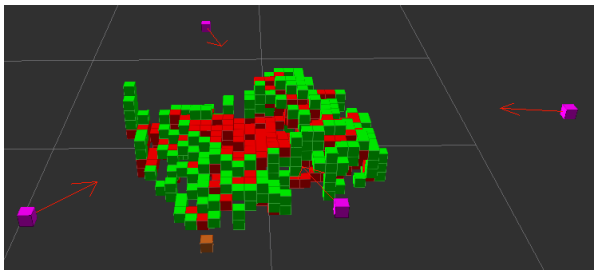
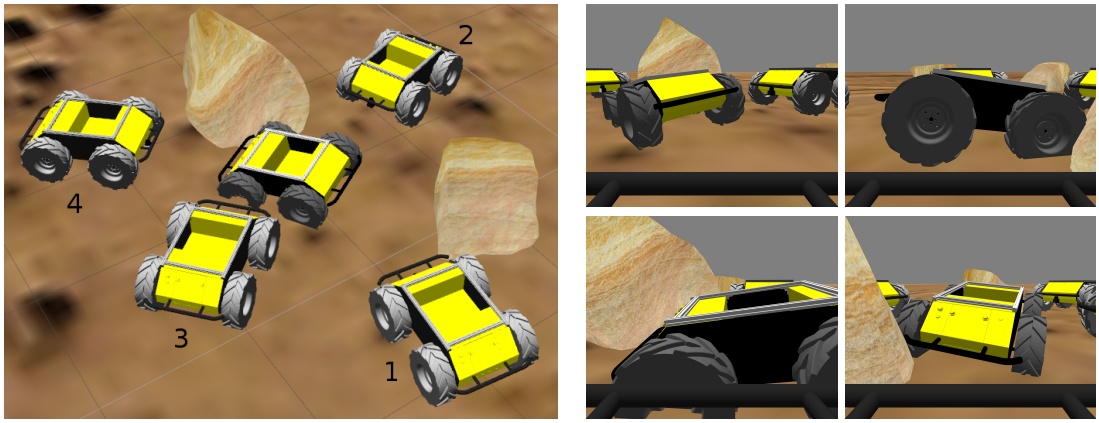


FIGURE 5.9 – Donnée d'optimisation lors du placement de capteur pour observer un rover embourbé. Les cubes vert, jaune et rouge sont respectivement observés avec la densité de pixels demandée, observés avec une densité plus faible que demandée et non observés.

effet, la figure 5.9 présente les données d'optimisation à la fin d'une expérimentation. On y observe que l'intérieur du véhicule embourbé n'est pas vu par les capteurs alors que toutes les surface extérieures le sont. Ces surfaces non observées font augmenter l'erreur normalisée de densité de pixels minimale atteignable. Finalement, la figure 5.10 présente une configuration finale obtenue par ACC ainsi que la vue de chaque capteur. On y voit bien que toutes les faces du robot enlisé sont observées.

5.2.2 Station spatiale

Le second scénario, imagé à la figure 5.11, met en scène 15 robots ayant pour but l'inspection d'un panneau solaire de la Station spatiale internationale avec une résolution beaucoup plus



(a) (b)

FIGURE 5.10 – Images acquises par les quatre Huskys de la configuration médiane trouvée par ACC lors des expérimentations du véhicule enlisé.

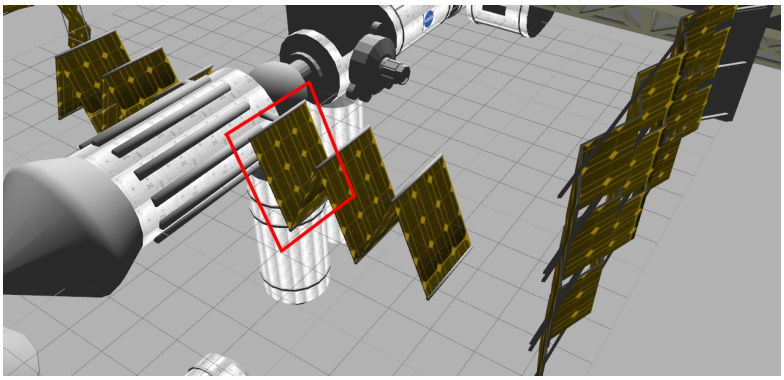


FIGURE 5.11 – Scénario d'inspection d'un panneau solaire de la Station spatiale internationale.

grande que ce que chaque robot peut produire. La tâche demandée au groupe de caméras mobiles est d'observer le panneau solaire tel qu'il serait observé par une caméra de 20 Mpx placée de manière à ce qu'il entre entièrement dans son champ de vue. Les paramètres de la caméra virtuelle sont une longueur focale de $f = 3014{,}52$ px et un champ de vue de $\Theta = 85°$ par $\Phi = 85°$.

Les robots déployés dans ces simulations peuvent se déplacer librement selon les trois axes et effectuer des rotations dans les plans horizontal et vertical. Une caméra haute définition d'environ 2,7 Mpx, de longueur focale $f = 1597{,}71$ px, de largeur de pixel $u = v = 1$ px et de champ de vue de $\Theta = 62°$ par $\Phi = 48{,}6°$ est montée sur chaque module. La caméra a la possibilité d'utiliser un zoom entre 1 et 10x ayant pour effet de réduire le champ de vue et d'augmenter la longueur focale. Par conséquent, chaque module a six degrés de liberté :

— trois pour la position x, y et z ;

— deux pour l'orientation horizontale et verticale ;
— un pour le facteur de zoom.

Ces expérimentations mettent à l'épreuve le système pour un cas d'utilisation requérant un grand nombre de capteurs avec chacun un nombre élevé de degrés de liberté. Le positionnement des caméras mobiles est relativement difficile dû à la forme en « v » du joint reliant le panneau à observer et le suivant. Pour ces simulations, l'environnement tridimensionnel a été préalablement acquis par un robot munis d'un capteur de type Velodyne manuellement déplacé dans l'environnement pour en balayer toutes les surfaces.

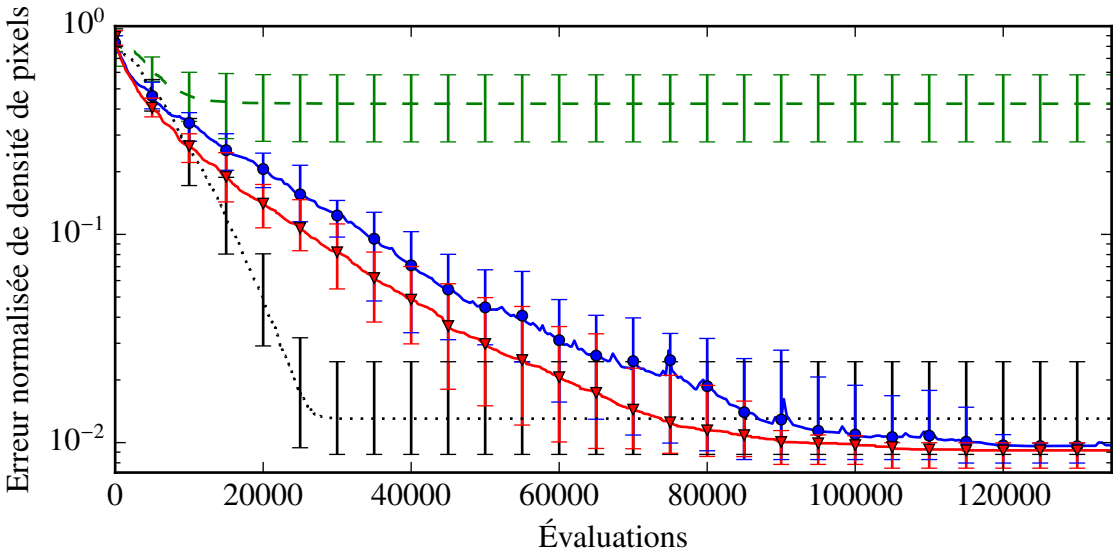
La figure 5.12 présente la moyenne des résultats de 25 répétitions des quatre algorithmes lors de l'optimisation du placement des capteurs pour l'observation du panneau solaire de la Station spatiale. On remarque tout d'abord que l'algorithme Vorace est celui présentant le départ le plus agressif. En effet, il parvient à son ENDP minimale de 1,3% beaucoup plus rapidement que les autres algorithmes. Cependant, il utilise aussi le plus grand nombre de capteurs, soit 14,2 en moyenne. Ensuite, les algorithmes coopératifs, quoique relativement plus lents atteignent une ENDP minimale inférieure à l'algorithme Vorace, respectivement 0,973% et 0,918% et tout utilisant moins de capteurs, respectivement 8,44 et 8,72. Cet avantage est clairement illustré à la figure 5.13, où la différence de performance par nombre de capteurs est mise en évidence. De plus, on note que le nombre de capteurs trouvé par les algorithmes coopératifs est très près de la valeur optimale, ce dernier étant 7,4 si la densité de pixels était uniforme dans le champ de vue des capteurs. Enfin, ACC-MAB affiche un avantage significatif quant au temps de convergence sur ACC selon un test de Mann-Whitney U avec un niveau de confiance supérieur à 99,9%, mais la supériorité de ACC quant au nombre de capteurs n'est pas significative avec ce même test.

5.2.3 Bâtiment endommagé

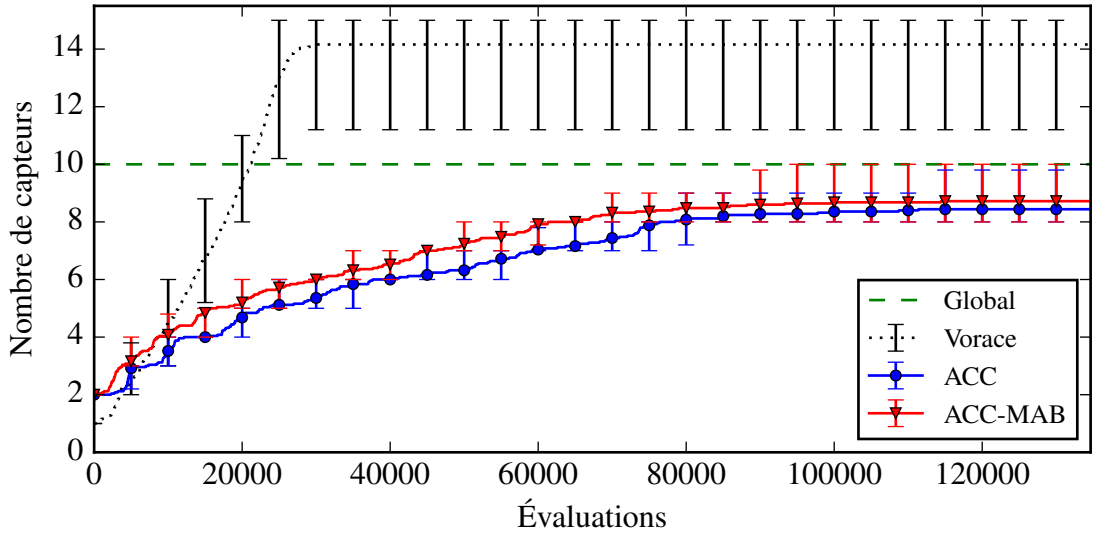
Le troisième scénario, présenté à la figure 5.14, met en scène deux types de robots pour l'inspection d'une poutre de support dans un bâtiment lourdement endommagé. Le travail demandé aux robots est d'observer sous tous ses angles la poutre qui menace de s'écrouler. Comme pour les expérimentations du rover enlisé, un parallélépipède rectangle englobe cette poutre. La densité de pixel requise pour toutes les surfaces à l'intérieur de ce volume est de $d = 0{,}3\,\text{Mpx}$.

Le premier type de robot est semblable à un Turtlebot[5] alors que le second type de robot est comparable à l'hélicoptère quadriporté (quadcoptère) Hector (Meyer et collab., 2012).

5. http://www.turtlebot.com



(a) Moyenne de l'erreur de densité de pixels



(b) Moyenne du nombre de capteurs

FIGURE 5.12 – Optimisation du positionnement des caméras autour de la station spatiale. Les barres d'erreur présentent les percentiles 5 et 95 des 25 répétitions.

