



# THÈSE

En vue de l'obtention du

## DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par :

Institut Supérieur de l'Aéronautique et de l'Espace

---

**Présentée et soutenue par :**

**Jean-Alexis DELAMER**

**le** mardi 1 octobre 2019

**Titre :**

Planification de stratégies de navigation et de guidage pour des drones autonomes dans des milieux encombrés

---

**École doctorale et discipline ou spécialité :**

EDSYS : Robotique et Automatique

**Unité de recherche :**

Équipe d'accueil ISAE-ONERA CSDV

**Directeur(s) de Thèse :**

Mme Yoko WATANABE (directrice de thèse)  
Mme Caroline CHANEL (co-directrice de thèse)

**Jury :**

M. Patrick DANÈS Professeur Université Toulouse 3 - Président  
Mme Caroline CHANEL Ingénieure de recherche ISAE-SUPAERO - Co-directrice de thèse  
M. Pascal MORIN Professeur Université Pierre & Marie Curie - Rapporteur  
M. Abdel-Ilah MOUADDIB Professeur Université de Caen Normandie  
Mme Yoko WATANABE Ingénieure de recherche ONERA - Directrice de thèse  
M. Bruno ZANUTTINI Professeur Université de Caen Normandie - Rapporteur



I love deadlines. I love the whooshing noise they make as they go by.

---

Douglas Adams

A learning experience is one of those things that says, 'You know that thing you just did? Don't do that.'

---

Douglas Adams



# Remerciements

Je tiens à rendre hommage à toutes les personnes qui ont rendu cette thèse possible.

Tout d'abord, je tiens à remercier les membres du jury pour leur temps et leur travail : M. Bruno Zanuttini, M. Pascal Morin, M. Patrick Danès et M. Adbell-Illah Mouaddib.

Je tiens également à remercier Yoko et Caroline pour ces trois années pendant lesquelles j'ai beaucoup appris. Merci pour votre confiance, pour tous vos conseils et pour le temps que vous m'avez consacré.

Merci à l'ONERA, à l'ISAE-SUPAERO et à EDSYS de m'avoir aidé à faire une mobilité au Canada.

Un grand merci à Mme Claudine Touron et Mme Valérie Cassignol pour leur aide dans les différentes démarches administratives, ainsi que leur patience pour répondre aux nombreuses questions que j'ai pu avoir. Merci à Camille Blosse pour l'aide apportée dans la mise en place de ma mobilité.

I would like to thank Sidney Givigi for these few months working with you at RMC. Thank you for your warm welcome and for all the great moments with you and the team.

Impossible d'oublier de remercier mon wingman de bureau et ami Anthony. Je n'oublierai pas les parties de Magic, surtout les nombreuses que tu as perdues ! Pour moi, tu seras toujours ce que Goose était à Maverick ! Qatlho, wIj jup.

Je remercie les hommes de l'ombre : Jacques et Tanguy mes vieux copains ;).

Je dédis cette thèse à mes parents et à Pia sans qui rien n'aurait été possible.



# Table des matières

<b>Table des sigles et acronymes</b>	<b>xv</b>
<b>Introduction</b>	<b>1</b>
<b>I État de l’art</b>	<b>7</b>
<b>1 Processus décisionnels de Markov partiellement observable - POMDP</b>	<b>9</b>
1.1 Processus décisionnels de Markov . . . . .	10
1.2 <i>Stochastic Shortest-Path</i> MDP . . . . .	12
1.3 Processus Décisionnels de Markov Partiellement Observables . . . . .	13
1.4 Extension du POMDP pour les espaces d’états et d’observations continus . .	20
1.5 Méthodes approximatives d’itération sur la valeur . . . . .	21
1.6 MOMDP - <i>Mixed Observability Markov Decision Process</i> . . . . .	27
1.7 Bilan . . . . .	29
<b>2 Méthodes existantes pour la planification de chemin sous incertitude de localisation et/ou d’exécution</b>	<b>31</b>
2.1 Algorithmes de planification de chemins sous l’incertitude sur la localisation .	31
2.2 Utilisation des POMDP pour le calcul de chemins . . . . .	37
2.3 Conclusion et Bilan . . . . .	39

<b>II</b>	<b>Planification de chemins efficaces et sûrs par la prise en compte de la dégradation de la précision de la localisation et de la disponibilité des capteurs</b>	<b>41</b>
<b>3</b>	<b>Modèle de transition GNC pour représenter le mouvement d'un véhicule</b>	<b>43</b>
3.1	Définition de l'architecture du système . . . . .	43
3.2	Modèle de transition GNC . . . . .	44
3.3	Bilan . . . . .	50
<b>4</b>	<b>Proposition d'un modèle MOMDP et des algorithmes de résolution avec représentation d'état de croyance par des modèles de mélange gaussiens</b>	<b>51</b>
4.1	Connaissances a priori . . . . .	52
4.2	Modèle de planification basé MOMDP . . . . .	54
4.3	Algorithmes de résolution . . . . .	65
4.4	Expérimentations . . . . .	71
4.5	Bilan . . . . .	76
<b>5</b>	<b>Évolution du modèle MOMDP et algorithme de résolution</b>	<b>79</b>
5.1	Présentation du nouveau modèle MOMDP . . . . .	80
5.2	Algorithme de résolution . . . . .	81
5.3	Expérimentations . . . . .	87
5.4	Bilan . . . . .	96
<b>6</b>	<b>Étude du lien entre la fonction de coût et les politiques acceptables</b>	<b>97</b>
6.1	Étude du lien entre la fonction de coût et les politiques acceptables . . . . .	97
6.2	Expérimentations . . . . .	102
6.3	Bilan . . . . .	120
<b>7</b>	<b>Conclusion et perspectives</b>	<b>121</b>
7.1	Conclusion . . . . .	121



7.2 Perspectives . . . . .	124
<b>A Étude de l'impact du coefficient d'exploration sur la convergence de la valeur</b>	<b>127</b>
<b>B Étude de l'impact de la durée de l'époque de planification sur la convergence et les chemins planifiés</b>	<b>135</b>
<b>C Représentativité des simulations</b>	<b>139</b>
<b>Bibliographie</b>	<b>145</b>



## Table des figures

1	Système GNC schématisé . . . . .	2
2	Calcule de PDOP la carte de la ville de San Diego. . . . .	3
1.1	Interaction d'un agent d'un MDP avec son environnement . . . . .	10
1.2	Le modèle de transition d'un MDP dans lequel un agent dans un état $s_t$ a une action $a_t$ définie. Après l'exécution de cette action, il se retrouve dans un nouvel état $s_{t+1}$ et une récompense $r(s_t, a_t)$ est alors obtenue de façon déterministe. . . . .	11
1.3	Exemple d'un plan conditionnel pour un MDP jusqu'à $t = 2$ . . . . .	11
1.4	Le modèle de transition d'un POMDP dans lequel un agent dans un état $s$ a une action $a$ définie. Après l'exécution de cette action, il se retrouve dans un nouvel état $s'$ et une récompense $r(s, a)$ est alors obtenue de façon déterministe. . . . .	14
1.5	Interaction d'un agent d'un POMDP avec son environnement . . . . .	16
1.6	Exemple d'une politique pour un POMDP jusqu'à $t = 2$ . . . . .	16
1.7	La fonction de valeur $V_n$ définie dans espace d'état comprenant deux états $s_0$ et $s_1$ . $V_n$ est défini comme le maximum de l'ensemble des $\alpha$ -vecteur $\Gamma_n = \{\alpha_n^1, \alpha_n^2, \alpha_n^3\}$ . Pour l'état de croyance $b_n$ , $\alpha_n^2$ est la meilleure valeur. . . . .	18
1.8	$\alpha_1$ domine $\alpha_2$ , car la valeur de $\alpha_1$ est supérieur à $\alpha_2$ sur l'espace entier de l'état de croyance. $\alpha_2$ peut donc être négligé pour calculer la politique optimale. . . . .	19
1.9	Évolution de l'historique durant l'optimisation. . . . .	27
1.10	Modèle factorisé de la transition d'état du MOMDP proposé par [S. C. ONG et al. 2010] pour une action $a_t$ . . . . .	28

2.1	Représentation de la différence entre deux chemins, le chemin en bleu passant par la zone où le capteur est disponible réduisant ainsi l'incertitude sur la position et permettant de passer entre les deux obstacles. Le deuxième chemin en rouge ne passant pas par la zone où le capteur est disponible. Le chemin en rouge subit alors un accroissement de l'incertitude sur la position ne permettant pas de passer entre les deux obstacles.[BRY et al. 2011] . . . . .	33
2.2	Illustration de la dominance entre deux nœuds, le nœud vert pointillé est dominé par le nœud bleu, car il minimise la distance et l'incertitude. Afin de trouver la solution la plus sûre, la solution orange se doit d'être considérée, car elle est la seule dont l'incertitude sur la localisation (représentée par les cercles) ne rentre pas en collision.[SCHIRMER et al. 2017] . . . . .	36
2.3	Modèle d'un planificateur POMDP sur deux niveaux. . . . .	38
3.1	Représentation de la différence de la fréquence de fonctionnement entre le système GNC et le planificateur. . . . .	50
4.1	Exemple du calcul du PDOP du GPS pour un environnement donné. . . . .	52
4.2	Représentation de l'ensemble des cartes de probabilités sur la disponibilité, ainsi d'une carte de l'environnement. . . . .	53
4.3	Différence entre un des modèles des MOMDP classiques et le modèle du MOMDP proposé et utilisé dans ces travaux. . . . .	55
4.4	Illustration des deux fonctions de transitions : $T_{S_h}$ and $T_{S_v}$ . . . . .	58
4.5	Exemple de résultat de l'algorithme Expectation Maximization pour l'apprentissage d'un modèle de mélange gaussien pour approximer l'état de croyance $b'$ . . . . .	60
4.6	Exemple d'un couloir d'incertitude en deux dimensions entre deux états. . . . .	63
4.7	Exemple du calcul du coût de transition entre deux états de croyances représentés par un modèle de mélange gaussien. . . . .	64
4.8	Ensemble des vitesses de références. . . . .	72
4.9	Exemple de cartes proposées sur le benchmark de test ainsi que des exemples de cartes de probabilités de disponibilité du GPS. . . . .	73
4.10	Chemins simulés sur la carte avec deux obstacles « CubeBaffle ». . . . .	74
4.11	Chemins simulés sur la carte avec un obstacle « Cube ». . . . .	75
4.12	Représentation en 3D des chemins simulés sur la carte « CubeBaffle ». . . . .	76