Les deux robots sont munis d'un capteur de type Microsoft Kinect [6] unissant les capacités de prise d'images 2D couleurs et de capteur tridimensionnel. Les propriétés de la caméra sont une longueur focale $f = 531{,}15$ px, une largeur de pixel $u = v = 1$ px et un champ de vue de $\Theta = 62°$ par $\Phi = 48{,}6°$ produisant une caméra de 640 par 480 pixels, soit environ 0,31 Mpx. Pour le Turtlebot, la caméra est fixée à 28 cm du sol et légèrement à l'arrière du robot. Dans

6. http://www.kinect.com

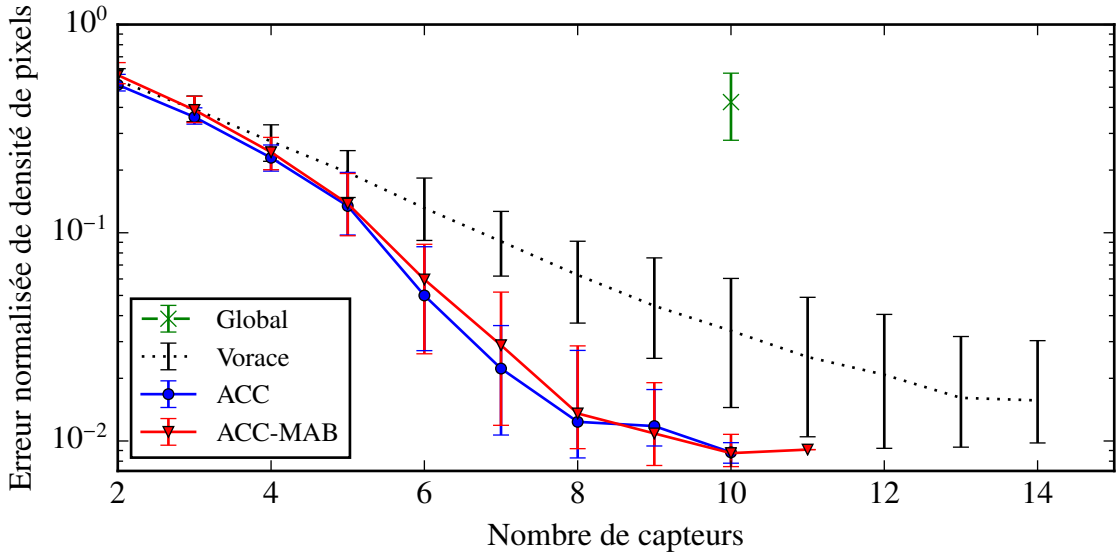


FIGURE 5.13 – Moyenne de l'erreur normalisée de densité de pixels minimale par nombre de capteurs pour le scénario de la station spatiale. Les barres d'erreur présentent les percentiles 5 et 95 des 25 répétitions.

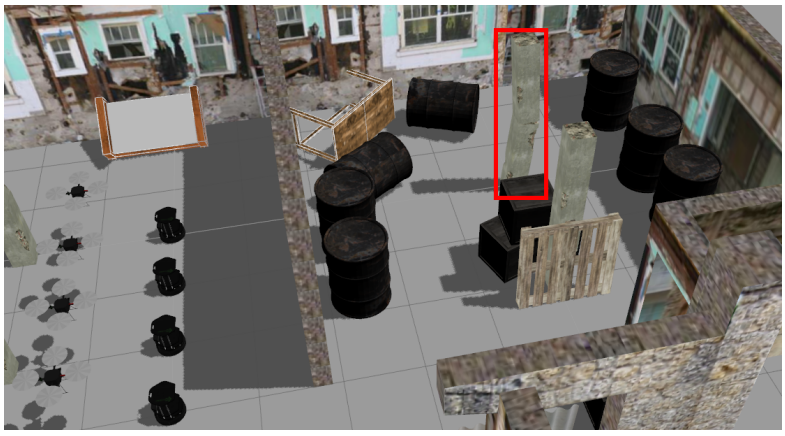


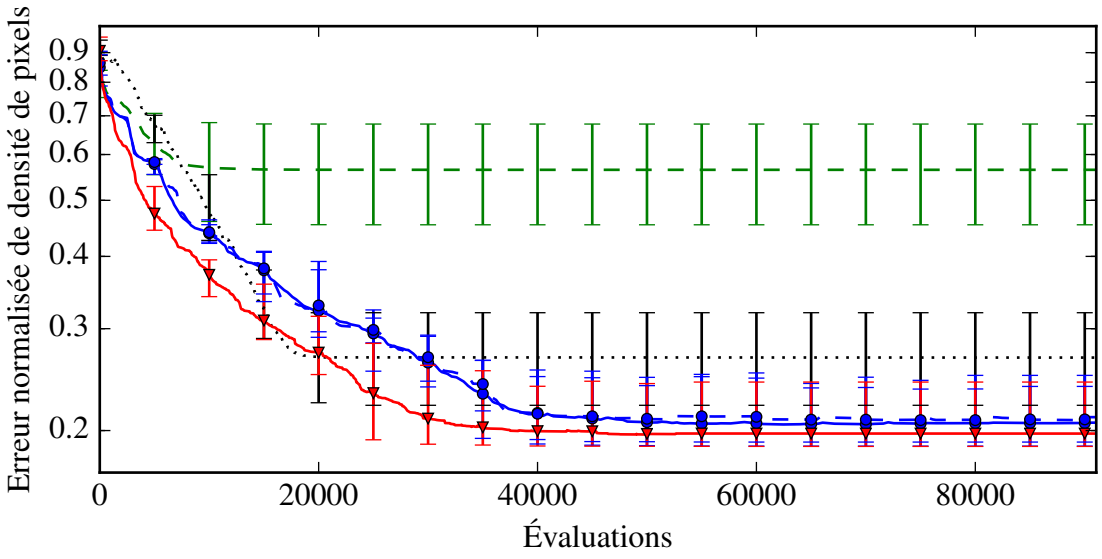
FIGURE 5.14 – Scénario d'inspection d'une poutre à l'intérieur d'un bâtiment endommagé.

le cas du Hector, la caméra est fixée à 6 cm sous l'origine de son système de référence local. La caméra du Turtlebot comporte donc trois degrés de liberté, sa position en x et y ainsi que son orientation dans le plan horizontal, alors que celle du Hector en présente un quatrième, soit sa position en z. Chaque robot est aussi doté d'un laser fixe de type Hokuyo URG-04LX-UG01 servant à la navigation et à la planification de la trajectoire. Pour cette expérience, l'algorithme Vorace est modifié de telle sorte qu'il choisit à chaque ajout de capteur le type réduisant le plus l'ENDP. Il effectue ce choix en testant chacun des types et en ajoutant uniquement celui ayant obtenu l'ENDP minimale.

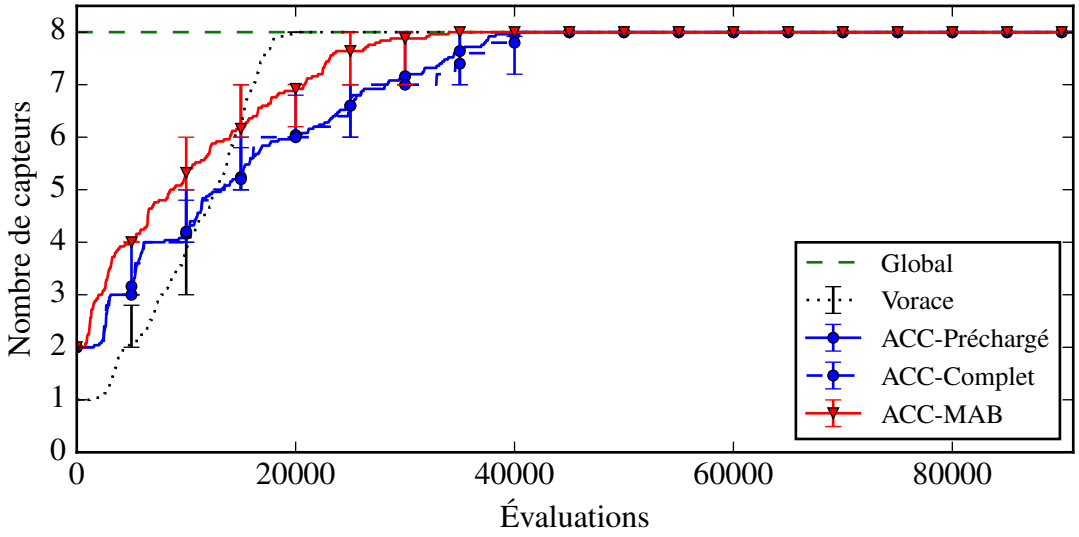
Ces simulations testent la capacité du système à déployer une flotte de robots hétérogènes dans un environnement très encombré. En effet, le haut de la poutre ne peut être observé convenablement que par les robots aériens. Le bas de la poutre quant à lui doit l'être de points bien précis dû à l'encombrement au sol de l'environnement. Pour ces simulations, cinq répétitions sont effectuées avec le système complet, soit un environnement initialement inconnu et une alternance de l'optimisation et des déplacements, pour l'algorithme ACC. Les quatre algorithmes sont ensuite exécutés à 25 répétitions en préchargeant l'environnement construit lors de la première simulation complète avec ACC.

Les figures 5.15 et 5.16 présentent l'ENDP en fonction du temps et du nombre de capteurs pour les quatre algorithmes et les deux cas d'utilisation de ACC. Tout d'abord, on note la similitude entre les courbes associées au cas où ACC débute dans avec un modèle de l'environnement vide (ACC-Complet) et préchargé (ACC-Préchargé). En effet, un test de Mann-Whitney U avec un niveau de confiance de 99,9% révèle que les deux cas d'utilisation sont semblables à la fois pour le nombre d'évaluations nécessaires pour atteindre l'ENDP minimale, la valeur de cette ENDP et le nombre de capteurs utilisés. Pour ce scénario, le système proposé est donc indépendant de l'état du modèle de l'environnement au début de son utilisation.

Ensuite, on remarque la même tendance qu'aux simulations précédentes, c'est-à-dire que la réduction de l'erreur pour les algorithmes coopératifs se fait plus lentement que pour l'algorithme Vorace. Cependant, pour ces simulations, l'algorithme Vorace ne peut utiliser un nombre supérieur de capteurs par limitation de la flotte disponible. En conséquence, la différence d'ENDP est supérieure à ce qu'elle était aux simulations précédentes. L'ENDP atteinte par les algorithmes coopératifs suggère qu'ils sont les seuls à être en mesure de placer convenablement les robots aériens pour observer le haut de la colonne, tâche irréalisable par un robot terrestre. Encore une fois, ACC-MAB a un léger avantage sur ACC au niveau du temps



(a) Moyenne de l'erreur de densité de pixels



(b) Moyenne du nombre de capteurs

FIGURE 5.15 – Optimisation du positionnement des caméras dans le bâtiment endommagé. Les barres d'erreur présentent les percentiles 5 et 95 des 25 répétitions.

de convergence vers la solution ayant une erreur minimale. Cet avantage est significatif selon un test de Mann-Whitney U avec un niveau de confiance supérieur à 99,9%.

Enfin, comme aux simulations du rover enlisé, l'utilisation du capteur omniscient augmente l'erreur minimale atteignable, car il inclut des surfaces qui ne sont pas visibles normalement par les capteurs. La figure 5.17 expose les données d'optimisation menant à cette erreur constante. En effet, on constate aux figures 5.17(a) et (b) que la très grande majorité des

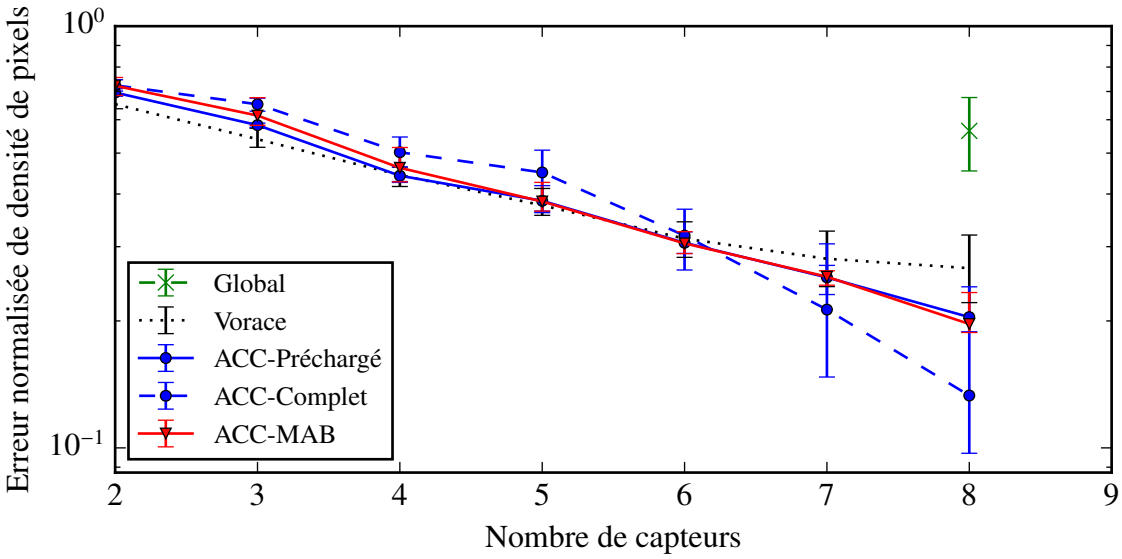


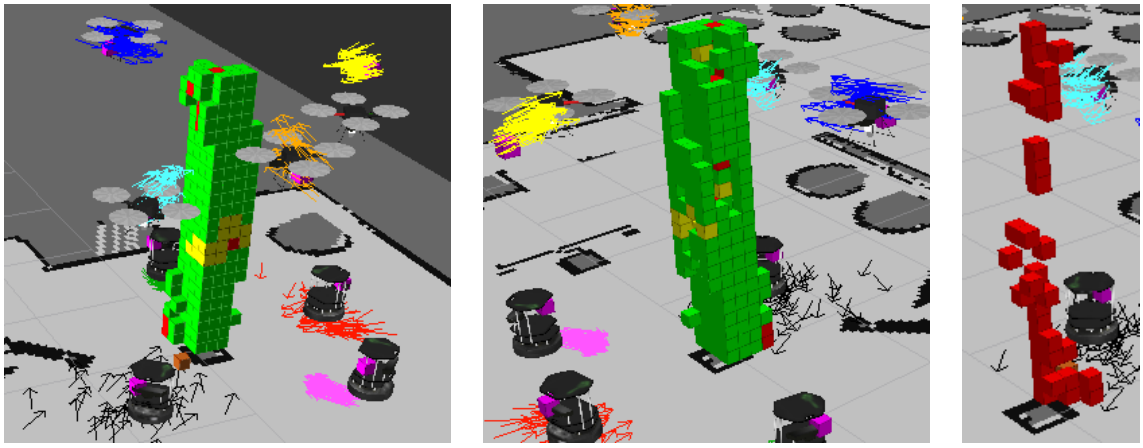
FIGURE 5.16 – Moyenne de l'erreur normalisée de densité de pixels minimale par nombre de capteurs pour le scénario du bâtiment endommagé. Les barres d'erreur présentent les percentiles 5 et 95 des 25 répétitions.

voxels sont observés avec la densité désirée. Toutefois, en retirant ces derniers, comme présenté à la figure 5.17(c), une couche cachée de voxels non observés apparaît. Ces voxels sont le produit du bruit de mesure des capteurs tridimensionnels, c'est-à-dire que lorsque la distance mesurée est plus grande que la distance réelle un voxel occupé est ajouté derrière la parois de la poutre. Ces voxels constituent l'erreur minimale constante observée aux graphiques précédents.

5.2.4 Musée

Le quatrième scénario tridimensionnel, présenté à la figure 5.18, se déroule dans un musée. Cette fois, comme pour un visite virtuelle, il est demandé aux robots d'observer une peinture désignée par l'utilisateur. On utilise une caméra virtuelle dont les paramètres font en sorte que seule la peinture désignée est visible, soit une longueur focale $f = 1000\,\text{px}$ et un champ de vue de $\Theta = 35°$ par $\Phi = 30°$, pour une caméra ayant environ 0,34 Mpx.

Ce scénario met en scène les deux mêmes types de robots qu'aux expérimentations du bâtiment endommagé de la section précédente. La position de la peinture et les paramètres de la caméra virtuelle rendent impossible aux robots terrestres d'observer la cible avec la densité de pixels désirée. La difficulté de ce scénario réside donc dans le choix des capteurs à utiliser. En effet, l'algorithme d'optimisation doit découvrir une solution uniquement composée de robots aériens. À nouveau pour ces simulations, cinq répétitions sont effectuées



(a) (b) (c)

FIGURE 5.17 – Solution finale d'un expérimentation dans le bâtiment endommagé. (a) et (b) présentent les données d'optimisation où les voxels verts, jaunes et rouges représentent respectivement les surfaces observées avec au moins la densité désirée, avec un densité plus faible que désirée et non observés, et (c) voxels occupé à l'intérieur de la colonne, ces voxels sont cachés par des voxels parfaitement observés des figures (a) et (b).

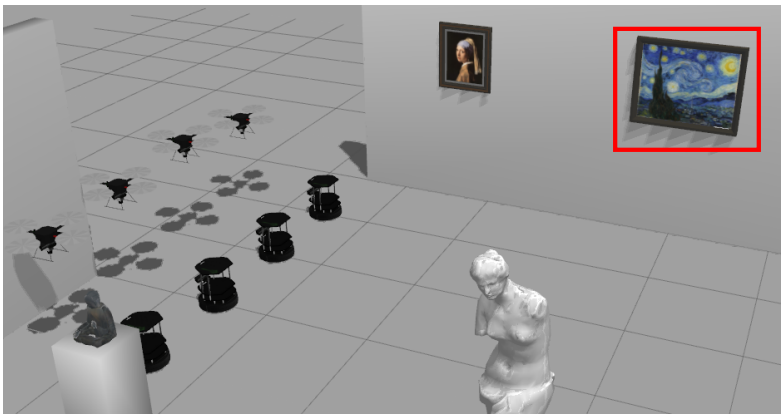
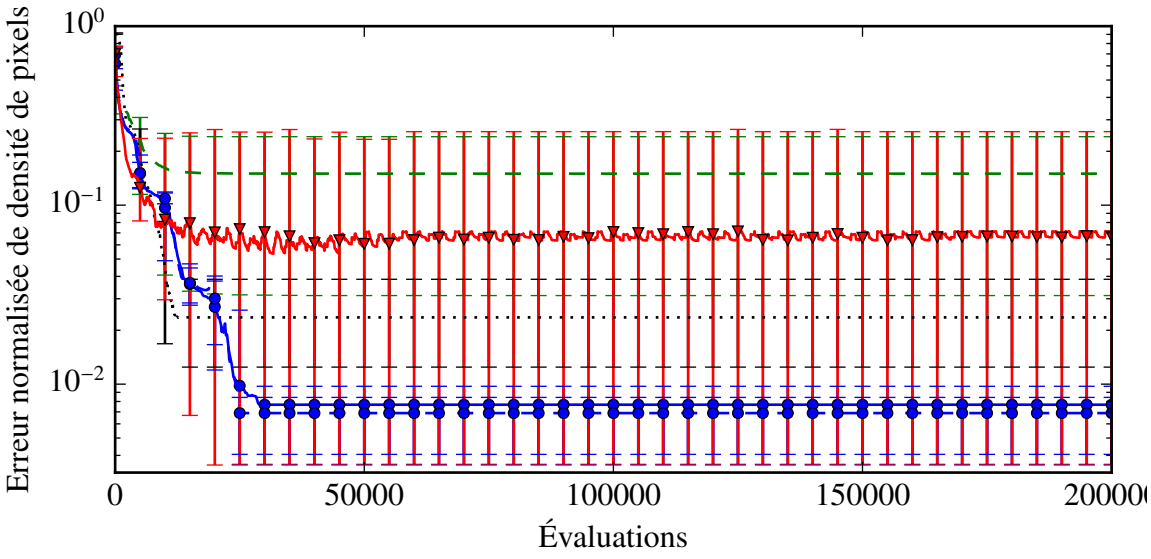


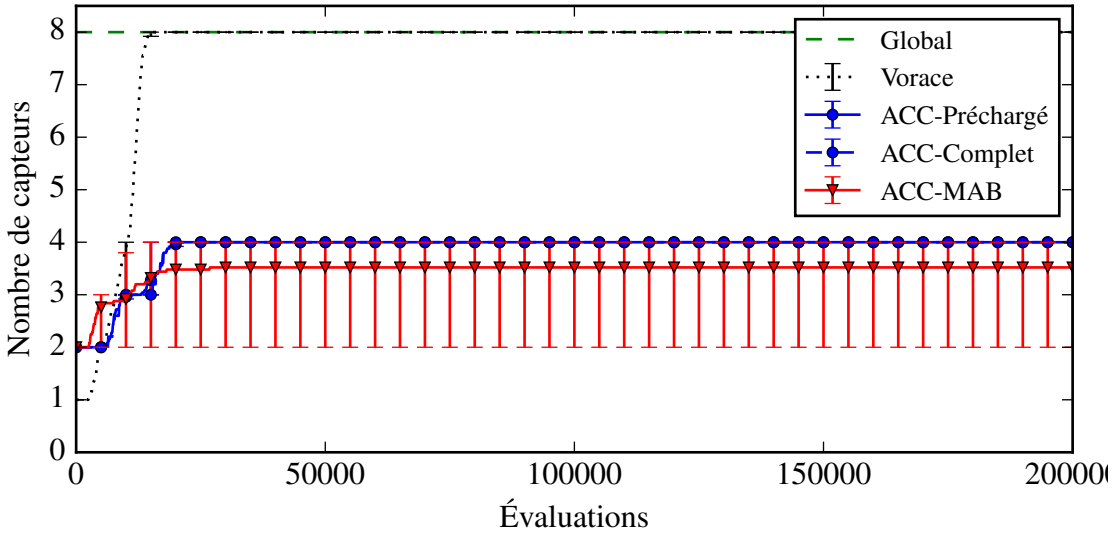
FIGURE 5.18 – Scénario de visite virtuelle d'un musée.

avec le système complet et un environnement initialement inconnu pour l'algorithme ACC. Puis, 25 répétitions sont réalisées en préchargeant l'environnement découvert par la première simulation complète avec l'algorithme ACC et ce, pour les quatre algorithmes.

Les figures 5.19 et 5.20 présentent la moyenne de l'erreur en fonction du temps et du nombre de capteurs pour les quatre algorithmes et les deux cas d'utilisation de ACC. À nouveau l'utilisation du système complet n'affecte pas les performances observées tel que démontré par la ressemblance des courbes associées aux deux cas d'utilisation de ACC. En effet, les algorithmes ACC-Complet et ACC-Préchargé parviennent à tout coup à trouver une configuration des quatre robots aériens permettant l'observation avec une erreur minimale de densité



(a) Moyenne de l'erreur de densité de pixels



(b) Moyenne du nombre de capteurs

FIGURE 5.19 – Optimisation du positionnement des caméras dans le musée. Les barres d'erreur présentent les percentiles 5 et 95 des 25 répétitions.

de pixels de la peinture désignée pour les deux cas d'utilisation. Le test de Mann-Whitney U avec un niveau de confiance de 99,9% révèle derechef que les deux cas d'utilisation sont semblables tant pour le nombre d'évaluations nécessaires pour atteindre l'ENDP minimale, la valeur de cette ENDP que le nombre de capteurs utilisés. Ceci montre à nouveau l'indépendance du système face à l'état initial du modèle de l'environnement.

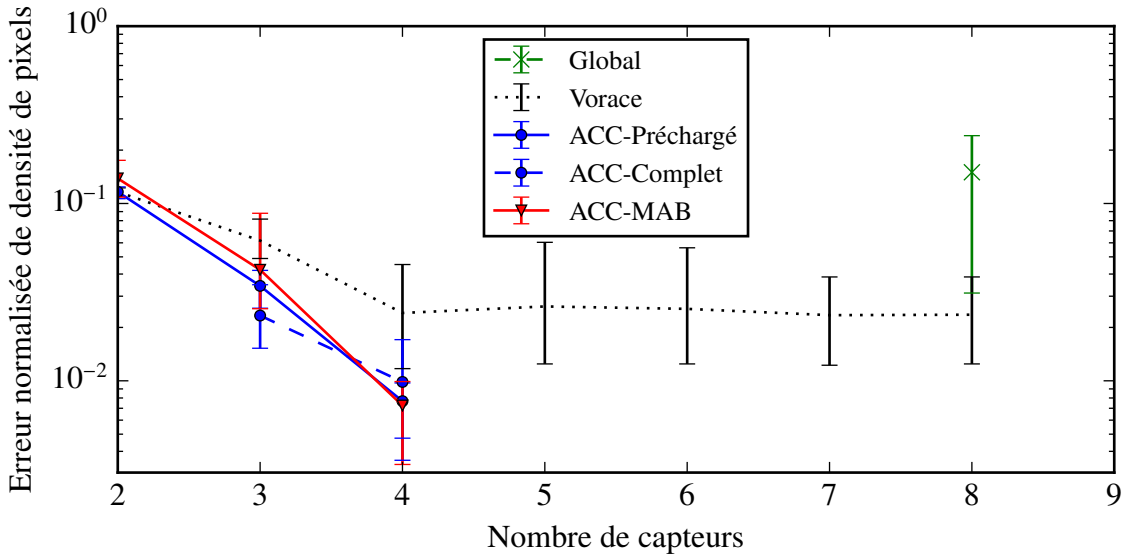
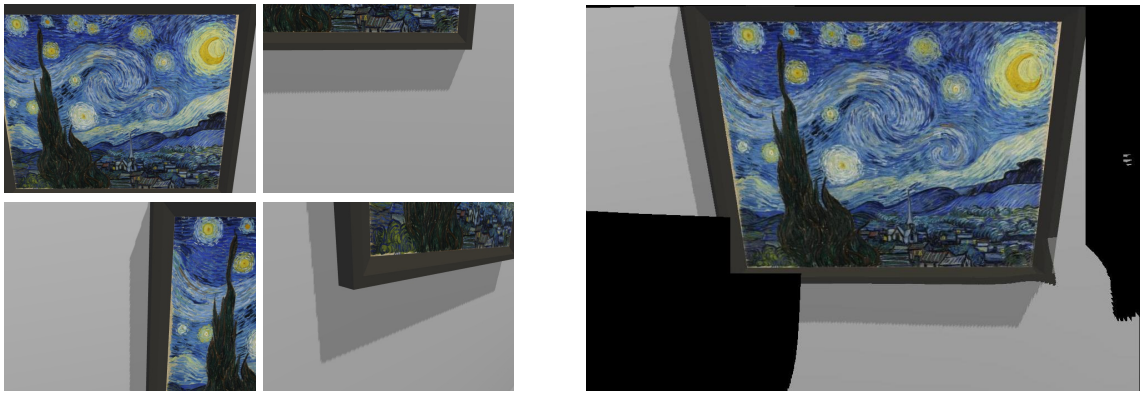


FIGURE 5.20 – Moyenne de l'erreur normalisée de densité de pixels minimale par nombre de capteurs pour le scénario du musée. Les barres d'erreur présentent les percentiles 5 et 95 des 25 répétitions.