5.1	Élagage de l'arbre de recherche. . . . .	85
5.2	Comparaison de l'évolution du nombre de nœud avec ou sans élagage. . . . .	85
5.3	Évolution de la fonction de valeur de la planification lors de l'optimisation (en haut) et lors des simulations (en bas) sur la carte <i>CubeBaffle</i> pour l'ensemble d'actions $A_-$ . . . . .	90
5.4	Résultats de la planification avec le taux de succès (en haut) et le temps du vol moyen (en bas) sur la carte <i>CubeBaffle</i> pour l'ensemble d'actions $A_-$ . . . . .	91
5.5	Exemples des chemins simulé à partir des différentes politiques calculé pour l'algorithme POMCP-GO (run 2) et POMCP (run 1). . . . .	91
5.6	Évolution de la fonction de valeur de la planification sur la carte <i>CubeBaffle</i> pour l'ensemble réduit d'actions et une pénalité $K = 450$ . . . . .	93
5.7	Résultats de la planification sur la carte <i>CubeBaffle</i> pour l'ensemble réduit d'actions et une pénalité $K = 450$ . . . . .	94
5.8	Exemples des chemins simulé à partir des différentes politiques calculé pour l'algorithme POMCP-GO et POMCP. . . . .	95
6.1	Représentation de $V(b_0)$ en fonction linéaire de la pénalité de collision $K$ . . . . .	101
6.2	Représentation des ensembles de directions $\mathcal{V}_{\text{ref}}$ utilisés. . . . .	103
6.3	Évolution de la fonction de valeur lors de la planification sur la carte <i>WallBaffle</i> pour l'ensemble réduit des actions $A^-$ et pour les pénalités de collision $K = 10^5$ et 450. . . . .	104
6.4	Résultats de la planification sur la carte <i>WallBaffle</i> pour l'ensemble réduit des actions et pour les pénalités de collision $K = 10^5$ et 450. . . . .	105
6.5	Exemples des chemins simulés à partir des différentes politiques. . . . .	106
6.6	Comparaison des différents résultats en fonction de l'ensemble d'actions utilisé et de la pénalité de collision. . . . .	107
6.7	Évolution de la fonction de valeur de la planification sur la carte <i>CubeBaffle</i> pour les deux ensembles d'actions, une pénalité de collision $K = 450$ et le coefficient d'exploration $c = 0.222 \times K$ . . . . .	109
6.8	Résultats de la planification sur la carte <i>CubeBaffle</i> pour les deux ensembles d'actions, une pénalité de collision $K = 450$ et le coefficient d'exploration $c = 0.222 \times K$ . . . . .	110
6.9	Exemples des chemins simulés à partir des différentes politiques. . . . .	111

6.10	Résultats de la planification sur la carte de San Diego pour $A^-$ et $A$ . . . . .	113
6.11	Exemples des chemins simulés à partir des différentes politiques. . . . .	114
6.12	Résultats de la planification sur la carte <i>WallBaffle</i> pour l'ensemble réduit des actions et différents taux de collisions. . . . .	117
6.13	Résultats de la planification sur la carte de San Diego pour les ensembles réduit des actions. . . . .	119
A.1	Évolution du coefficient variable en fonction de l'époque . . . . .	128
A.2	Évolution de la fonction de valeur lors de la planification sur la carte <i>WallBaffle</i> pour l'ensemble réduit des actions et pour les coefficients $c = 100$ , $c = 5$ et $c = c_{\text{var}}$ avec $K = 450$ . . . . .	129
A.3	Résultats de la planification sur la carte <i>WallBaffle</i> pour l'ensemble réduit des actions et pour les coefficients $c = 5$ et $c = c_{\text{var}}$ avec $K = 450$ . . . . .	130
A.4	Exemples des chemins simulé à partir des différentes politiques. . . . .	132
A.5	Comparaison des différents résultats en fonction du coefficient d'exploration utilisé. . . . .	133
B.1	Représentation de la carte utilisé pour les test, ainsi que la carte de probabilité de disponibilité du GPS. . . . .	135
B.2	Comparaison des différents résultats en fonction de la durée de l'époque pour l'ensemble d'action réduit $A^-$ . . . . .	137
C.1	Évolution du taux de succès et du temps de vol en fonction du nombre de simulations pour $K = 450$ avec l'ensemble réduit $A^-$ et l'ensemble complet d'actions $A$ . . . . .	140

# Liste des tableaux

4.1	Comparaison des performances entre les deux algorithmes proposés basés sur LRTDP et MCBTS. . . . .	74
-----	--	----





## Table des sigles et acronymes

<b>CBTS</b>	<i>Continuous Belief Tree Search</i>
<b>DOP</b>	Dilution de la Précision
<b>EKF</b>	Extended Kalman Filter
<b>GMM</b>	<i>Gaussian Mixture Model</i>
<b>GNC</b>	Guidage, Navigation et Contrôle
<b>GPS</b>	<i>Global Positioning System</i>
<b>IMU</b>	<i>Inertial Measurement Unit</i>
<b>LIDAR</b>	<i>Light detection and ranging</i>
<b>MCTS</b>	<i>Monte-Carlo Tree Search</i>
<b>MDP</b>	Processus Décisionnels de Markov
<b>MOMDP</b>	<i>Mixed Observability Markov Decision Process</i>
<b>PBVI</b>	<i>Point-Based Value Iteration</i>
<b>POMDP</b>	Processus de Décision Markoviens Partiellement Observables
<b>PWLC</b>	<i>Piecewise Linear and Convex</i>
<b>UAV</b>	Véhicule aérien inhabité
<b>RRBT</b>	<i>Rapidly-exploring Random Belief Trees</i>



# Introduction

À une époque où les projets industriels et de recherches sur les véhicules autonomes terrestres, maritimes et aériens se multiplient [EATON et al. 2017; ATAËI et al. 2015; DEVAURS et al. 2016], les besoins et leurs utilisations se diversifient. On constate notamment une demande croissante pour leurs utilisations dans des milieux encombrés et plus spécifiquement en milieu urbain [PADEN et al. 2016]. Les missions de ces véhicules peuvent consister à de la recherche de personnes pour des missions de sauvetages [WAHARTE et al. 2010], des missions de surveillance d'une ou plusieurs cibles [CHANEL et al. 2013; CAPITAN et al. 2016; VANEGAS et al. 2016] ou à du déplacement dans l'environnement, c'est-à-dire aller d'un point A à un point B [DOLGOV et al. 2010; PADEN et al. 2016], que ce soit pour du transport de biens ou de personnes.

La sûreté et l'efficacité de ces missions se doivent d'être assurées, notamment quand circuler ou naviguer dans une zone urbaine est encore un problème complexe pour les véhicules autonomes dû aux contraintes inhérentes à ces environnements. En effet, les zones urbaines sont des zones avec une haute densité d'obstacles (humains, véhicules, bâtiments, etc.) qui peuvent en cas de collision mettre en péril les personnes, engendrer des dégâts matériels ainsi que faire échouer la mission. En effet, un véhicule autonome est composé de capteurs embarqués et d'un système embarqué, qui comprend le système de guidage, le système de navigation et le système de contrôle (GNC) (Figure 1). La navigation est la partie du système dont le rôle est de permettre d'estimer l'état du véhicule (position, vitesse, etc.). Le guidage se sert de l'estimation d'état fournie par le système de navigation pour calculer les manœuvres à exécuter pour définir la trajectoire de référence pour une période future qui mènera à la réalisation d'une étape de la mission. Le contrôle se charge de calculer et donner la commande d'actionneurs à exécuter au véhicule pour réaliser la trajectoire de référence. En représentant de façon plus formelle, alors la boucle de contrôle du GNC ressemble à celle de la figure 1. Cependant, des écarts de trajectoires réalisées par rapport à la trajectoire de référence peuvent apparaître dus à des perturbations extérieures qui provoquent des erreurs d'exécution ou à l'incertitude de navigation.

La navigation estime l'état du véhicule en se basant sur les mesures des capteurs embarqués. C'est pourquoi la performance de « navigation » dépend fortement de la précision et de

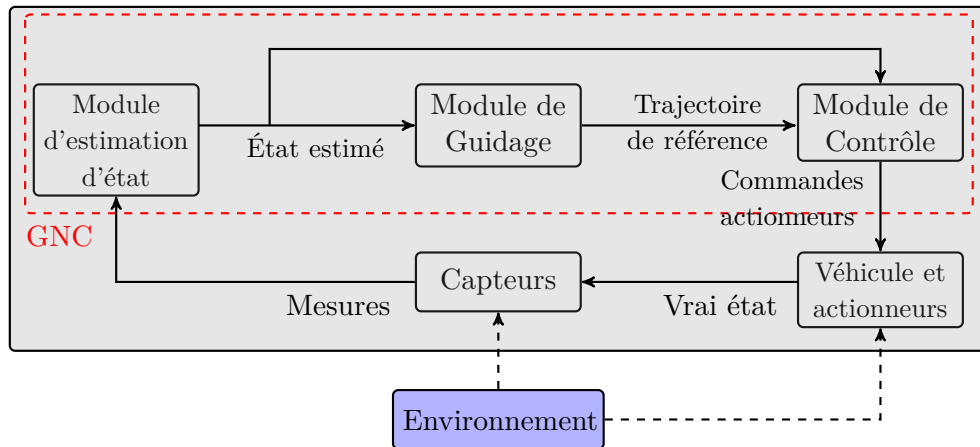


FIGURE 1 – Système GNC schématisé

la disponibilité de leurs capteurs embarqués qui peut varier selon l'environnement. En général, les véhicules autonomes en extérieur utilisent un capteur GPS (Global Positioning System) pour se localiser. En effet, la précision du GPS dépend de la visibilité de la constellation de satellites, qui dépend elle-même de la géolocalisation, du temps et de l'environnement. Les signaux GPS sont souvent masqués en milieu urbain et l'effet de multi-trajet (lorsqu'un signal se propage par plusieurs chemins) peut donner une mesure de localisation largement erronée. D'autre part, la vision embarquée peut être utilisée pour localiser des véhicules dans l'environnement quand il est difficile d'obtenir le GPS (environnement urbain, intérieur, espace) [CABALLERO et al. 2009; ARTIEDA et al. 2009], mais la disponibilité et la précision de ces capteurs sont aussi influencées par l'environnement tel que la texture des images, la visibilité des points d'intérêt, la luminosité, etc.