Pour ce scénario, on note que l'erreur moyenne de ACC-MAB est largement supérieure à celle de ACC et Vorace tandis que le nombre de capteurs utilisé est inférieur. En analysant plus en profondeur ces données, nous constatons que lors de huit simulations ACC-MAB ne réussit pas à ajouter un troisième ou un quatrième capteur dont la contribution dépasse la contribution minimale T_c. L'algorithme reste donc coincé avec uniquement deux ou trois capteurs et ne peut continuer l'optimisation comme les autres algorithmes. Pour éviter cette situation, il serait possible de diminuer la contribution minimale avant extinction, ce qui aurait pour conséquence de laisser un capteur moins performant dans la flotte en espérant qu'il s'améliore avec le temps. Il serait de plus possible d'augmenter la densité de pixels demandée de manière à forcer les capteurs à observer individuellement une zone moins grande, mais augmenter la contribution possible des nouveaux capteurs. Lors des 17 autres simulations, ACC-MAB trouve des configurations de quatre capteurs similaires à celle de ACC tel qu'indiqué par les barres d'erreurs de la figure 5.19 et l'ENDP minimale par nombre de capteurs de la figure 5.20.

La figure 5.21 présente la reconstruction de la peinture observée par les quatre robots pour le résultat médian de l'algorithme ACC. La reconstruction est effectuée par la méthode d'assemblage d'images. Seules les trois premières images sont utilisées, car le recouvrement entre l'image acquise par le quatrième robot et les autres n'est pas assez grand pour trouver un nombre satisfaisant de points associés pour estimer l'homographie. Ce problème pourrait



(a) (b)

FIGURE 5.21 – Vue acquises par les quatre Hector (a) et assemblage de ces images (b) pour le résultat médian des expérimentations dans le musée.

être abordé dans la fonction objectif en ajoutant un critère encourageant un certain recouvrement entre les différents capteurs. Nous observons tout de même que le résultat de la reconstruction est plus que satisfaisant.

5.3 Conclusion

Les simulations présentées démontrent l'efficacité du système à placer une flotte de capteurs hétérogènes tant dans un environnement bidimensionnel que tridimensionnel. Dans un premier temps, nous avons démontré que la fonction objectif et l'algorithme d'optimisation proposés dans cette thèse rivalisent avantageusement avec l'état de l'art en matière de placement de capteurs pour observer un environnement bidimensionnel lorsque le nombre de capteurs est connu. Nous avons aussi montré que l'algorithme proposé est en mesure de trouver des solutions de qualité supérieure même lorsque le nombre de capteurs n'est pas spécifié. En effet, l'algorithme est capable de trouver le nombre de capteurs nécessaire à l'obtention d'une densité de pixels désirée sur l'ensemble d'une zone d'intérêt.

Ensuite, nous avons vérifié que nos résultats se transposent bien au placement de capteurs dans des environnements tridimensionnels. Cinq scénarios de simulations en 3D ont été testés. Le premier demandait le placement de quatre rovers pour l'inspection de toutes les faces d'un véhicule enlisé. Ce scénario démontrait l'avantage d'optimiser de manière individuelle les capteurs plutôt que d'utiliser une méthode d'optimisation commune à tous les capteurs. Le second scénario impliquait une flotte de robots ayant chacun six degrés de liberté pour observer l'état d'un panneau solaire. Ce scénario testait l'habileté du système à coordonner une flotte de capteurs lorsque la densité de pixel demandée est largement supérieure celle que peut

produire chaque capteur. Il établissait que les algorithmes coopératifs arrivaient, comme dans le cas des environnements bidimensionnels à trouver des solutions de grande qualité avec un nombre plus faible de capteurs qu'un algorithme vorace ou global. Le troisième scénario, se déroulant dans un bâtiment endommagé, démontrait de la capacité du système à trouver une configuration de capteurs pour un environnement très encombré et un groupe de capteurs hétérogènes. Encore une fois, les algorithmes coopératifs proposés y arrivaient mieux que tout autre algorithme. Finalement, le quatrième scénario, demandant aux robots d'observer avec précision une toile dans un musée, montrait la capacité des algorithmes à choisir uniquement les capteurs utiles parmi une flotte de robots hétérogènes. Nous avons montré que le système, de par son exploitation algorithmes d'optimisation coopératifs, parvient à une solution quasi optimale même avec ces réglages.

En résumé, nous avons montré le bon fonctionnement du système proposé dans plusieurs environnements bi- et tridimensionnels de niveaux de difficulté différents et que ce fonctionnement n'est pas affecté lorsque l'environnement est préalablement inconnu. En effet, les résultats suggèrent une indifférence du système proposé quant à l'état initial du modèle de l'environnement. Nous avons aussi vérifié que la sortie du système, soit les images prises par les capteurs, peut constituer l'entrée d'un système d'assemblage de l'information comme l'assemblage d'images et que la fonction objectif peut être ajustée pour obtenir des configurations de capteurs aux caractéristiques bien précises.

Chapitre 6

Expérimentations réelles

Ce chapitre décrit les expérimentations réelles effectuées avec le système complet. Deux scénarios intérieurs sont utilisés pour démonter l'applicabilité du système proposé dans un environnement et des conditions réels. Pour ces deux scénarios se déroulant sur un plancher et comme pour les simulations précédentes, une carte bidimensionnelle de l'environnement est préalablement construite par SLAM. Toutefois, à l'instar des simulations, la carte est aussi utilisée pour la localisation des robots plutôt qu'uniquement pour la planification des trajectoires. Cette configuration permet de limiter l'utilisation de l'unité de calculs des robots et ainsi économiser de l'énergie. Le reste du système est identique à celui utilisé aux simulations tridimensionnelles.

Quatre robots de type iRobot Create sont utilisés pour ces expérimentations. Chacun est équipé d'un capteur laser fixe de type Hokuyo URG-04LX-UG01 pour la navigation et à la planification de la trajectoire, d'un capteur tridimensionnel Microsoft Kinect pour capter les images de l'environnement et en construire un modèle, et d'un gyroscope aidant à la navigation. Les paramètres de la caméra de la Kinect sont une longueur focale de 531 px et un champ de vue de $\Theta = 62°$ par $\Phi = 48{,}6°$ pour une caméra de 0,3 Mpx. L'unité de calcul installée sur les robots est un Asus Eee-PC. Elle sert aussi de point d'accès au réseau Wi-Fi du bâtiment dans lequel se déroulent les expériences. L'optimisation et la représentation tridimensionnelle de l'environnement sont déployées sur une station de travail Intel Core i7 920 avec 8 GB de mémoire RAM. Toutes ces unités sont reliées entre elles par le module ROS Multimaster.

Maintenant que l'efficacité des algorithmes coopératifs a été démontrée en simulation, nous nous concentrons sur l'applicabilité du système proposé à résoudre le problème de placement de capteurs dans des scénarios réels. Pour cette raison, dans ce chapitre, seul l'algorithme

FIGURE 6.1 – Scénario d'inspection d'une mosaïque avec la caméra virtuelle placée et orientée vers la mosaïque telle qu'indiquée par la flèche rouge.

d'optimisation ACC est employé. Nous vérifions donc la capacité du système entier à positionner des capteurs dans un environnement tridimensionnel initialement inconnu lorsque les robots sont munis de vrais capteurs ayant un bruit non gaussien et que leur localisation n'est pas parfaite.

6.1 Mosaïque

Le premier scénario d'expérimentation, tel qu'illustré à la figure 6.1, se déroule dans un grand corridor sous-terrain sur le campus de l'Université Laval où un utilisateur veut inspecter l'état d'un mur. La vue désirée est commandée par une caméra virtuelle placée à hauteur d'homme de telle sorte qu'elle observe entièrement la mosaïque « Biodivertcité ». Les paramètres de la caméra virtuelle sont une longueur focale $f = 666\,\text{px}$, et un champ de vue de $\Theta = 120°$ par $\Phi = 45°$ résultant en une caméra de 1,27 Mpx.

La figure 6.2 présente la décroissance moyenne de l'erreur normalisée de densité de pixels au cours des expérimentations. On note que l'erreur minimale moyenne est d'environ 0,31 alors que la surface observée par la caméra virtuelle est approximativement 11,6 m^2 (7 m par 1,66 m). Donc en moyenne la densité de pixels désirée est de 0,11 Mpx/m^2. Comme illustré à la figure 6.3, les robots les plus près de la mosaïque en observent la partie du bas et ceux les plus éloignée en observent la partie du haut. Ces deux groupes de robots se trouvent respectivement à environ 1,55 m et 2,44 m de la mosaïque. La densité de pixels résultante pour les capteurs observant le bas et le haut de la mosaïque est donc respectivement estimée à 0,12 Mpx/m^2 et 0,04 Mpx/m^2. On constate donc que la densité d'échantillonage des robots observant le bas de la mosaïque est optimale alors que celle des robots en observant le haut est

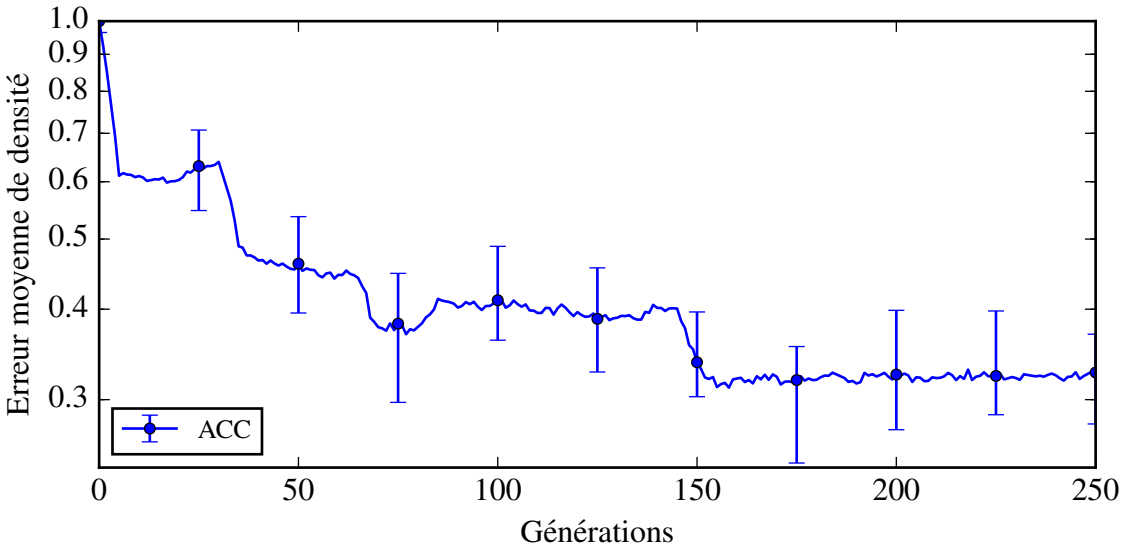


FIGURE 6.2 – Moyenne de l'erreur normalisée de densité de pixels au cours de l'optimisation du placement des capteurs pour l'expérimentation de la mosaïque. Les barres d'erreur présentent les percentiles 5 et 95 des 5 répétitions.

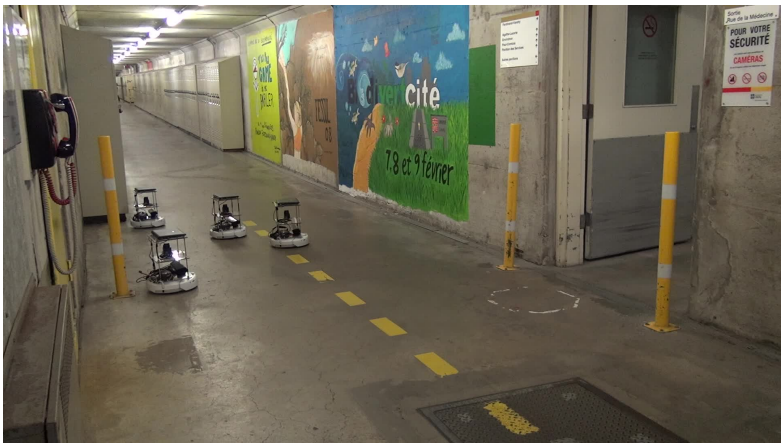


FIGURE 6.3 – Configuration finale des robots pour l'observation de la mosaïque.

plus élevée en raison de la contrainte d'élévation des robots. L'assemblage d'images associée à cette configuration est présentée à la figure 6.4. On note que le coin supérieur gauche de la mosaïque n'est pas couvert, cette situation est principalement due au manque de capteurs. En effet, l'algorithme d'optimisation semble préférer couvrir avec la bonne résolution le bas de la mosaïque plutôt que de faire diminuer de manière moins significative l'erreur en couvrant avec une résolution trop faible le haut de la mosaïque.

FIGURE 6.4 – Assemblage des images prises par les quatre robots placés à leur position finale lors de l'inspection de la mosaïque.

FIGURE 6.5 – Scénario d'inspection d'une porte de voiture avec la caméra virtuelle placée et orientée vers la porte de voiture telle qu'indiquée par la flèche rouge.

6.2 Porte de voiture

Le second scénario d'expérimentation avec le système réel se déroule dans une pièce fermée et légèrement encombrée, tel qu'illustré à la figure 6.5. Dans ce scénario, les robots doivent observer l'intérieur d'une porte de voiture pour en faire l'inspection tout en évitant d'avoir la vue obstruée par les obstacles. La vue désirée est commandée par une caméra virtuelle placée où se situe la boîte la plus près de la porte de voiture. Les paramètres de la caméra virtuelle sont une longueur focale $f = 666\,\text{px}$, et un champ de vue de $\Theta = 60°$ par $\Phi = 45°$ résultant en une caméra de 0,48 Mpx.

La figure 6.6 présente l'évolution moyenne de l'erreur normalisée de densité de pixels au cours de l'optimisation. On y note bien le même type de décroissance de l'erreur qu'aux simulations du chapitre précédent. L'erreur normalisée de densité de pixels minimale moyenne

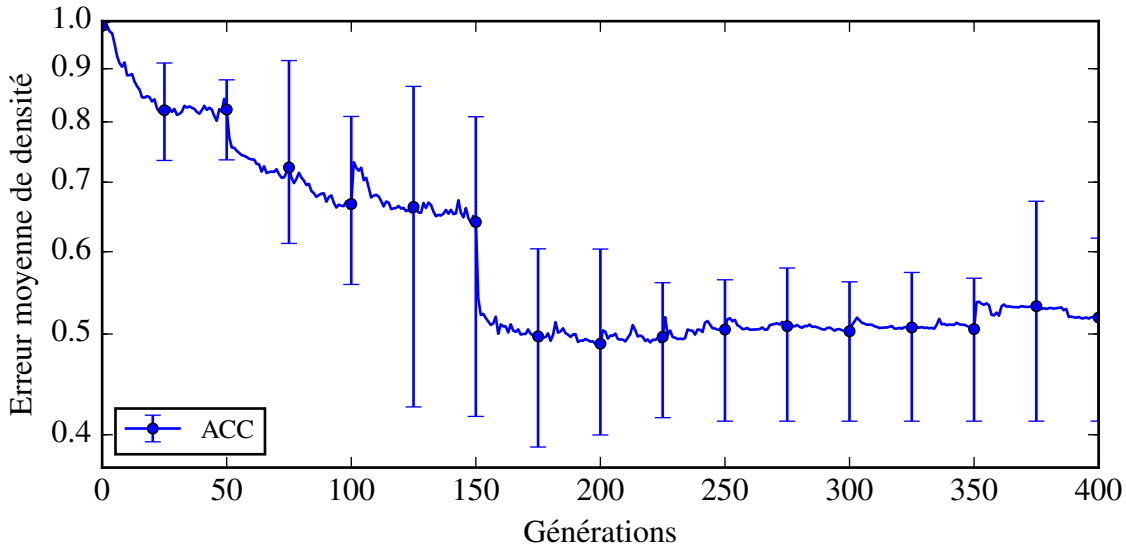
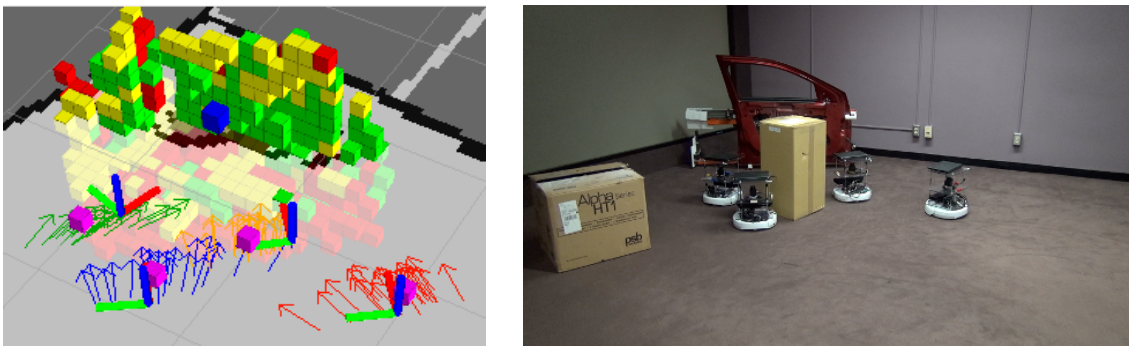


FIGURE 6.6 – Moyenne de l'erreur normalisée de densité de pixels au cours de l'optimisation du placement des capteurs pour l'expérimentation de la porte de voiture. Les barres d'erreur présentent les percentiles 5 et 95 des 5 répétitions.



(a) (b)

FIGURE 6.7 – Données d'optimisation, où les voxels verts, jaunes et rouges représentent respectivement les surfaces observées avec au moins la densité désirée, avec un densité plus faible que désirée et non observées (a) et configurations finale (b) pour le scénario de la porte de voiture.

atteinte lors de ces simulations est de 0,52. Comme illustré à la figure 6.7(a), cette erreur est principalement due à la présence d'une partie du plancher dans le champ de vue de la caméra virtuelle. Cette partie ne peut être observée par les capteurs réels de par leur angle de vue, soit une élévation d'environ 15°. On voit bien que la majorité des voxels de la porte sont observés avec la densité de pixels demandée alors qu'une bande correspondant au plancher n'est pas observée.

(a) (b)

FIGURE 6.8 – Assemblage d'images des robots de gauche (a) et de droite (b) pour les expérimentations de la porte de voiture.

La figure 6.8 présente l'assemblage d'images pour les groupes de robots à gauche et droite de la boîte. On constate que la porte est bel et bien entièrement observée par l'union des quatre vues. En effet, les robots se sont placés le plus près possible de la porte pour maximiser la densité de pixel tout en la gardant entièrement dans leur champ de vue. On remarque aussi que les robots sont positionnés de telle manière qu'aucun obstacle n'est dans leur champ de vue et qu'ils ne s'obstruent pas non plus la vue entre eux. Un vidéo disponible à l'adresse http://vision.gel.ulaval.ca/~fmdrainville/msensor_opt.mp4 présente le déroulement d'une des expérimentations.

6.3 Conclusion

Cette section a présenté les expérimentations effectuées avec le système complet dans deux scénarios réels. Nous avons montré que malgré les difficultés inhérentes aux systèmes robotisés réels, soit un bruit de mesure non gaussien et une localisation imparfaite, le système parvient à trouver et réaliser une configuration de capteurs fort satisfaisante pour la tâche demandée. En effet, lors du premier scénario, les robots avaient à observer une partie d'un mur où se trouve une mosaïque et en dépit de la contrainte d'angle de vue, ces derniers ont réussi à observer convenablement la cible. Ensuite, lors du second scénario, pour l'inspection de la porte de voiture, chaque robot a réussi à se positionner non seulement en évitant

les obstacles, mais aussi en évitant d'avoir sa vue bloquée par les autres robots. On constate donc le fonctionnement adéquat du système même en situation réelle attestant la robustesse des algorithmes développés et des composantes utilisées.

Chapitre 7

Conclusion

Dans cette thèse, nous avons présenté une solution systémique au problème de placement de capteurs dans un environnement tridimensionnel initialement inconnu. Tout d'abord, au chapitre 2, nous avons mis de l'avant une nouvelle méthode pour évaluer la qualité d'une configuration de capteurs. Cette méthode simule les pixels des capteurs dans l'environnement et combine les résultats en une fonction objectif agnostique du nombre de capteurs et de leur type. Ensuite, au chapitre 3, nous avons introduit un algorithme d'optimisation capable de décomposer automatiquement un problème en un nombre adéquat de sous-problèmes pour lesquels il est plus facile de trouver une solution. Une fois combinées, ces solutions permettent de résoudre le problème original. Nous y avons aussi abordé un algorithme d'optimisation global sans dérivée très efficace pour l'optimisation de problèmes à nombres réels. Ces deux algorithmes ont été intégrés pour former l'algorithme d'optimisation employé pour l'optimisation des capteurs. Par la suite, au chapitre 4, nous avons combiné la fonction objectif et l'algorithme d'optimisation à un système complet de placement de capteur. Ce système comprend tous les mécanismes nécessaires à la localisation, la navigation, l'optimisation des points de vue et la reconstruction de la vue virtuelle. Enfin, au chapitres 5 et 6, nous avons montré en simulation et en expérimentations réelles le bon fonctionnement du système. En effet, dans un premier temps, nous avons comparé avantageusement le système de placement de capteurs proposé à l'état de l'art dans le domaine des réseaux robotiques pour des robots aériens observant un environnement bidimensionnel. Dans un second temps, nous avons vérifié que nos résultats se transposent bien aux mondes tridimensionnels. Il a été montré que le système parvient à des configurations de capteurs plus que satisfaisantes dans de multiples configurations d'environnement et de zone d'intérêt. Dans un troisième temps, les expérimentations réelles ont confirmé l'efficacité du système dans des conditions de localisation imparfaite et de bruit de mesure des capteurs.

Les différentes contributions faites durant la thèse sont présentées à la section 7.1 et les travaux futurs à la section 7.2. Pour une liste des publications, voir l'annexe B.

7.1 Contributions

Plusieurs contributions significatives ont été faites à différents domaines de recherche découlant de cette thèse. Cette section en dresse une liste exhaustive en les classant dans quatre catégories : la qualité d'une configuration de capteurs, l'optimisation coopérative, le système déployé et l'outil logiciel d'algorithmes évolutionnaires.

7.1.1 Qualité d'une configuration de capteurs

Une première contribution majeure de cette thèse, présentée au chapitre 2, est la proposition d'une mesure de qualité de la configuration d'un groupe de capteurs pour l'acquisition d'information dans une scène d'intérêt. L'originalité de cette contribution réside dans la combinaison de deux caractéristiques majeures des stratégies de calcul de la meilleure prochaine vue et des réseaux robotiques. Au meilleur de notre connaissance, nous sommes les premiers à simuler, avec un tel niveau de détails, la projection des pixels d'un capteur caméra dans l'environnement. La simulation est effectuée selon un modèle de caméra sténopé formant une image sur un capteur tessellé de pixels. Pour ce faire, un rayon est lancé à partir du centre d'un pixel et passant par le centre de projection du capteur. Ensuite, la distance parcourue par ce rayon et l'orientation de la première surface atteinte sont utilisées pour calculer l'aire de la base de la pyramide induite par ce pixel. En inversant cette aire nous arrivons à calculer la densité d'échantillonnage en pixels par unité d'aire d'un capteur caméra en ce point.

Nous intégrons ensuite une fonction de combinaison unifiant la densité de pixels de tous les capteurs à une fonction d'erreur mesurant l'écart entre la vue des capteurs placés dans l'environnement et le capteur virtuel. Cette intégration permet de développer une fonction objectif non seulement pour le placement de capteurs hétérogènes, mais aussi indépendante du nombre de capteurs. Cette fonction objectif supporte la résolution du problème posé dans cette thèse, soit le placement de capteurs dans des environnements tridimensionnels non convexes.

7.1.2 Algorithme d'optimisation coopérative

Une seconde contribution importante de cette thèse, présentée au chapitre 3, est le développement d'un algorithme coévolutionnaire efficient. Nous proposons en effet de distribuer les ressources d'optimisation en fonction du rendement des espèces lors de la coévolution coopérative. Nous avons vu que ce problème de distribution se formule en problème de décision où, à chaque génération, on veut maximiser le rendement global de l'algorithme en termes de progression de la fonction objectif. Ce problème étant comparable au problème du bandit manchot, nous suggérons alors l'utilisation d'une modification de l'algorithme UCB pour la sélection des espèces à chaque génération. Les résultats montrent qu'une telle sélection permet d'allouer plus de ressources aux espèces nouvellement introduites dans la coévolution afin de déterminer leur potentiel et, plus tard, de moduler ces ressources en fonction de leur rendement. Nous avons montré que la coévolution coopérative efficiente réussit à trouver une solution satisfaisante plus facilement et plus rapidement que l'algorithme original pour les deux configurations du problème synthétique présentées. Nous avons aussi établi l'avantage de l'algorithme efficient développé lors de plusieurs simulations de placement de capteurs.