Fort heureusement, la performance de certains capteurs peut être prédite en constituant des connaissances a priori sur l'environnement. Par exemple, pour une constellation de satellites connue et pour une position donnée il est possible d'obtenir des informations sur la précision de la mesure du GPS sous la forme de Dilution de la Précision (DOP). La dilution de la précision est une métrique permettant d'estimer a priori la dégradation de la précision du GPS. Plus précisément, la PDOP (Position Dilution of Precision) correspond à l'écart-type de l'erreur sur la position calculé par un récepteur GPS. Cette DOP est obtenue en fonction d'un environnement en 3 dimensions (3D) et la géométrie de la constellation satellite qui est spécifiée par la géolocalisation et d'une fenêtre de temps [KLEIJER et al. 2009].

Par exemple, il est possible de calculer la DOP pour chaque point donné dans une carte 3D géoréférencée. La figure 2 montre un exemple de carte 3D ainsi que la carte de PDOP associée pour une altitude de  $0\text{m}^1$ . La carte de PDOP a été obtenue avec le simulateur GPS OKTAL. Ces informations pourraient être très utiles pour effectuer de la planification de chemin sûr, car cela permettrait au planificateur d'anticiper la possible dégradation de la précision capteurs et son influence sur l'incertitude de localisation et d'exécution le long du chemin et donc de minimiser le risque de collision [PRENTICE et al. 2010; WATANABE, DESSUS

1. ONERA/DEMR

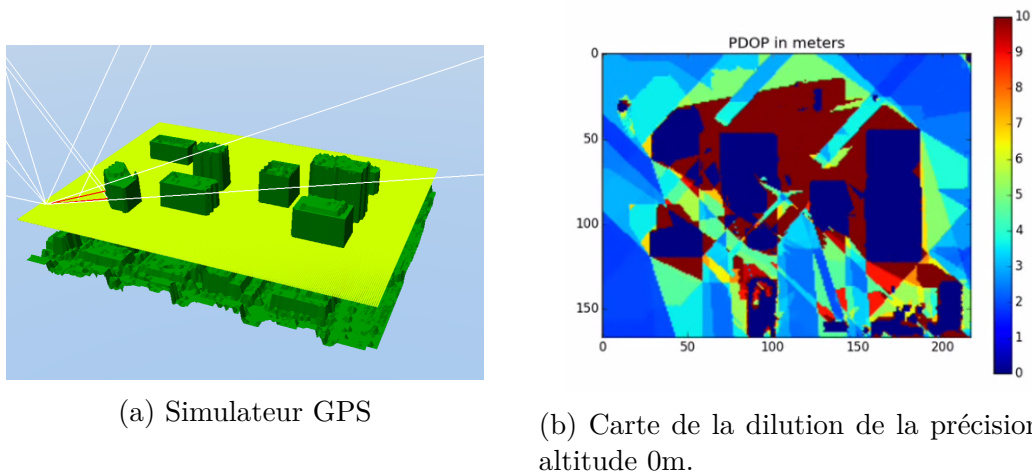


FIGURE 2 – Calcul de PDOP la carte de la ville de San Diego.

et al. 2014].

Dans ce contexte, cette thèse traite le problème de planification de chemins sûrs et efficaces sous incertitude en tenant compte de la disponibilité des capteurs qui dépendent de l'environnement. Des travaux précédents [PRENTICE et al. 2010 ; BRY et al. 2011 ; WATANABE, DESSUS et al. 2014 ; ACHELNIK et al. 2014] ont traité du problème de planification de chemin déterministe et discret en considérant l'incertitude sur la localisation du véhicule et en la propageant le long d'un chemin calculé en fonction de l'environnement. À titre d'exemple, [WATANABE, DESSUS et al. 2014] applique l'algorithme  $A^*$  et utilise la notion de couloir d'incertitude en tant que coût pour évaluer un chemin planifié afin de choisir le chemin le plus sûr et efficace. [BRY et al. 2011] et [ACHELIK et al. 2014] propagent l'incertitude sur la position pendant la recherche du chemin le plus court en utilisant l'algorithme Rapidly-exploring Random Belief Trees (RRBT). Cependant, aucune de ces approches ne considère le modèle complet de mouvement du véhicule en boucle fermée avec les fonctions du GNC (Guidage, Navigation, et Contrôle) dans le processus de décision tout en tenant compte de la disponibilité probable des capteurs.

Il est à noter que ce problème de planification de chemins est un problème de décision séquentielle dans l'incertain sous observabilité partielle. En effet, le véhicule autonome doit effectuer une suite d'actions afin de mener à bien sa mission avec une incertitude sur sa position dans un environnement lui-même incertain, car la disponibilité des capteurs est probabiliste. Typiquement, ce type de problème pourrait être abordé par l'optimisation de Processus de Décision Markoviens Partiellement Observable (POMDP) [SMALLWOOD et al. 1973]. Ce modèle a été utilisé sur des véhicules autonomes pour des problèmes similaires [S. C. W. ONG et al. 2009 ; PRENTICE et al. 2010 ; BRECHTEL et al. 2013 ; EATON et al. 2017 ; RICHTER et al. 2018]. Cependant, à notre connaissance peu de travaux [BERG et al. 2012] sur la planification de chemin de véhicule autonome utilisant des POMDP intègrent le modèle de transition GNC pour propager l'état de croyance en fonction de la disponibilité probable des capteurs.

En ce sens, cette thèse consiste à proposer un cadre de modélisation et de résolution pour des problèmes de planification de chemins minimisant le temps de vol, ou la distance parcourue, ainsi que le risque de collision, qui évoluent dans des environnements encombrés. Dans cette thèse, un des enjeux a été d'incorporer le module GNC, qui peut être considéré comme un système « bas niveau », dans un modèle POMDP donc de "haut niveau", afin de propager l'influence de la disponibilité des capteurs sur l'incertitude de la trajectoire exécutée. Un autre enjeu était de prendre en compte la disponibilité probable des capteurs embarqués lors de la planification. Comme la disponibilité d'un capteur tel que le GPS a un impact direct sur la précision de l'exécution de la trajectoire, il nous a semblé impératif de le prendre en compte directement dans la planification pour assurer la sécurité de la mission du véhicule.

Ces objectifs nous ont amenés dans une première partie à étudier l'état de l'art sur la modélisation et la résolution de POMDP ainsi que la résolution des problèmes de planification de chemins sous incertitude de localisation. Dans cet état de l'art, nous avons cherché à poser et expliquer les bases sur lesquelles nous nous sommes appuyés pour résoudre le problème traité dans cette thèse. Dans le chapitre 1, nous présenterons les POMDP et une de ses extensions appelée MOMDP (*Mixed Observability Markov Decision Process*) [S. C. ONG et al. 2010 ; ARAYA-LÓPEZ et al. 2010], ainsi que différents algorithmes de la littérature pour les résoudre. Enfin dans le chapitre 2, nous aborderons les autres méthodes proposées dans la littérature pour résoudre les problèmes de planifications de chemins sous incertitude de localisation.

La deuxième partie de cette thèse est consacrée à la présentation de nos contributions. Dans cette partie intitulée « Planification de chemins efficaces et sûrs par la prise en compte de la dégradation de la précision de la localisation et de la disponibilité des capteurs », nous proposons différentes approches pour la résolution de ce problème. Cette partie est divisée en quatre chapitres qui sont décrits succinctement ci-dessous.

Le chapitre 3 permet de modéliser le mouvement d'un véhicule autonome en bouclant avec le module GNC. Pour cela, nous avons tout d'abord modélisé le mouvement du véhicule (y compris le contrôle bas niveau) par un modèle cinématique translationnelle en définissant son état et sa fonction de transition. Puis nous avons utilisé cette modélisation pour concevoir le module de navigation en appliquant un filtre de Kalman. L'étape suivante a consisté à définir le module de guidage qui utilise l'état estimé de la navigation pour réaliser une action choisie. Ce modèle GNC sera intégré dans un planificateur que l'on a développé par la suite.

Dans le chapitre 4, nous proposons un planificateur générant des chemins en minimisant le volume du couloir de navigation qui dépend de la distance et de l'incertitude sur la position du véhicule. Pour cela, nous avons modélisé ce problème de planification en nous basant sur le formalisme MOMDP [S. C. ONG et al. 2010 ; ARAYA-LÓPEZ et al. 2010]. Les MOMDP sont une extension des POMDP qui permettent d'améliorer l'efficacité des calculs lors la résolution du problème en divisant l'espace des états en deux ensembles de variables, les variables partiellement observables et les variables totalement observables. Dans cette étude, nous avons adapté ce formalisme à notre problème en divisant notre espace d'état en deux sous-ensembles de variables, les variables continues non observables correspondant aux variables d'état du modèle GNC et les variables totalement observables correspondant entre autres à la disponi-

bilité des capteurs. Toutefois, la mise à jour d'états de croyance dans un MOMDP reste une étape complexe due à la modélisation qui considère des variables d'états continues et discrètes. C'est pourquoi nous avons approximé dans notre planificateur les états de croyances par des modèles de mélange gaussien. Pour cela, nous nous sommes appuyés sur le fait que le modèle de transition conserve l'incertitude sous forme de fonction gaussienne. La discrétisation en grilles, de la carte des obstacles et des cartes de disponibilité des capteurs, ne permet pas de garder une fonction gaussienne après mise à jour de l'état de croyance. Donc pour pouvoir garder la représentation d'un état de croyance par une fonction gaussienne ou un modèle de mélange gaussien, nous avons mis en place un algorithme d'apprentissage qui approxime un état de croyance en modèle de mélange gaussien à partir d'un ensemble de particules. Une fois que le modèle a été défini, nous avons proposé deux algorithmes pour résoudre le MOMDP. Le premier algorithme est basé sur Labeled RTDP [BONET et Hector GEFNER 2003] et RTDP-bel [BONET et Héctor GEFNER 2009] et le second algorithme est basé sur l'algorithme POMCP [BAI, HSU et al. 2010].