Une contribution complémentaire est l'application des algorithmes évolutionnaires coopératifs au domaine du placement de capteurs. En effet, leur utilisation dans ce domaine est, à notre connaissance, inexistante. Nous avons vu que ce type d'algorithme possède des propriétés très intéressantes pour le domaine, comme sa capacité à déterminer le nombre nécessaire de capteurs à l'observation d'une cible et son habileté à gérer une flotte de capteurs hétérogènes. Les résultats présentés au chapitre 5 montrent que leur application permet d'atteindre des configurations de capteurs de qualité supérieure tout en utilisant moins de capteurs et ce plus rapidement qu'un algorithme global et vorace.

7.1.3 Système complet de placement de capteurs

Le système complet de placement de capteurs dans des environnements tridimensionnels fait lui même partie de la liste des contributions importantes de la thèse. Nous avons montré que l'intégration des différentes composantes choisies permet de faire le saut de la simulation à la réalité. En effet, l'implémentation du système sur des robots réels et les résultats présentés au chapitre 6 prouvent que celui-ci est applicable dans des conditions réalistes et avec les technologies disponibles aujourd'hui. Cette démonstration est plus qu'essentielle dans le monde de l'ingénierie où l'on doit concevoir des solutions à des problèmes bien réels avec les outils disponibles. Cette démonstration est aussi incontournable dans le domaine de la recherche

où elle constitue une base solide pour l'avancement des systèmes robotisés dans plusieurs sphères d'activité.

L'équivalence entre le système devant modéliser l'environnement et celui débutant avec un modèle préconstruit constitue une seconde contribution déterminante du système. Nous avons en effet montré aux sections 5.2.3 et 5.2.4 que les configurations de capteurs obtenues avec les deux cas d'utilisation du système sont similaires. Cette similitude fait preuve de la robustesse combinée de l'algorithme d'optimisation et du modèle incrémental de l'environnement. Cette robustesse accentue la valeur de la contribution compte de tenu du type d'application auquel le système est destiné. En effet, un utilisateur peut maintenant s'attendre à une performance similaire du système de placement de capteurs qu'il soit utilisé dans un environnement dont le modèle est disponible ou non.

Une troisième contribution du système est la définition d'un paradigme de manipulation du point de vue désiré. Le capteur virtuel, tel qu'établi dans cette thèse, permet de délimiter la zone d'intérêt de manière simple et transparente. La simplicité d'utilisation de cet instrument vient de son analogie avec un capteur réel. En effet, un utilisateur n'a qu'à penser au capteur réel qu'il voudrait employer, sans se soucier des contraintes physiques, et le placer dans le modèle de l'environnement. Ainsi, l'utilisateur n'a pas à s'embarrasser à tracer des polygones ou des volumes englobants. Des paramètres ayant une signification réelle comme la position, l'orientation, la longueur focale et le champ de vue du capteur virtuel jouent ce rôle. D'autant plus, ces paramètres sont beaucoup plus intuitifs à modifier en cours d'utilisation que les sommets d'un polyèdre. Nous croyons fermement que le capteur virtuel peut trouver son application dans multiples travaux de recherche et applications en placement de capteurs.

7.1.4 Outils logiciels d'algorithmes évolutionnaires

Une dernière contribution majeure du projet doctoral réalisée en collaboration avec Félix-Antoine Fortin étudiant au doctorat au Laboratoire de vision et systèmes numériques de l'Université Laval et qui n'a pas été mise de l'avant dans cette thèse, mais qui a un impact certain dans le domaine de la recherche en algorithme évolutionnaire est le développement d'un outil logiciel d'algorithmes évolutionnaires distribués en Python (DEAP, de l'anglais *Distributed Evolutionary Algorithms in Python*). DEAP n'a peut-être pas présenté en détail dans cette thèse, mais il a rendu possible l'implémentation rapide et efficace de toutes les expérimentations présentées dans cette thèse. En effet, DEAP fournit à la fois un cadre d'application extensible et des algorithmes standards conformes aux implémentations originales qui ont accéléré le processus de développement et de vérification de notre système. De plus,

notre expérience nous confirme que DEAP se marie à merveille avec un logiciel de développement de robots comme ROS, en faisant aussi une contribution significative au domaine de la robotique.

DEAP met de l'avant des principes inédits dans le domaine des logiciels permettant d'implémenter des algorithmes évolutionnaires. Tout d'abord, plutôt que d'encapsuler les algorithmes dans une boîte noire, il promeut leur utilisation explicite. Il force aussi la définition de structures de données transparentes plutôt que d'en cacher les détails. DEAP est conçu sur la base de trois postulats :

1. Les structures de données sont un élément clé des algorithmes évolutionnaires. Leur utilisation doit faciliter l'implémentation des algorithmes et être facile à personnaliser.
2. Les opérateurs et les paramètres des algorithmes ont une forte influence sur les résultats de l'optimisation et dépendent aussi du problème abordé. Leur modification doit donc être simple et centralisée.
3. Les algorithmes évolutionnaires sont généralement hautement parallèles. En conséquence, leur distribution sur une architecture de calcul distribuée doit être triviale.

Ces trois fondements sont respectivement mis en oeuvre de manière élégante par

— un *créateur* permettant de créer des structures de données en une seule ligne de code ;
— une *boîte à outils* contenant tous les opérateurs et leurs paramètres en un endroit centralisé et ;
— un module [1] de parallélisation transformant un algorithme série en algorithme parallèle en manipulant une seule ligne de code.

Avec des parutions à la conférence internationale sur les algorithmes évolutionnaires GECCO (De Rainville et collab., 2012a), dans le journal sur la recherche en apprentissage machine JMLR (Fortin et collab., 2012) et dans le pamphlet du groupe d'intérêt sur les algorithmes évolutionnaires SIGEVOlution (De Rainville et collab., 2014a), DEAP est devenu un logiciel incontournable dans la recherche et l'utilisation des algorithmes évolutionnaires. En effet, il s'est vu attribuer, au cours des deux dernières années, plus de 55 citations par des articles de conférences et de journaux. Le site web de sa documentation est visité par plus de 500 utilisateurs de 50 pays par semaine, tel que rapporté par les outils d'analyse de Google. DEAP fait part intégrante, depuis l'automne 2014, de l'apprentissage des algorithmes évolutionnaires dans le cadre du cours « Advanced Evolutionary Computation : Theory and Practice » donné à la Pontifical Catholic University of Rio de Janeiro, Brésil [2]. Ceci fait de DEAP une

1. SCOOP : Scalable Concurent Operations in Python http://www.pyscoop.org
2. http://lmarti.com/aec-2014

contribution importante des travaux doctoraux, même s'il est hors du sujet principal de cette thèse.

7.2 Perspectives

Les travaux présentés dans cette thèse ouvrent de nombreux axes de recherche aux systèmes de placement de capteurs. En effet, la possibilité d'optimiser le placement de capteurs dans des environnements tridimensionnels sans contraintes ouvre la porte à maintes possibilités.

Tout d'abord, nous avons présenté une qualité d'acquisition de type membrane ne prenant en compte que les surfaces de l'environnement. Toutefois, dans un cadre de surveillance, un capteur caméra doit observer non pas seulement les surfaces de l'environnement, mais aussi son volume. Le modèle de capteur proposé permettrait en effet de développer un tel type de qualité d'acquisition en mesurant le volume de la pyramide de vision des pixels plutôt que l'aire de leur base.

Ensuite, la fonction objectif proposée ne pénalise ni n'encourage directement le recouvrement des vues des capteurs. Un recouvrement minimal est souvent souhaitable pour l'assemblage de l'information contenue dans différentes vues. Il serait fort intéressant de contrôler et garantir un recouvrement optimal en incorporant cette information préalable dans la fonction objectif. Dans ce même ordre d'idée, nous pourrions aussi inclure toute autre hypothèse faite par l'algorithme d'assemblage dans la fonction objectif afin de garantir une composition optimale de l'information de sortie du système.

Par la suite, il serait digne d'intérêt d'étudier le comportement du système lorsque les paramètres de la caméra virtuelle changent en cours d'utilisation. En effet, les simulations présentées n'exploitent pas le plein potentiel de la caméra virtuelle. Elles se limitent à une caméra virtuelle restant constante tout au long d'une expérimentation. Nous pourrions ainsi ajouter un mécanisme permettant d'étendre la couverture des capteurs réels afin de prévoir les changements de la caméra virtuelle et éviter de déplacer les capteurs à chaque modification de paramètre. Ce mécanisme pourrait prendre plusieurs formes. Par exemple, la coévolution compétitive pourrait être utilisée afin d'opposer un ensemble de caméras virtuelles aux configurations de capteurs dans le but de trouver un ensemble de capteurs répondant bien à toutes ces différentes configurations. Ou encore, l'apprentissage machine pourrait fournir un ensemble de caméra virtuelles probables choisies en fonction des dernières modifications effectuées à sa configuration.

Un dernier point partiellement abordé dans cette thèse est le dynamisme de l'environnement. En effet, l'optimisation se déroule dans un modèle changeant au fil de l'exploration, mais l'environnement lui-même ne bouge pas. Pour considérer le mouvement de l'environnement, il faudrait tout d'abord développer un modèle pouvant le représenter. Cette représentation serait tout à fait possible dans une grille d'occupation où, en plus de la probabilité d'occupation, chaque voxel pourrait contenir une valeur modélisant la dynamique de cette partie du monde. Une fois cette modélisation faite on pourrait demander aux capteurs d'éviter d'avoir une ligne de vue passant par ces zones ou, au contraire, leur demander de les observer en priorité pour étudier ce qui s'y déroule.

7.3 Impact

La révolution robotique, telle qu'annoncée par Rodney Brooks, est bel et bien en marche (Ulbrick, 2008). En effet, la multiplication récente des applications des drones et leur réglementation ne sont que le début de ce qui s'annonce comme étant la personnalisation de la robotique. Cette révolution devrait démocratiser l'utilisation des robots sous toutes leurs formes de manière similaire à ce qu'a vécu le *World Wide Web* il y a une quinzaine d'années. Ces différents robots ne seront pas uniquement chargés d'entretenir les foyers et de conduire des véhicules, ils permettront aussi de communiquer, d'explorer et d'inspecter. Dans ces cadres d'application, l'impact de la présente thèse peut être immense. En généralisant l'application des capteurs mobiles aux environnements tridimensionnels non convexes et initialement inconnus, elle permettra de mettre à contribution le gigantesque parc de capteurs que seront ces robots pour des applications allant de la visite d'un musée à l'évaluation de la situation après un cataclysme.

Bibliographie

Agarwala, A., M. Dontcheva, M. Agrawala, S. Drucker, A. Colburn, B. Curless, D. Salesin et M. Cohen. 2004, «Interactive digital photomontage», *ACM Transactions on Graphics*, vol. 23, n^{o} 3, p. 294–302.

Agrawal, S. et N. Goyal. 2013, «Further optimal regret bounds for Thompson sampling», dans *Proc. of International Conference on Artificial Intelligence and Statistics*, p. 99–107.

Akbarzadeh, V., C. Gagné, M. Parizeau, M. Argany et M. A. Mostafavi. 2013, «Probabilistic sensing model for sensor placement optimization based on line-of-sight coverage», *IEEE Transactions on Instrumentation and Measurement*, vol. 62, n^{o} 2, p. 293–303.

Akbarzadeh, V., C. Gagné, M. Parizeau et M. A. Mostafavi. 2011, «Black-box optimization of sensor placement with elevation maps and probabilistic sensing models», dans *Proc. of the International Symposium on Robotics and Sensor Environments*, p. 89–94.

Amanatides, J. et A. Woo. 1987, «A fast voxel traversal algorithm for ray tracing», dans *Proc. of Eurographics*, p. 3–10.

Auer, P., N. Cesa-Bianchi et P. Fischer. 2002, «Finite-time analysis of the multi-armed bandit problem», *Machine Learning*, vol. 47, n^{o} 2–3, p. 235.

Auger, A., S. Finck, N. Hansen et R. Ros. 2010, «Comparison tables : BBOB 2009 function testbed.», cahier de recherche 0383, INRIA.

Auger, A. et N. Hansen. 2009, «Benchmarking the (1 + 1)-CMA-ES on the BBOB-2009 testbed.», dans *Proc. of the Genetic and Evolutionary Computation Conference Companion*, p. 2459–2466.

Auger, A. et N. Hansen. 2013, «On proving linear convergence of comparison-based step-size adaptive randomized search on scaling-invariant functions via stability of markov chains», *arXiv preprint arXiv :1310.7697*.

Axelrod, R. 1987, «The evolution of strategies in the iterated prisoner's dilemma», dans *Genetic algorithms and simulated annealing*, édité par L. Davis, Pitman Publishing, p. 32–41.

Beyer, H.-G. et H.-P. Schwefel. 2002, «Evolution strategies – a comprehensive introduction», *Natural Computing*, vol. 1, p. 3–52.