L'objectif étant de planifier un chemin sûr et efficace, nous proposons dans ce chapitre une fonction de coût exploitant la particularité de notre état de croyance pour créer une fonction de coût basée sur le couloir d'incertitude [WATANABE, DESSUS et al. 2014]. Ce couloir d'incertitude, pour un chemin donné, représente le tunnel comprenant l'incertitude sur l'erreur d'exécution du véhicule. Le volume total de ce tunnel représente ainsi l'incertitude sur la trajectoire après exécution, mais aussi sa longueur. Nous avons par la suite testé notre planificateur sur un problème « benchmark » pour la navigation de drone dans des environnements contenant des obstacles<sup>2</sup> [METTLER et al. 2010]. Ces problèmes benchmark proposent plusieurs cartes contenant différents types d'obstacles, et nous avons testé notre planificateur sur deux cartes simples contenant respectivement un et deux obstacles. Nous avons montré que le planificateur génère des politiques permettant des chemins sans risque de collision sur ces cartes simples. En revanche, ce planificateur a montré ses limites sur des cartes ayant une grande densité d'obstacle tel que San Diego (Figure 2). Ce nombre important d'obstacles engendre une grande variabilité de la disponibilité des capteurs et donc un calcul des probabilités de chaque observation plus complexe qui affecte le temps de calcul. Les travaux de ce chapitre ont donné lieu à deux communications [DELAMER et al. 2017a; DELAMER et al. 2017b].

Dans le chapitre 5, nous avons pris en compte les limites du précédent planificateur et nous l'avons changé afin de pallier certaines limites. Le premier objectif a été de changer la représentation des états de croyances qui affecte le temps de calcul des algorithmes, car elle nécessitait un calcul complexe de distribution de probabilité. Pour cela, nous avons représenté les états de croyances par des particules, chaque particule représentant un état possible de l'état de croyance. Nous avons fait le choix de ne plus baser la fonction de coût sur le couloir d'incertitude, mais de tout simplement la baser sur le temps de vol entre deux états et une pénalité en cas de collision. Nous avons ensuite appliqué ce planificateur sur deux cartes simples sur le benchmark ainsi que sur la carte de San Diego issu du même benchmark (Figure 2). Nous avons aussi proposé un nouvel algorithme de résolution POMCP-GO (POMCP Goal Oriented) basé sur POMCP et RTDP-Bel. Pour montrer l'intérêt de l'algorithme proposé,

---

2. Benchmark issue de : [www.aem.umn.edu/people/mettler/projects/AFDD/AFFDwebpage.htm](http://www.aem.umn.edu/people/mettler/projects/AFDD/AFFDwebpage.htm)

nous comparons les performances de POMCP-GO avec l'algorithme POMCP. Ces travaux ont donné lieu à deux communications. [DELAMER et al. 2018 ; DELAMER et al. 2019].

Dans le chapitre 6, l'objectif a été de proposer une méthode de définition de la fonction de coût qui permettrait à la politique calculée de respecter un taux de collision maximum autorisé tout en minimisant le temps de vol. Pour cela, nous avons redéfini la fonction de coût pour que la fonction de valeur du MOMDP devienne une fonction linéaire de la pénalité de collision avec une pente définît par la probabilité de collision. Il nous a été alors possible de choisir la valeur de la pénalité de collision à utiliser lors de l'optimisation afin de définir un taux de collision maximum acceptable spécifié par la mission. En effet, les algorithmes classiques de résolution des POMDP ne nous permettaient pas de résoudre le problème traité dans cette thèse dans un temps raisonnable. Nous donc proposé un nouvel algorithme POMCP-GO pour résoudre ces problèmes. Nous avons ensuite testé cette méthode sur un problème réduit pour vérifier la validité de la méthode, puis sur des problèmes plus réalistes.



**Partie I**  
**État de l'art**



# Processus décisionnels de Markov partiellement observable - POMDP

## Sommaire

---

1.1	Processus décisionnels de Markov . . . . .	10
1.2	<i>Stochastic Shortest-Path</i> MDP . . . . .	12
1.3	Processus Décisionnels de Markov Partiellement Observables . . . . .	13
1.3.1	État de croyance . . . . .	14
1.3.2	Politique et fonction de valeur . . . . .	15
1.3.3	Représentation de la fonction de valeur par des $\alpha$ -vecteur . . . . .	17
1.3.4	Itération exacte sur la valeur . . . . .	18
1.4	Extension du POMDP pour les espaces d'états et d'observations continus . . . . .	20
1.5	Méthodes approximatives d'itération sur la valeur . . . . .	21
1.5.1	Les algorithmes <i>point-based</i> . . . . .	21
1.5.2	RTDP-bel . . . . .	24
1.5.3	POMCP . . . . .	24
1.6	MOMDP - <i>Mixed Observability Markov Decision Process</i> . . . . .	27
1.7	Bilan . . . . .	29

---

Ce chapitre a pour objectif de présenter le formalisme des Processus décisionnels markoviens - MDP afin d'établir les notions de base utilisées par la suite. Nous présenterons ensuite les Processus décisionnels de Markov partiellement observable - POMDP qui sont une extension des MDP et qui permettent de modéliser des problèmes de décision séquentielle sous incertitude avec observabilité partielle. Les POMDP sont bien adaptés aux problèmes de planification en robotique où il est nécessaire de prendre des décisions malgré les incertitudes liées à l'environnement et l'incapacité à observer l'état du système. Le but d'un (PO)MDP est de trouver une stratégie permettant d'optimiser les décisions de l'agent pour accomplir la mission. Nous présenterons dans ce chapitre les modèles MDP et POMDP ainsi qu'une extension des POMDP, les Processus décisionnels de Markov sous observabilité mixte (MOMDP) qui sera appliquée à notre problème.

## 1.1 Processus décisionnels de Markov

Un Processus décisionnel de Markov (MDP) est un formalisme qui modélise des problèmes de décisions séquentiels pour lesquels un agent interagit avec un environnement dynamique et dont le résultat est incertain [BELLMAN 1957]. Dans ce formalisme, l'agent a une connaissance totale de son environnement et de son état, et seul le résultat de ses actions est incertain. Les MDP peuvent être considérés comme une extension des chaînes de Markov en étant contrôlés. Dans un MDP, l'agent influence le processus de décision à l'aide d'un ensemble d'actions pour lesquelles il reçoit des récompenses (ou des coûts). L'objectif de résolution souvent établi grâce à un critère mathématique sera alors de trouver la politique (séquence d'actions) qui maximise (minimise) ces récompenses (coûts) sur le long terme.

De façon formelle, un MDP à horizon infini est défini par le  $n$ -uplet  $(S, A, T, R, s_0)$  où :

- $S$  est l'ensemble fini des états discrets du système (appelé espace d'état) ;
- $A$  est l'ensemble fini des actions discrètes que l'agent peut réaliser ;
- $T : S \times A \times S \rightarrow [0, 1]$  est la fonction de transition d'état qui définit la probabilité  $T(s, a, s') = p(s' = s_{t+1} | s = s_t, a = a_t)$  d'arriver dans l'état  $s'$  après avoir exécuté l'action  $a \in A$  dans l'état  $s \in S$ . Les MDP étant des processus stochastiques dynamiques qui respectent la propriété de Markov d'ordre 1, le fait d'être dans l'état  $s'$  ne dépend donc que de l'état  $s$  et de l'action  $a$ . Ce modèle de transition peut être représenté par la figure 1.2.
- $R : S \times A \rightarrow \mathbb{R}$  est la fonction de récompense qui retourne la valeur  $R(s, a)$  quand le système exécute l'action  $a$  dans l'état  $s$ . Cette fonction indique à quel point cet état est souhaitable pour l'agent.
- $s_0$  est l'état initial du système.

La figure 1.1 représente l'interaction d'un agent avec son environnement. Du fait de l'incertitude sur le résultat des actions, l'agent ne peut qu'influencer l'évolution de son environnement et non le contrôler. Pour être robuste face à cette incertitude, il est nécessaire de mettre au point une méthode qui permet de choisir l'action à exécuter, et ceci pour tous les états possibles. Pour un MDP, il s'agit d'une politique  $\pi : S \rightarrow A$  qui est une fonction qui

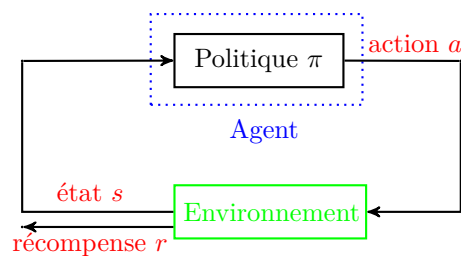


FIGURE 1.1 – Interaction d'un agent d'un MDP avec son environnement

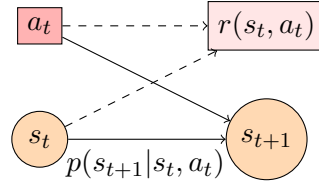


FIGURE 1.2 – Le modèle de transition d'un MDP dans lequel un agent dans un état  $s_t$  a une action  $a_t$  définie. Après l'exécution de cette action, il se retrouve dans un nouvel état  $s_{t+1}$  et une récompense  $r(s_t, a_t)$  est alors obtenue de façon déterministe.

donne pour chaque état  $s \in S$  l'action  $a \in A$  à réaliser. Pour un état initial  $s_0$  et un horizon infini, la politique peut être représentée par un arbre (plan conditionnel) dont la racine est l'état initial  $s_0$ . À chaque étape du plan, un nœud état contient l'action à effectuer (Figure 1.3).

L'objectif de l'optimisation d'un MDP est de trouver une politique  $\pi^*$  qui maximise la fonction de valeur  $V^\pi(s_0)$  pour un état initial  $s_0$ . La fonction de valeur  $V^\pi(s_0)$  peut être définie comme la récompense totale espérée décomptée par l'agent en suivant la politique  $\pi$  depuis l'état  $s_0$  :

$$V^\pi(s_0) = \mathbb{E} \left[ \sum_{t=0}^{\infty} \gamma^t r(s_t, \pi(s_t)) | s_0 \right] \quad (1.1)$$

où  $\gamma$  est le facteur d'actualisation dont la valeur est restreinte à  $0 \leq \gamma < 1$ .

La fonction de valeur optimale  $V^*$  est alors définie de la façon suivante :

$$V^{\pi^*}(s_0) = \mathbb{E} \left[ \sum_{t=0}^{\infty} \gamma^t r(s_t, \pi^*(s_t)) | s_0 \right] \quad (1.2)$$

où  $\pi^*$  est la politique optimale,

$$\pi^*(s_0) = \arg \max_{\pi \in \Pi} \mathbb{E} \left[ \sum_{t=0}^{\infty} \gamma^t r(s_t, \pi(s_t)) | s_0 \right] \quad (1.3)$$

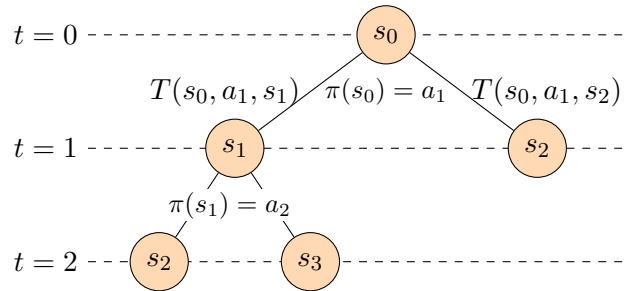


FIGURE 1.3 – Exemple d'un plan conditionnel pour un MDP jusqu'à  $t = 2$ .

## 1.2 *Stochastic Shortest-Path* MDP

Il est possible de dériver les processus décisionnels de Markov pour une sous-classe de problème qui est la planification du plus court chemin dans un environnement stochastique. Le MDP qui modélise et résout cette classe de problèmes est appelé *Stochastic Shortest-Path MDP* (SSP MDP) [BERTSEKAS 1995 ; KOLOBOV 2012].

L'idée est qu'une politique acceptable calculée par le SSP MDP va toujours mener l'agent à un état but, quel que soit l'état actuel de l'agent malgré l'incertitude. C'est pourquoi il est nécessaire de définir ce qu'est une politique acceptable pour SSP MDP.

### Définition 1.1

*Étant donné un ensemble  $G \subseteq S$  d'états buts  $s_g \in G$  pour lesquels toutes les actions  $a \in A$ ,  $T(s_g, a, s_g) = 1$  et  $R(s_g, a) = 0$ . Une politique  $\pi$  est dite acceptable en l'état  $s$  si  $\mathbb{E}[s_t \in G] = 1$ ,  $s_t$  étant l'état terminal de  $s$  en suivant la politique  $\pi$ . Alors la politique est dite acceptable si elle est acceptable pour tous les états  $s \in S$ . Autrement elle est dite non acceptable.*

La définition 1.1 implique qu'une politique acceptable ait un horizon défini. C'est pourquoi il est nécessaire de s'assurer qu'il existe une politique acceptable. Pour cela, il est possible de définir un coût pour chaque action au lieu d'une récompense afin que l'agent paie de plus en plus s'il n'atteint pas le but. Et s'il n'existe pas de politique acceptable (qui mène au but avec une probabilité de 1 pour chaque état  $s \in S$ ) alors le coût total espéré sera infini. En conséquence, toute politique acceptable sera choisie par rapport à n'importe quelle politique inacceptable.

Un SSP MDP est alors défini comme un tuple  $(S, A, T, C, G, s_0)$  :

- $S$  est l'ensemble fini des états discrets possibles du système (appelé espace d'état) ;
- $A$  est l'ensemble fini des actions discrètes que l'agent peut réaliser ;
- $T : S \times A \times S \rightarrow [0, 1]$  est la fonction de transition qui définit la probabilité  $T(s, a, s') = p(s' = s_{t+1} | s = s_t, a = a_t)$  d'arriver dans l'état  $s'$  après avoir exécuté l'action  $a \in A$  dans l'état  $s \in S$ .
- $C : S \times A \rightarrow [0, \infty)$  est la fonction de coût qui retourne la valeur  $C(s, a) > 0$  quand l'agent exécute l'action  $a$  dans l'état  $s$  et  $C(s, a) = 0$  si  $s \in G$ .
- $G \subseteq S$  est l'ensemble des états buts, tel que pour chaque état  $s_g \in G$ , pour toute action  $a \in A$  et pour tous les états  $s' \notin G$ , la transition  $T(s_g, a, s_g) = 1$  et  $T(s_g, a, s') = 0$
- $s_0$  est l'état initial du système.

Dans le cas d'un SSP MDP, l'objectif de l'optimisation est de trouver une politique  $\pi^*$  qui minimise la fonction de valeur  $V^\pi(s_0)$  pour un état initial  $s_0$ . La fonction de valeur est alors définie comme le coût total espéré par l'agent en suivant la politique  $\pi$  depuis l'état  $s_0$  :

$$V^\pi(s_0) = \mathbb{E} \left[ \sum_{t=0}^{\infty} C(s_t, \pi(s_t)) | s_0 \right] \quad (1.4)$$

La politique  $\pi^*$  d'un SSP MDP est alors définie par la fonction de valeur optimale  $V^{\pi^*}$  de la façon suivante :

$$\begin{aligned} V^{\pi^*}(s) &= \min_{a \in A} \left[ \sum_{s' \in S} T(s, a, s') [C(s, a) + V^{\pi^*}(s')] \right] \\ &= \min_{a \in A} \left[ C(s, a) + \sum_{s' \in S} p(s'|s, a) V^{\pi^*}(s') \right] \end{aligned} \quad (1.5)$$

Les plus grandes différences par rapport à l'optimisation d'un MDP classique sont la transformation d'un problème de maximisation en un problème de minimisation, le remplacement de la fonction de récompense par une fonction de coût et la suppression du coefficient d'actualisation  $\gamma$  qui est fixé à 1.

Il est à noter que les MDP et les SSP MDP considèrent que l'état du système est connu parfaitement par l'agent. Or dans le cas des véhicules autonomes, la connaissance sur l'environnement est partielle et imparfaite et la connaissance sur l'état est souvent incertaine.

### 1.3 Processus Décisionnels de Markov Partiellement Observables

Contrairement au MDP, les Processus Décisionnels de Markov Partiellement Observables (POMDP) sont un cadre formel mathématique qui permet de représenter des problèmes de décision séquentielle dans l'incertain sous observabilité partielle. Dans un POMDP, l'état du système est caché et a besoin d'être estimé. L'agent ne peut percevoir l'état que partiellement à l'aide des observations reçues de l'environnement.

Par rapport au modèle MDP, l'ensemble des observations  $\Omega$  possibles que l'agent peut recevoir de l'environnement est rajouté au modèle. Il est alors possible pour l'agent d'inférer sur l'état  $s \in S$  en utilisant une fonction d'observation qui donne la probabilité  $p(o|s, a)$  de recevoir l'observation  $o \in \Omega$  dans cet état. Le modèle de transition d'un POMDP est alors représenté par la figure 1.4.

Formellement, un POMDP est défini comme un  $n$ -uplet  $(S, A, \Omega, T, O, R, b_0)$  où :

- $S$  est l'ensemble fini des états possibles du système (appelé espace d'état) ;
- $A$  est l'ensemble fini des actions discrètes que l'agent peut réaliser ;
- $\Omega$  est l'ensemble fini des observations que l'agent peut recevoir ;
- $T : S \times A \times S \rightarrow [0, 1]$  est la fonction de transition d'état qui définit la probabilité  $T(s, a, s') = p(s' = s_{t+1} | s = s_t, a = a_t)$  d'arriver dans l'état  $s' \in S$  après avoir exécuté l'action  $a \in A$  dans l'état  $s \in S$ .
- $O : S \times A \times \Omega \rightarrow [0, 1]$  est la fonction d'observation qui définit la probabilité  $O(s', a, o) = p(o = o_{t+1} | s' = s_{t+1}, a = a_t)$  que l'agent reçoive l'observation  $o$  dans l'état  $s'$  après avoir effectué l'action  $a$  ;
- $R : S \times A \rightarrow \mathbb{R}$  est la fonction de récompense qui retourne la valeur  $R(s, a)$  quand le système exécute l'action  $a$  dans l'état  $s$ . Cette fonction indique à quel point cet état est souhaitable pour l'agent.
- $b_0$  : est la distribution initiale de probabilité sur les états possibles, aussi appelé *état de croyance*.

### 1.3.1 État de croyance

Contrairement à un MDP, un agent ne connaît pas exactement l'état du système. L'agent doit donc utiliser les observations pour inférer sur cet état. Cependant, une ou plusieurs observations peuvent ne pas suffire à connaître l'état actuel avec certitude. Il est donc nécessaire pour l'agent de prendre en compte tout l'historique d'actions-observations à un instant  $t$  afin d'obtenir un état d'information complet que nous définissons tels que :

$$h_t = \{a_0, o_1, \dots, o_{t-1}, a_{t-1}, o_t\} \quad (1.6)$$

Un agent dans un POMDP peut à chaque instant décider d'exécuter une action parmi  $|A|$  actions. Après avoir exécuté l'action, l'état du système aura changé et l'agent recevra alors

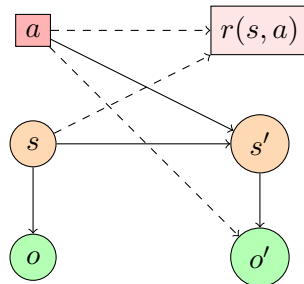


FIGURE 1.4 – Le modèle de transition d'un POMDP dans lequel un agent dans un état  $s$  a une action  $a$  définie. Après l'exécution de cette action, il se retrouve dans un nouvel état  $s'$  et une récompense  $r(s, a)$  est alors obtenue de façon déterministe.



pour ce nouvel état une observation  $o$  parmi  $|\Omega|$  observations. Le nombre possible d'historiques est donc exponentiel en taille  $t$  de l'historique  $(|A| \times |\Omega|)^t$ . Il est donc coûteux pour des raisons de taille en mémoire de travailler sur des historiques.