Bhattacharya, S., N. Michael et V. Kumar. 2010, «Distributed coverage and exploration in unknown non-convex environments», *Distributed Autonomous Robotic Systems*, vol. 83.

Blaer, P. S. et P. K. Allen. 2009, «View planning and automated data acquisition for three-dimensional modeling of complex sites», *Journal of Field Robotics*, vol. 26, p. 865–891.

Bortoff, S. A. 2000, «Path planning for UAVs», dans *Proc. of the American Control Conference*, vol. 1, IEEE, p. 364–368.

Breitenmoser, A., M. Schwager, J.-C. Metzger, R. Siegwart et D. Rus. 2010, «Voronoi coverage of non-convex environments with a group of networked robots», dans *Proc. of the IEEE International Conference on Robotics and Automation*, p. 4982–4989.

Brooks, R. A. 1985, «Visual map making for a mobile robot», *Artificial Intelligence*, p. 824–829.

Brown, M. et D. G. Low. 2007, «Automatic panoramic image stitching using invariant features», *Internation Journal of Computer Vision*, vol. 74, n° 1, p. 59–73.

Broyden, C. G. 1970, «The convergence of a class of double-rank minimization algorithms», *Journal of the Institute os Mathematics and Its Application*, vol. 6, p. 76–90.

Bullo, F., J. Cortés et S. Martínes. 2009, *Distributed control of robotic networks : a mathematical approach to motion coordination algorithms*, Princeton University Press.

Burgard, W., M. Moors, C. Stachniss et F. E. Schneider. 2005, «Coordinated multi-robot exploration», *IEEE Transactions on Robotics*, vol. 21, n° 3, p. 376–386.

Chatila, R. et J.-P. Laumond. 1985, «Position referencing and consistent world modeling for mobile robots», dans *Proc. of the IEEE International Conference on Robotics and Automation*, vol. 2, p. 138–145.

Chen, W., Y. Wang et Y. Yuan. 2013, «Combinatorial multi-armed bandit : General framework, results and applications», dans *Proc. of the International Conference on Machine Learning*, p. 151–159.

Connolly, C. 1985, «The determination of next best views», dans *Proc. of the International Conference on Robotics and Automation*, p. 432–435.

Cortés, J., S. Martínes, T. Karatas et F. Bullo. 2004, «Coverage control for mobile sensing networks», *Transaction on Robotics and Automation*, vol. 20, n^{o} 2, p. 243–255.

Da Costa, L., A. Fialho, M. Schoenauer et M. Sebag. 2008, «Adaptive operator selection with dynamic multi-armed bandits», dans *Proc. of the Genetic and Evolutionary Computation Conference*, p. 913–920.

Dai, W., Y. Gai, B. Krishnamachari et Q. Zhao. 2011, «The non-bayesian restless multi-armed bandit : A case of near-logarithmic regret», dans *Proc. of the International Conference on Acoustics, Speech and Signal Processing*, IEEE, p. 2940–2943.

De Filippis, L., G. Guglieri et F. Quagliotti. 2012, «Path planning strategies for UAVS in 3D environments», *Journal of Intelligent & Robotic Systems*, vol. 65, n^{o} 1-4, p. 247–264.

De Rainville, F.-M., F.-A. Fortin, M.-A. Gardner, M. Parizeau et C. Gagné. 2012a, «DEAP : A python framework for evolutionary algorithms», dans *Proc. of the Genetic and Evolutionary Computation Conference Companion*, ACM, p. 85–92.

De Rainville, F.-M., F.-A. Fortin, M.-A. Gardner, M. Parizeau et C. Gagné. 2014a, «DEAP – Enabling nimbler evolutions», *SIGEVOlution*, vol. 6, n^{o} 2, p. 17–26.

De Rainville, F.-M., C. Gagné et D. Laurendeau. 2012b, «Co-adapting mobile sensor networks to maximize coverage in dynamic environments», dans *Proc. of the Genetic and Evolutionary Computation Conference Companion*, p. 1409–1410.

De Rainville, F.-M., C. Gagné et D. Laurendeau. 2014b, «Automatic sensor placement for complex three-dimensional inspection and exploration», dans *International Symposium on Artificial Intelligence, Robotics and Automation in Space*.

De Rainville, F.-M., J.-P. Mercier, C. Gagné, P. Giguère et D. Laurendeau. 2015, «Multisensor placement in 3D environments via visibility estimation and derivative-free optimization», dans *Proc. of the International Conference on Robotics and Automation*.

De Rainville, F.-M., M. Sebag, C. Gagné, M. Schoenauer et D. Laurendeau. 2013, «Sustainable cooperative coevolution with a multi-armed bandit», dans *Proc. of the Genetics and Evolutionary Computation Conference*, p. 1517–1524.

Dornhege, C., A. Kleiner et A. Kolling. 2013, «Coverage search in 3D», dans *Proc. of the Int. Symp. on Safety, Security, and Rescue Robotics*, p. 1–8.

Dudek, G., M. R. M. Jenkin, E. Milios et D. Wilkes. 1996, «A taxonomy for multi-agent robotics», *Autonomous Robots*, vol. 3, n° 4, p. 375–397.

Eden, A., M. Uyttendaele et R. Szeliski. 2006, «Seamless image stitching of scenes with large motions and exposure differences», dans *IEEE Conference on Computer Vision and Pattern Recognition*, vol. 2, IEEE, p. 2498–2505.

Elfes, A. 1989, «Using occupancy grids for mobile robot perception and navigation», *Computer*, vol. 22, n° 6, p. 46–57.

Fialho, A., M. Schoenauer et M. Sebag. 2010, «Toward comparison-based adaptive operator selection», dans *Proc. of the Genetic and Evolutionary Computation Conference*, p. 767–774.

Fletcher, R. 1970, «A new approach to variable metric algorithms», *Computer Journal*, vol. 13, p. 317–322.

Forrest, S., B. Javornik, R. E. Smith et A. S. Perelson. 1993, «Using genetic algorithms to explore pattern recognition in the immune system», *Evol. Comput.*, vol. 1, n° 3, p. 191–211.

Fortin, F.-A., F.-M. D. Rainville, M.-A. Gardner, M. Parizeau et C. Gagné. 2012, «DEAP : Evolutionary algorithms made easy», *Journal of Machine Learning Research, Machine Learning Open Source Software*, vol. 13.

Fox, D. 2003, «Adapting the sample size in particle filters through kld-sampling», *The International Journal of Robotics Research*, vol. 22, n° 12, p. 985–1003.

Fox, D., W. Burgard, F. Dellaert et S. Thrun. 1999, «Monte carlo localization : Efficient position estimation for mobile robots», dans *Proc. of the Conference of Innovative Applications of Artificial Intelligence (AAAI)*, p. 343–349.

Fox, D., W. Burgard et S. Thrun. 1997, «The dynamic window approach to collision avoidance», *Robotics & Automation Magazine*, vol. 4, n° 1, p. 23–33.

Friedrichs, F. et C. Igel. 2005, «Evolutionary tuning of multiple SVM parameters», *Neurocomputing*, vol. 64, p. 107–117.

Gage, D. W. 1992, «Command control for many-robot systems», *Unmanned Systems Magazine*, vol. 10, n° 4, p. 28–34.

García-Pedrajas, N., C. Hervás-Martínez et D. Ortiz-Boyer. 2005, «Cooperative coevolution of artificial neural network ensembles for pattern classification», *Transaction on Evolutionary Computation*, vol. 9, n° 3, p. 271–302.

Gittins, J. C. 1979, «Bandit processes and dynamic allocation indices», *Journal of the Royal Statistical Society*, vol. 41, n° 2, p. 148–177.

Goldfarb, D. 1970, «A family of variable metric updates derived by variational means», *Mathematics of Computation*, vol. 24, p. 23–26.

Gonzáles-Baños, H., A. Efrat, J. C. Latombe, E. Mao et T. M. Murali. 2000, «Planning robot motion strategies for efficient model construction», dans *Proc. of the International Symposium on Robotics Research*, p. 345–352.

Grisetti, G., C. Stachniss et W. Burgard. 2007, «Improved techniques for grid mapping with rao-blackwellized particle filters», *IEEE Transactions on Robotics*, vol. 23, p. 34–46.

Hansen, N. et A. Ostermeier. 2001, «Completely derandomized self-adaptation in evolution strategies», *Evolutionary Computation*, vol. 9, n° 2, p. 159–195.

Hillis, W. D. 1990, «Co-evolving parasites improves simulated evolution as an optimization procedure», *Physica D : Nonlinear Phenomena*, vol. 42, n° 1–3, p. 228–234.

Hornung, A., K. M. Wurm, M. Bennewitz, C. Stachniss et W. Burgard. 2013, «OctoMap : an efficient probabilistic 3D mapping framework based on octrees», *Autonomous Robots*, vol. 34, n° 3, p. 189–206.

Howard, A., M. J. Matarić et G. S. Sukhatme. 2002, «Mobile sensor network deployment using potential fields : A distributed, scalable solution to the area coverage problem», dans *Proc. of the International Symposium on Distributed Autonomous Robotics Systems*, p. 299–308.

Kamon, I., E. Rimon et E. Rivlin. 1998, «TangentBug : A range-sensor-based navigation algorithm», *International Journal of Robotics Research*, vol. 17, n° 9, p. 934–953.

Kantaros, Y., M. Thanou et A. Tzes. 2014, «Visibility-oriented coverage control of mobile robotic networks on non-convex regions», dans *Proc. of the IEEE International Conference on Robotics and Automation*, p. 1126–1131.

Kirkpatrick, J., C. D. Gelatt et M. P. Vecchi. 1983, «Optimization by simulated annealing», *Science*, vol. 220, n° 4598, p. 671–680.

Klodt, L., D. Haumann et V. Willert. 2014, «Revisiting coverage control in nonconvex environments using visibility sets», dans *Proc. of the IEEE International Conference on Robotics and Automation*, p. 82–89.

Koenig, N. et A. Howard. 2004, «Design and use paradigms for Gazebo, an open-source multi-robot simulator», dans *Proc. of the International Conference on Intelligent Robots and Systems*, p. 2149–2154.

Lai, T. et H. Robbins. 1985, «Asymptotically efficient adaptive allocation rules», *Adv. Appl. Math.*, vol. 6, n^{o} 1, p. 4–22.

Laventall, K. et J. Cortés. 2009, «Coverage control by multi-robot networks with limited-range anisotropic sensory», *International Journal of Control*, vol. 82, n^{o} 6, p. 1113–1121.

Lin, Y. et S. Saripalli. 2014, «Path planning using 3d dubins curve for unmanned aerial vehicles», dans *Proc. of the International Conference on Unmanned Aircraft Systems*, p. 296–304.

Lorensen, W. E. et H. E. Cline. 1987, «Marching Cubes : A high resolution 3D surface construction algorithm», *Computer Graphics*, vol. 21, n^{o} 4, p. 163–169.

Low, K.-L. et A. Lastra. 2006, «Efficient constraint evaluation algorithms for hierarchical next-best-view planning», dans *Proc. of the International Symposium on 3D Data Processing, Visualization, and Transmission*, p. 830–837.

Marder-Eppstein, E., E. Berger, T. Foote, B. Gerkey et K. Konolige. 2010, «The office marathon : Robust navigation in an indoor office environment», dans *Proc. of the International Conference on Robotics and Automation*, p. 300–307.

Marier, J.-S., C.-A. Rabbath et N. Léchevin. 2012, «Visibility-limited coverage control using nonsmooth optimization», dans *Proc. of the American Control Conference*, p. 6029–6034.

Meyer, J., A. Sendobry, S. Kohlbrecher, U. Klingauf et O. von Stryk. 2012, «Comprehensive simulation of quadrotor UAVs using ROS and Gazebo», dans *Simulation, Modeling, and Programming for Autonomous Robots*, Springer, p. 400–411.

Mitchell, M. 1998, *An Introduction to Genetic Algorithms*, MIT Press.

Moravec, H. P. 1988, «Sensor fusion in certainty grids for mobile robots», *AI Magazine*, vol. 9, n^{o} 2, p. 61–74.

Murphy, K. 1999, «Bayesian map learning in dynamic environments.», dans *Proc. of the Conference on Neural Information Processing Systems*, p. 1015–1021.

Nüchter, A., H. Surmann et J. Hertzberg. 2003, «Planning robot motion for 3D digitalization of indoor environments», dans *Proc. of the International Conference on Advanced Robotics*, p. 222–227.

O'Rourke, J. 1987, *Art Gallery Theorems and Algorithms*, Oxford University Press.

Panait, L. et S. Luke. 2005, «Cooperative multi-agent learning : The state of the art», *Auton. Agent Multi Agent Syst.*, vol. 11, p. 387–434.

Pérez, P., M. Gangnet et A. Blake. 2003, «Poisson image editing», *ACM Transactions on Graphics*, vol. 22, n° 3, p. 313–318.