Cependant, il a été montré qu'il n'est pas nécessaire de travailler directement avec l'historique tel que définit précédemment (Eq. 1.6). L'agent peut estimer son état en fusionnant le dernier couple action-observation avec sa connaissance antérieure sur l'état. Cette estimation de l'état appelé *état de croyance* est donnée sous la forme d'une distribution de probabilité sur les états possibles, et l'ensemble de ces états de croyances est noté  $B$ . Il est prouvé qu'un état de croyance sous forme de distribution de probabilité apporte autant d'information que l'utilisation de l'historique complet [SMALLWOOD et al. 1973; BUFFET et al. 2008]. Cet état de croyance ou distribution de probabilité à un instant  $t$  est noté  $b_t$  et la probabilité sur l'état  $s$  est notée  $b_t(s)$  tel que :

$$b_t \in B, \quad b_t(s) = p(s_t = s | h_t) \quad \forall s \in S \quad \text{et} \quad \sum_{s \in S} b_t(s) = 1 \quad (1.7)$$

L'état de croyance  $b_o^a$  obtenu après avoir exécuté une action  $a$  dans l'état de croyance  $b$  et reçu une observation  $o$  peut être calculé en utilisant la règle de Bayes et cela au moyen de la fonction de transition et de la fonction d'observation. Nous obtenons alors :

$$\begin{aligned} b_o^a(s') &= p(s' | b, a, o) = \frac{p(o, s' | b, a)}{p(o | b, a)} = \frac{p(o | s', a) p(s' | b, a)}{p(o | b, a)} \\ &= \frac{p(o | s', a) p(s' | b, a)}{\sum_{s' \in S} \sum_{s \in S} p(o | b, a, s, s') p(s, s' | b, a)} \\ &= \frac{p(o | s', a) \sum_{s \in S} p(s' | s, a) b(s)}{\sum_{s' \in S} p(o | s', a) \sum_{s \in S} p(s' | s, a) b(s)} \end{aligned} \quad (1.8)$$

Avec cette fonction de mise à jour, les états de croyance forment toujours un processus Markovien, car l'état de croyance à l'instant  $t + 1$  dépend seulement de l'état de croyance à l'instant  $t$ , de l'action à l'instant  $t$  et de l'observation à l'instant  $t + 1$ . La figure 1.5 représente comment l'agent interagit avec son environnement en fonction de son état de croyance.

### 1.3.2 Politique et fonction de valeur

Comme pour un MDP, l'objectif de l'optimisation d'un POMDP est de trouver une politique optimale  $\pi^*$  qui maximise la fonction de valeur  $V^\pi$ . Cependant, un agent dans un POMDP ne connaît pas l'état initial du système et donc nous ne pouvons pas définir la politique et la fonction de valeur sur les états comme pour un MDP. Une politique Markovienne déterministe  $\pi(b)$  est une fonction  $\pi(b) : B \mapsto A$  qui assigne une action  $a$  pour chaque état

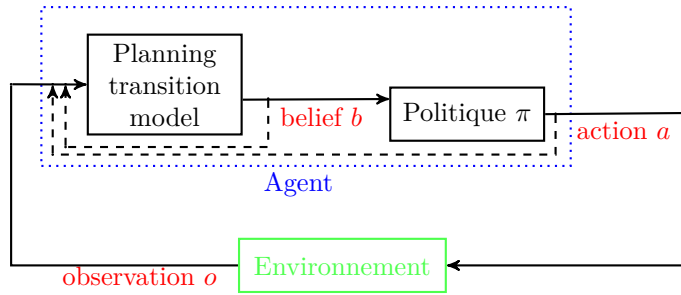
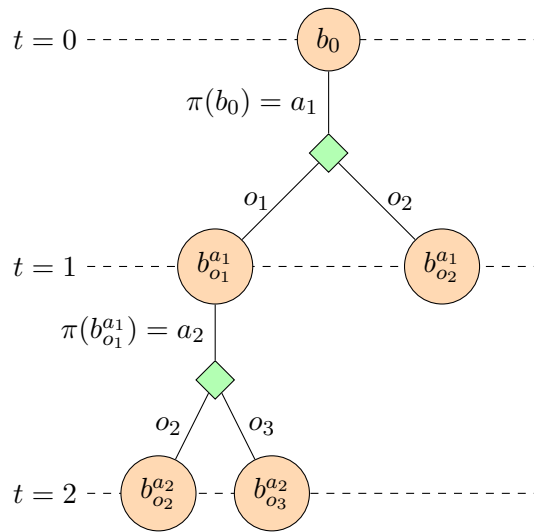


FIGURE 1.5 – Interaction d'un agent d'un POMDP avec son environnement

FIGURE 1.6 – Exemple d'une politique pour un POMDP jusqu'à  $t = 2$ .

de croyance [BUFFET et al. 2008]. Pour un état d'information complet initial  $h$ , la politique optimale d'horizon  $t$  peut être représentée par un arbre qui correspond à une sorte de plan conditionnel. Par exemple la figure 1.6 représente un arbre à horizon  $t = 2$  obtenu depuis l'état de croyance initiale  $b_0$ .

La fonction de valeur  $V^\pi(b_0) : B \rightarrow \mathbb{R}$  d'un POMDP peut alors être définie comme la récompense totale espérée par l'agent en suivant la politique  $\pi$  depuis l'état de croyance  $b_0$ . La fonction de valeur devient alors une fonction qui réalise une projection d'un état de croyance sur une valeur dans  $\mathbb{R}$  tel que :

$$V^\pi(b_0) = \mathbb{E} \left[ \sum_{t=0}^{\infty} \gamma^t r(b_t, \pi(b_t)) | b_0 \right] \quad (1.9)$$

où  $0 \leq \gamma < 1$  est le facteur d'actualisation qui assure la convergence en horizon infini [SIGAUD et al. 2008] et :

$$r(b_t, \pi(b_t)) = \sum_{s \in S} r(s, \pi(b_t)) b_t(s) = \mathbb{E} [r(s, \pi(b_t)) | b_t] \quad (1.10)$$

La politique optimale  $\pi^*$  d'un POMDP est alors définie par la fonction de valeur optimale  $V^{\pi^*}$  de la façon suivante :

$$V^{\pi^*}(b) = \max_{a \in A} \left[ \sum_{s \in S} r(s, a) b(s) + \gamma \sum_{o \in \Omega} p(o|b, a) V^{\pi^*}(b_o^a) \right] \quad (1.11)$$

La politique est garantie optimale quand (Éq. 1.11) converge pour tout  $b$  [SIGAUD et al. 2008], mais en pratique résoudre un POMDP est difficile [PAPADIMITRIOU et al. 1987]. Résoudre les cas les plus classiques des POMDP discrets est même indécidable [MADANI et al. 1999]. [Michael L LITTMAN et al. 1995] ont prouvé que pour les POMDP à horizon infini avec des récompenses booléennes est EXPTIME-hard. C'est pourquoi résoudre des POMDP c'est surmonter à la fois la malédiction de la dimension [KAELBLING et al. 1998] et celle de l'historique [PINEAU et al. 2003].

### 1.3.3 Représentation de la fonction de valeur par des $\alpha$ -vecteurs

La fonction de valeur d'un POMDP à horizon fini est linéaire par morceaux et convexe (PWLC) [SMALLWOOD et al. 1973] et à horizon infini elle peut aussi être approchée par une fonction de valeur PWLC. En effet, la fonction de valeur d'un  $b$  donné à une étape  $n$  peut être représentée par un ensemble  $\Gamma_n$  de vecteurs appelé  $\alpha$ -vecteurs [KAELBLING et al. 1998] :

$$V_n(b) = \max_{\alpha_n^i \in \Gamma_n} \sum_{s \in S} b(s) \alpha_n^i(s) = \max_{\alpha_n^i \in \Gamma_n} \langle b, \alpha_n^i \rangle \quad (1.12)$$

où  $\langle \cdot, \cdot \rangle$  représente le produit scalaire. La convexité et linéarité par morceaux de la fonction de valeur découle du fait que  $V_n$  réalise le maximum des valeurs des actions (Éq. 1.11), qui sont individuellement linéaires sur l'espace des états de croyances (Éq. 1.10). C'est pourquoi chaque  $\alpha$ -vecteur définit une région de l'espace d'état de croyance pour lequel il maximise la valeur de  $V_n$  (Figure 1.7). La convexité de la fonction de valeur est justifiable intuitivement. Les états de croyances situés aux extrémités signifient que l'agent est certain qu'il est dans l'état correspondant et il peut donc maximiser directement l'action à exécuter. Dans un autre état de croyance,  $b$  par exemple (Figure 1.7) qui est proche d'une distribution uniforme ( $b = [0.6 \ 0.4]$ ), l'agent est très incertain de l'état réel et donc l'action qui maximise a une récompense espérée plus basse.

Un  $\alpha$ -vecteur est une projection de la valeur espérée d'une action sur l'ensemble de l'espace des états de croyances. Pour obtenir l' $\alpha$ -vecteur à l'étape  $n$  pour l'action  $a \in A$ , la projection a besoin d'être calculée pour tous les états  $s \in S$  de la façon suivante :

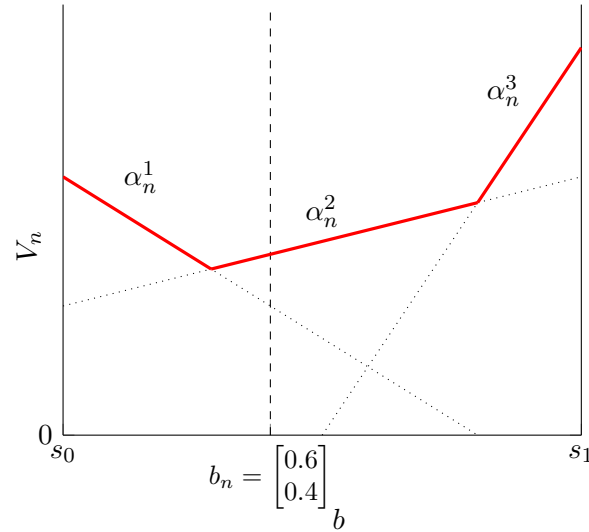


FIGURE 1.7 – La fonction de valeur  $V_n$  définie dans espace d'état comprenant deux états  $s_0$  et  $s_1$ .  $V_n$  est défini comme le maximum de l'ensemble des  $\alpha$ -vecteurs  $\Gamma_n = \{\alpha_n^1, \alpha_n^2, \alpha_n^3\}$ . Pour l'état de croyance  $b_n$ ,  $\alpha_n^2$  est la meilleure valeur.