Pimenta, L. C. A., V. Kumar, R. C. Mesquita et G. A. S. Pereiea. 2008, «Sensing and coverage for a network of heterogeneous robots», dans *Proc. of the IEEE Conference on Decision and Control*, p. 3947–3952.

Pito, R. 1999, «A solution to the next best view problem for automated surface acquisition», *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 21, n° 10, p. 1016–1030.

Potter, M. A. et K. A. De Jong. 2001, «Cooperative coevolution : An architecture for evolving co-adapted subcomponents», *Evolutionary Computation*, vol. 8, n° 1, p. 1–29.

Quigley, M., K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler et A. Y. Ng. 2009, «ROS : An open-source Robot Operating System», dans *Proc. of the ICRA Workshop on Open Source Software*, p. 5.

Rechenberg, I. 1973, *Evolutionsstrategie*, Friedrich Frommann Verlag, Stuttgart, Stuttgart.

Robbins, H. 1952, «Some aspects of the sequential design of experiments», *Bulletin of the American Mathematical Society*, vol. 58, n° 5, p. 527–535.

Şahin, E. 2005, «Swarm robotics : From sources of inspiration to domains of application», dans *Swarm Robotics*, *Lecture Notes in Computer Science*, vol. 3342, p. 10–20.

Sasiadek, J. et I. Dulęba. 2000, «3D local trajectory planner for UAV», *Journal of Intelligent and Robotic Systems*, vol. 29, n° 2, p. 191–210.

Schwager, M., B. J. Julian, M. Angermann et D. Rus. 2011a, «Eyes in the sky : Decentralized control for the deployment of robotic camera networks», *Proceedings of the IEEE*, vol. 99, n° 9, p. 1541–1561.

Schwager, M., B. J. Julian et D. Rus. 2009, «Optimal coverage for multiple hovering robots with downward facing cameras», dans *Proc. of the International Conference on Robotics and Automation*, p. 3515–3522.

Schwager, M., D. Rus et J. J. Slotine. 2011b, «Unifying geometric, probabilistic, and potential field approaches to multi-robot deployment», *International Journal of Robotics Research*, vol. 30, n° 3, p. 371–383.

Shanno, D. F. 1970, «Conditioning of quasi-newton methods for function minimization», *Mathematics of Computation*, vol. 24, p. 647–656.

Smith, R. E., S. Forrest et A. S. Perelson. 1993, «Searching for diverse, cooperative populations with genetic algorithms», *Evolutionary Computation*, vol. 1, n° 2, p. 127–149.

Tarbox, G. H. et S. N. Gottschlich. 1995, «Planning for complete sensor coverage in inspection», *Computer Vision and Image Understanding*, vol. 61, n° 1, p. 84–111.

Tettamanzi, A. G., C. Dartigues-Pallez, C. da Costa Pereira, D. Pallez et P. Gourbesville. 2011, «Coastal current prediction using CMA evolution strategies», dans *Proc. of the Conference on Genetic and Evolutionary Computation*, ACM, p. 1715–1722.

Thrun, S. 2003, «Learning occupancy grid maps with forward sensor models», *Autonomous Robots*, vol. 15, p. 111–127.

Thrun, S. et J. J. Leonard. 2008, «Simultaneous localization and mapping», dans *Springer Handbook of Robotics*, édité par B. Siciliano et O. Khatib, Springer Berlin Heidelberg, p. 871–889.

Thrun, S. et Y. Liu. 2003, «Multi-robot SLAM with sparse extended information filers», dans *Proc. of the International Symposium on Robotics Research*, p. 254–266.

Ulbrick, A. 2008, «Rodney's Robot Revolution», Directed by Andrea Ulbrick.

Whittle, P. 1988, «Restless bandits : Activity allocation in a changing world», *Journal of Applied Probability*, p. 287–298.

Wu, S. X. et W. Banzhaf. 2010, «A hierarchical cooperative evolutionary algorithm», dans *Proc. of the Genetic and Evolutionary Computation Conference*, p. 233–240.

Wurm, K. M., A. Hornung, M. Bennewitz, C. Stachniss et W. Burgard. 2010, «OctoMap : A probabilistic, flexible, and compact 3D map representation for robotic systems», dans *Proc. of the ICRA Workshop on Best Practice in 3D Perception and Modeling for Mobile Manipulation*, p. 8.

Yap, F. G. H. et H.-H. Yen. 2014, «A survey on sensor coverage and visual data capturing/processing/transmission in wireless visual sensor networks», *Sensors*, vol. 14, n° 2, p. 3506–3527.

Yguel, M., O. Aycard et C. Laugier. 2008, «Update policy of dense maps : Efficient algorithm and sparse representation», dans *Results of the International Conference Field and Service Robotics*, vol. 42, p. 23–33.

Zaragosa, J., T.-J. Chin, Q.-H. Tran, M. S. Brown et D. Suter. 2014, «As-projective-as-possible image stitching with moving DLT», *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, n° 7, p. 1285–1298.

Zhang, Z. 1997, «Parameter estimation techniques : A tutorial with application to conic fitting», *Image and Vision Computing*, vol. 15, n° 1, p. 59–76.

Annexe A

Détails d'implémentation

Dans cette annexe nous verrons quelques détails d'implémentation qui rendent possibles certains aspects du système. Ces détails ne sont pas indispensables, cependant ils règlent plusieurs irrégularités pouvant survenir lors des expérimentations.

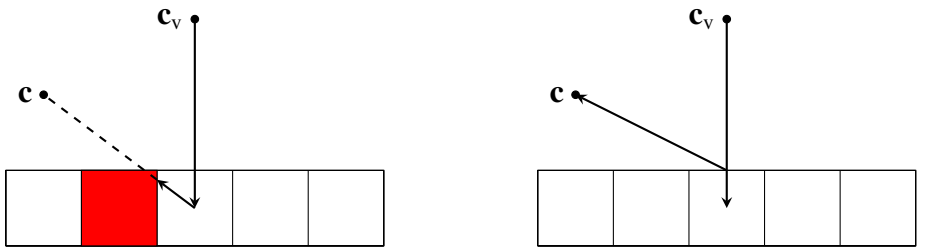
A.1 Sélection des points d'intérêt

La sélection des points d'intérêt décrite à la sélection 2.3.4 requiert un échantillonnage du champ de vue du capteur virtuel. Un échantillonnage aléatoire provoquerait une instabilité de l'algorithme d'optimisation alors que deux configurations de capteurs identiques pourraient obtenir une erreur normalisée de densité de pixels différente. La méthode d'échantillonnage déterministe est donc

- pour le **capteur caméra** une grille projetée sur une sphère, c'est-à-dire que les points sont sélectionnés à intervalle régulier d'angle et
- pour le **capteur omniscient** une grille tridimensionnelle sélectionnant tous les voxels occupés et inconnus.

A.2 Point de contact dans l'octree

Lors du lancer de rayons pour déterminer les points d'intérêt du capteur virtuel de type caméra, les points retournés par l'algorithme sont les centres des voxels atteints. Tel que présenté à la figure A.1(a), la rétroprojection à partir du centre du voxel retourné par le lancer du rayon de $\mathbf{c}_v$ est impossible en direction de $\mathbf{c}$, car le rayon frappe le voxel occupé situé à gauche de voxel trouvé. Pour éviter cette erreur, plutôt que de lancer le rayon de rétroprojection du centre du voxel, nous calculons le point d'entrée dans le voxel du rayon partant du



(a) Erreur provoquée par la rétroprojection d'un rayon à partir du centre d'un voxel vers un capteur réel, mais frappant le voxel voisin (en rouge) dû à l'angle de retour.

(b) Correction de l'erreur par rétroprojection à partir du point d'entrée du rayon et non du centre du voxel.

FIGURE A.1 – Erreur due aux angles de rétroprojection rasants.

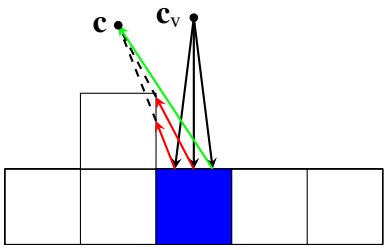


FIGURE A.2 – Projection et rétroprojection de trois rayons échantillonnés dans le champ de vue du capteur virtuel $\mathbf{c}_v$ et erreur causée par la vue partielle du voxel bleu par $\mathbf{c}$.

capteur virtuel et utilisons ce point d'entrée comme point de départ du rayon de rétroprojection vers le capteur réel, tel qu'illustré à la figure A.1(b). Dans le cas du capteur omniscient, il est impossible de prévenir cette erreur.

A.3 Discrétisation du modèle d'environnement

La grille d'occupation discrétise l'environnement de telle manière que l'ensemble des points d'intérêt échantillonnés peut contenir plusieurs échantillons par voxel. De plus, il est probable qu'un capteur ne voie que partiellement un voxel donné tel que présenté à la figure A.2. Nous avons choisi de ne garder que la valeur maximale de densité de pixels par voxel pour chaque capteur, car il est fort peu probable dans le monde réel que le rayon du centre ou même le rayon de gauche soit réellement obstrués. Ainsi, la densité effective du capteur $\mathbf{c}$ pour le voxel bleu est donnée par le rayon de rétroprojection complet (vert).

A.4 Filtres appliqués aux données 3D

Les filtres appliqués aux données provenant des capteurs tridimensionnels permettent de réduire la bande passant requise pour la transmission de ces données et la complexité de mise à jour du modèle de l'environnement. Nous utilisons deux filtres, l'un réduisant le taux d'échantillonnage du capteur tridimensionnel et l'autre filtrant les points correspondant aux robots hors du modèle de l'environnement. Le premier filtre applique une grille tridimensionnelle aux données 3D. Les données sont décimées en approximant tous les points à l'intérieur d'un voxel de la grille par leur centroïde. La taille des voxels est choisie égale à celle du modèle de l'environnement. Le second filtre est une boîte apposée à la position de chaque robot. Toutes les données à l'intérieur de cette boîte sont retranchées afin de ne pas inclure les robots dans le modèle de l'environnement. La taille de la boîte est déterminée par le type de robot.

A.5 Prévention de l'observation des robots

Le filtre boîte permet au système d'exclure de la vue désirée les robots se trouvant dans son champ de vue. Malheureusement, ce filtre interfère aussi avec une caractéristique importante du système. Il retranche la possibilité d'éviter que les robots se bloquent la vue entre eux. En effet, les simulations se déroulant dans le modèle de l'environnement, si celui-ci ne possède pas d'information sur la position des robots, il est impossible de prévenir qu'ils ne s'obstruent pas la vue. Pour parer cette éventualité, nous maintenons deux modèles de l'environnement : l'un modélisant la géométrie complète incluant les robots et l'autre les excluant. Ainsi, la simulation des capteurs réels est faite dans le premier modèle pour que leur vision soit affectée par les autres robots et la simulation de la vue virtuelle se fait dans le second modèle pour ne pas être perturber par ces derniers. Les données entrant dans chacun de ces modèles sont respectivement celles filtrées uniquement par le filtre en grille et celles filtrées par les deux filtres énoncés précédemment.

Annexe B

Liste des publications

B.1 Revues scientifiques

— Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner, Marc Parizeau and Christian Gagné, 2012, « DEAP : Evolutionary Algorithms Made Easy », *Journal of Machine Learning Research, Machine Learning Open Source Software*, p. 2171–2175, vol. 13.

B.2 Conférences

— François-Michel De Rainville, Jean-Philippe Mercier, Christian Gagné, Philippe Giguère et Denis Laurendeau, 2015, « Multisensor Placement in 3D Environments via Visibility Estimation and Derivative-Free Optimization », dans *Proc. of the International Conference on Robotics and Automation*.

— François-Michel De Rainville, Christian Gagné et Denis Laurendeau, 2014, « Automatic Sensor Placement For Complex Three-dimensional Inspection and Exploration », dans *Proc. of the International Symposium on Artificial Intelligence, Robotics, and Automation in Space*.

— François-Michel De Rainville, Michèle Sebag, Christian Gagné, Marc Schoenauer et Denis Laurendeau, 2013, « Sustainable Cooperative Coevolution with a Multi-Armed Bandit », dans *Proc. of the Genetic and Evolutionary Computation Conference*, p. 1517–1524.

— François-Michel De Rainville, Félix-Antoine Fortin, Marc-André Gardner, Marc Parizeau et Christian Gagné, 2012, « DEAP : A Python Framework for Evolutionary

Algorithms », dans *EvoSoft Workshop, Companion proc. of the Genetic and Evolutionary Computation Conference*, p. 85–92.

— François-Michel De Rainville, Christian Gagné et Denis Laurendeau, 2012 « Coadapting Mobile Sensor Networks to Maximize Coverage in Dynamic Environments », dans *Companion proc. of the Genetic and Evolutionary Computation Conference*, p. 1409–1410.

B.3 Autres

— François-Michel De Rainville, Félix-Antoine Fortin, Marc-André Gardner, Marc Parizeau et Christian Gagné, 2014, « DEAP – Enabling Nimbler Evolutions », *SIGEvolution*, vol. 6, n^{o} 3, p. 17–26.