---

**Algorithme 1** : Algorithme d'itération de la valeur pour les POMDP

---

```

1  $\Gamma \leftarrow \Gamma_0$  (L'ensemble initial d' $\alpha$ -vecteurs)
2  $n \leftarrow 1$ 
3 while  $n \leq N$  do
4   for  $a \in A$  do
5     for  $s \in S$  do
6        $\alpha_n(s) \leftarrow r(s, a) + \gamma \sum_{o \in \Omega} \sum_{s' \in S} p(o|s', a)p(s'|s, a)\alpha_{n-1}(s')$ 
7      $\Gamma' \leftarrow \{\Gamma', \alpha\}$ 
8    $\Gamma \leftarrow \Gamma'$ 
9    $n++$ 
10 return  $\Gamma$ 

```

---

$$\alpha_{n+1}^a(s) = r(s, a) + \gamma \sum_{o \in \Omega} \sum_{s' \in S} p(o|s', a)p(s'|s, a)\alpha_n(s'), \forall \alpha_n \in \Gamma_n \quad (1.13)$$

### 1.3.4 Itération exacte sur la valeur

L'espace des états de croyance dans un POMDP est continu, il existe donc un nombre infini d'états de croyance. Cependant, à la  $n$ -ième étape la fonction de valeur peut être représentée comme un ensemble fini  $\Gamma_n$  d' $\alpha$ -vecteurs (Équation 1.12). [SMALLWOOD et al. 1973] propose un algorithme (Alg. 1) pour calculer toutes les projections d' $\alpha$ -vecteurs possibles.

Cet algorithme permet de calculer la fonction de valeur sur l'espace entier des états de croyances. Malheureusement à chaque étape un nombre exponentiel en  $|\Omega|$  est généré :

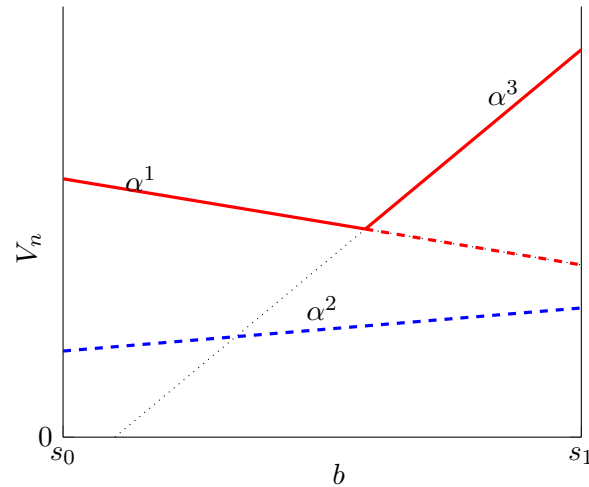


FIGURE 1.8 –  $\alpha_1$  domine  $\alpha_2$ , car la valeur de  $\alpha_1$  est supérieure à  $\alpha_2$  sur l'espace entier de l'état de croyance.  $\alpha_2$  peut donc être négligé pour calculer la politique optimale.

$|\Gamma_{n+1}| = |A||\Gamma_n|^{|\Omega|}$ . Il est à noter que de nombreux vecteurs seront inutiles, car dominés par d'autres sur tout l'espace d'état de croyance. Il est dit d'un vecteur  $\alpha_1$  qu'il domine un vecteur  $\alpha_2$  s'il contient des valeurs supérieures pour chaque point dans l'espace de croyance selon :

$$\forall b \in B, \quad \langle b, \alpha_1 \rangle \geq \langle b, \alpha_2 \rangle \quad (1.14)$$

Les vecteurs dominés ne contiennent donc aucune information utile pour calculer la valeur optimale, car il n'y a aucune situation dans laquelle ces vecteurs maximisent la valeur d'une partie de l'espace de croyance (Figure 1.8). Beaucoup de méthodes d'itération exacte sur les valeurs cherchent à trouver un ensemble d' $\alpha$ -vecteurs réduit. Certaines de ces méthodes cherchent à élaguer les  $\alpha$ -vecteurs dominés comme l'algorithme de Monahan [MONAHAN 1982] et le filtrage de Lark [WHITE 1991]. Malgré toutes ces recherches, l'itération exacte sur les valeurs pour un POMDP reste seulement applicable pour des petits problèmes. L'élagage exige la résolution d'un programme linéaire pour chaque  $\alpha$ -vecteur et est coûteux [BUFFET et al. 2008] puisqu'il ajoute des traitements supplémentaires. Cependant, « l'élagage incrémental » est plus efficace, car il sépare les vecteurs plus tôt dans l'optimisation [ZHANG et al. 1996; CASSANDRA et al. 1997]. De même l'algorithme « One-Pass » de Sondik [SMALLWOOD et al. 1973] et l'algorithme « Witness » [Michael Lederman LITTMAN 1996; KAEHLING et al. 1998] visent à construire directement cet ensemble restreint d' $\alpha$ -vecteurs.

En général, calculer une solution exacte pour un POMDP devient impossible pour des problèmes réalistes. Pour pallier ce problème, il est possible d'utiliser des méthodes approximatives d'itération sur la valeur afin d'approcher la fonction de valeur optimale.

## 1.4 Extension du POMDP pour les espaces d'états et d'observations continus

Certains problèmes peuvent être modélisés par des états et des observations continus notamment dans les problèmes de navigation. [PORTA et al. 2006] propose une adaptation des POMDP pour prendre en compte des états et observations continus.

Dans un POMDP un agent doit mettre à jour son état de croyance en fonction de l'observation  $o$  reçue après avoir exécuté l'action  $a$ . Nous avons vu comment mettre à jour l'état de croyance pour des actions discrètes (Éq. 1.8).

Dans le cas des états continus la propagation  $p(s'|b, a)$  de l'état de croyance  $b$  avec une action  $a$  est défini de la façon suivante :

$$p(s'|b, a) = \int_{s \in S} p(s'|s, a)b(s)ds \quad (1.15)$$

La mise à jour de l'état de croyance se définit alors comme :

$$\begin{aligned} b_o^a(s') &= p(s'|b, a, o) = \frac{p(o, s'|b, a)}{p(o|b, a)} = \frac{p(o|s', a)p(s'|b, a)}{p(o|b, a)} \\ &= \frac{p(o|s', a)p(s'|b, a)}{\int_{s' \in S} \int_{s \in S} p(o|b, a, s, s')p(s, s'|b, a)dsds'} \\ &= \frac{p(o|s', a) \int_{s \in S} p(s'|s, a)b(s)ds}{\int_{s' \in S} p(o|s', a) \int_{s \in S} p(s'|s, a)b(s)dsds'} \end{aligned} \quad (1.16)$$

Nous pouvons alors définir la politique optimale  $\pi^*$  par la fonction de valeur optimale  $V^{\pi^*}$  de la façon suivante :

$$V^{\pi^*}(b) = \max_{a \in A} \left[ \int_{s \in S} r(s, a)b(s)ds + \gamma \int_{o \in \Omega} p(o|b, a)V^{\pi^*}(b_o^a)do \right] \quad (1.17)$$

[PORTA et al. 2006] a montré qu'il est toujours possible de représenter la fonction de valeur par un ensemble continu d'«  $\alpha$ -fonction » (Éq. 1.12).

Cependant, même s'il est possible de continuer à représenter la fonction de valeur par un ensemble d' $\alpha$ -vecteurs, la résolution devient encore plus difficile.

## 1.5 Méthodes approximatives d'itération sur la valeur

L'impossibilité de calculer des solutions exactes pour des problèmes réalistes a longtemps été un frein pour l'application des POMDP. Dans les faits, les approximations de la politique optimale peuvent être meilleures que des solutions qui ne prennent pas en compte l'incertitude et l'observabilité partielle. En effet, les algorithmes d'approximation s'attaquent soit au fléau de la dimension, soit au fléau de l'historique ou les deux.

### 1.5.1 Les algorithmes *point-based*

Certains algorithmes de résolution de POMDP sont dits *point-based* parce qu'ils maintiennent un ensemble fini d'états de croyance atteignable  $\Delta$  qui est un sous-espace de l'ensemble des états de croyances, tel que  $\Delta \subset B$ . Cela permet de travailler sur un ensemble d'états de croyance plus restreint et ainsi générer moins de  $\alpha$ -vecteurs et donc de garder sous une taille raisonnable la représentation de la fonction de valeur. Ce type de méthode met à jour à chaque itération non seulement les valeurs, mais aussi les  $\alpha$ -vecteurs seulement pour ces points (état de croyances) i.e.  $\forall b \in \Delta$ . Cette mise à jour aussi appelée *backup* en un point  $b$  est le remplacement de son  $\alpha$ -vecteur  $\alpha$  par un nouvel  $\alpha$ -vecteur (Éq. 1.13) qui maximise l'équation (1.12) pour  $b$  :

$$\alpha' = \arg \max_{\alpha \in \Gamma} \langle b, \alpha \rangle \quad (1.18)$$

Cependant, cette mise à jour requiert l'ensemble  $\Gamma$  (calculé avec l'Équation 1.13) génère  $\mathcal{O}(|A||\Gamma|^{|\Omega|})$  nouveaux  $\alpha$ -vecteurs, si l'ensemble précédent était de taille  $|\Gamma|$ . Le nombre d' $\alpha$ -vecteurs est exponentiel,  $\mathcal{O}(|A||\Gamma|^{|\Omega|^t})$  à l'horizon  $t$ . Il est possible de calculer le *backup* d'un vecteur en temps polynomial [PINEAU et al. 2003]. Tout d'abord, les projections  $a, o$  peuvent être calculées comme suit :

$$\begin{aligned} \sum_{o \in \Omega} p(o|b, a) V(b_o^a) &= \sum_{o \in \Omega} p(o|b, a) \left[ \max_{\alpha \in \Gamma} \sum_{s \in S} b_o^a(s) \alpha(s) \right] \\ &= \sum_{o \in \Omega} p(o|b, a) \left[ \max_{\alpha \in \Gamma} \frac{\sum_{s' \in S} p(o|s', a) \sum_{s \in S} p(s'|a, s) b(s) \alpha(s')}{p(o|b, a)} \right] \\ &= \sum_{o \in \Omega} \left[ \max_{\alpha \in \Gamma} \sum_{s' \in S} p(o|s', a) \sum_{s \in S} p(s'|a, s) b(s) \alpha(s') \right] \\ &= \sum_{o \in \Omega} \max_{\alpha \in \Gamma} \sum_{s' \in S} g_o^{a, \alpha}(s') b(s) \end{aligned} \quad (1.19)$$

où  $g_o^{a, \alpha}$  est le vecteur défini comme :

$$g_o^{a, \alpha}(s') = p(o|s', a) \sum_{s \in S} p(s'|a, s) \alpha(s') \quad (1.20)$$

Il est possible de réécrire la somme intérieure sous forme d'un produit scalaire :

$$\max_{\alpha \in \Gamma} \langle g_o^{a,\alpha}, b \rangle = \langle \arg \max_{g_o^{a,\alpha}, \alpha \in \Gamma} \langle g_o^{a,\alpha}, b \rangle, b \rangle \quad (1.21)$$

Nous pouvons obtenir :

$$\begin{aligned} \sum_{o \in \Omega} p(o|b, a) V(b_o^a) &= \sum_{o \in \Omega} \left[ \max_{\alpha \in \Gamma} \langle g_o^{a,\alpha}, b \rangle \right] \\ &= \sum_{o \in \Omega} \langle \arg \max_{g_o^{a,\alpha}, \alpha \in \Gamma} \langle g_o^{a,\alpha}, b \rangle, b \rangle \\ &= \left\langle \sum_{o \in \Omega} \arg \max_{g_o^{a,\alpha}, \alpha \in \Gamma} \langle g_o^{a,\alpha}, b \rangle, b \right\rangle \end{aligned} \quad (1.22)$$

où la somme dans l'équation (1.22) est une somme de vecteur. Finalement, il est possible de redéfinir l'Équation (1.13) comme un vecteur de telle sorte que chaque composante de ce vecteur est :

$$g_b^a(s) = r(s, a) + \gamma \sum_{o \in \Omega} g_{o,b}^a(s) \quad (1.23)$$

où le vecteur  $g_{o,b}^a$  est le vecteur qui maximise le produit scalaire  $\langle g_o^{a,\alpha}, b \rangle$  pour tous les  $\alpha$ -vecteur  $\alpha \in \Gamma$ . Alors la valeur  $V$  mise à jour pour l'état de croyance  $b$  devient :

$$\begin{aligned} V(b) &= \max_{a \in A} r(b, a) + \gamma \left\langle \sum_{o \in \Omega} \arg \max_{g_o^{a,\alpha}, \alpha \in \Gamma} \langle g_o^{a,\alpha}, b \rangle, b \right\rangle \\ &= \max_{a \in A} \langle g_b^a, b \rangle \end{aligned} \quad (1.24)$$

on définit alors l'opérateur de backup par :

$$\text{backup}(b) = \arg \max_{g_b^a, a \in A} \langle g_b^a, b \rangle \quad (1.25)$$

Le résultat de cet opérateur est l' $\alpha$ -vecteur qui maximise la récompense pour l'état de croyance  $b$  et qui vient se rajouter à l'ensemble  $\Gamma$  qui représente la fonction de valeur. Cette méthode permet de calculer plus efficacement le vecteur qui met à jour la fonction de valeur  $V$  dans l'état de croyance  $b$ . En effet, il y a  $|A|$  vecteurs  $g_b^a$  qui peuvent être calculés en  $\mathcal{O}(|S|)$  si tous les vecteurs  $g_o^{a,\alpha}$  ont été calculés, une opération qui peut être réalisée en temps  $\mathcal{O}(|S|^2|\Gamma||\Omega||A|)$ . De plus, les vecteurs  $g_o^{a,\alpha}$  ne dépendent pas de l'état de croyance  $b$  et donc une fois qu'il est calculé ce vecteur peut être réutilisé pour le calcul des autres états de croyances.

Le premier algorithme *point-based* était PBVI (*Point-Based Value Iteration*) [PINEAU et al. 2003 ; SHANI, PINEAU et al. 2013] montré dans l'algorithme 2. PBVI commence par initialiser la valeur avec l'état de croyance (point) initial  $b_0$ . Tout le temps que l'algorithme n'a pas convergé ou que le nombre maximal d'itérations n'est pas atteint, l'algorithme étend



**Algorithme 2** : PBVI - *Point-Based Value Iteration*


---

```

1  $V_0 \leftarrow \alpha^a(s), \forall a \in A$ 
2  $i \leftarrow 0$ 
3 tant que  $i \leq N$  faire
4    $i++$ 
5   pour chaque  $b \in \Delta$  faire
6      $\Delta' \leftarrow \emptyset$ 
7      $s \leftarrow b$ 
8     pour chaque  $a \in A$  faire
9        $s' \leftarrow p(s'|s, a)$ 
10       $o \leftarrow p(o|s', a)$ 
11       $\Delta' \leftarrow b_a^o(s')$ 
12      $\Delta \leftarrow \arg \max_{b' \in \Delta'} \|b' - b\|, \forall b \in \Delta$ 
13    $V_n \leftarrow \emptyset$ 
14    $\Gamma^{a,o} \leftarrow \alpha_i^{a,o}, \forall \alpha_i \in V_n$ 
15   pour chaque  $b \in \Delta$  faire
16      $V_n \leftarrow \bigcup_{b \in \Delta} \text{backup}(b)$  (Éq. 1.12)
17 retourner  $V$ 

```

---

les états de croyances atteignables (Lignes 5-12). PBVI calcule toutes les projections des  $V_{t-1}$  (Ligne 14) pour mettre à jour la valeur pour tout  $b \in B$  (Ligne 16). Une fois le nombre d'itérations maximales atteint, PBVI retourne la fonction de valeur approchée.

D'autres algorithmes tels que PERSEUS [PORTA et al. 2006] fonctionnent aussi sur le même principe. Cependant, une des différences est que tout d'abord PERSEUS explore aléatoirement un ensemble d'états de croyance atteignable. C'est sur cet ensemble de points que l'algorithme calculera la fonction de valeur. De plus, un autre algorithme HSVI [SMITH et al. 2004] améliore les performances en temps de calcul et en récompenses. HSVI maintient une borne inférieure  $\underline{V}(b)$  et une borne supérieure  $\bar{V}(b)$  de la fonction de valeur pour chaque état de croyances visité. L'algorithme visite l'espace des états de croyances en sélectionnant à chaque étape l'action la plus prometteuse  $a^*$  et l'observation pour laquelle  $p(o|b, a^*)(\underline{V}(b) - \bar{V}(b))$  est maximale. Le critère d'arrêt de la recherche d'HSVI est quand  $\underline{V}(b) - \bar{V}(b) < \epsilon\gamma^{-t}$ . SARSOP [KURNIAWATI, HSU et al. 2008] est un algorithme basé sur HSVI qui reprend lui aussi la même stratégie de sélection d'action pour explorer les états de croyances. La différence majeure entre ces deux algorithmes vient de la condition d'arrêt de la recherche. SARSOP propage la borne inférieure et une valeur approchée de la borne supérieure afin de définir le critère d'arrêt suivant :  $\left[ (\hat{V} \leq L) \wedge (\bar{V}(b) \leq \max\{U, \underline{V}(b) + \epsilon\gamma^{-t}\}) \right]$  où  $\hat{V}$  est la valeur optimale prédite de  $b$ ,  $L$  et  $U$  représentent les bornes supérieures et inférieures propagées, respectivement. Ce critère permet de rechercher des points (états de croyances) moins profonds que HSVI, ce qui devrait permettre d'orienter la recherche vers des états de croyances appartenant à l'ensemble des états de croyances optimaux, mais aucune preuve théorique n'est fournie.

### 1.5.2 RTDP-bel

L'algorithme RTDP-bel (Alg. 3) [Hector GEFNER et al. 1998 ; BONET et Héctor GEFNER 2009] est une adaptation pour les POMDP de l'algorithme RTDP [BARO et al. 1995] conçu initialement pour la résolution de SSP MDP. Cet algorithme fait partie des méthodes de type *trial-based* qui sont basées sur des simulations de trajectoires dans l'espace d'états de croyances. Une fois qu'une trajectoire est simulée, la mise à jour s'effectue depuis l'état de croyance le plus profond.

RTDP-bel considère à l'instar des SSP MDP, des POMDP orientés « but » pour lesquels le modèle de récompense est transformé en modèle de coûts. En effet, RTDP suppose qu'il existe un état but qui est atteignable avec une probabilité de 1 après  $t$  itérations pour chaque état initial possible. RTDP-bel fonctionne sur le même principe et considère que pour un état initial, il existe une trajectoire qui mène vers un état de croyance « but ». RTDP-bel discrétise les états de croyances et se sert de cette discrétisation pour maintenir une  $Q$ -valeur sur ces états de croyances. Une  $Q$ -valeur d'un état de croyance  $b$  et d'une action  $a$  noté  $Q(b, a)$  est la valeur à une époque de l'action  $a$  dans l'état de croyance  $b$  en supposant que la politique optimale sera exécutée après. RTDP-bel n'utilise pas d' $\alpha$ -vecteurs pour représenter la fonction de valeur, mais utilise une table de hachage dont la clé est l'état de croyance discrétisé. À chaque étape, l'état de croyance  $b$  est projeté vers un état de croyance proche afin de mettre à jour cet état de croyance discrétisé. Cette fonction de discrétisation projette chaque  $b(s)$  dans  $d(b(s)) = \text{ceil}(D * b(s))$  où  $D$  est un entier positif et la fonction  $\text{ceil}(x)$  est l'entier supérieur ou égal à  $x$ . Cette discrétisation est seulement utilisée pour les opérations d'accès à la valeur de l'état de croyance et n'affecte en rien l'état de croyance lui-même. C'est une fonction d'approximation qui représente par la même valeur tous les états de croyances  $b'$  qui se discrétise tel que  $d(b') = d(b)$ . Cette approximation repose sur l'hypothèse que si des états de croyances ont des valeurs proches, alors ils doivent être proches les uns des autres.

Cette discrétisation des états de croyances rend la fonction de valeur finie, mais cela empêche certaines garanties de RTDP d'être généralisé à RTDP-bel. Tout d'abord, la convergence de l'algorithme n'est pas garantie. Ensuite, la fonction de valeur approximée de cette façon n'est pas assurée de rester la fonction de valeur minimale. De plus, il ne peut être garanti qu'un trial de RTDP-bel atteindra le but, même si ordinairement c'est le cas.

### 1.5.3 POMCP

POMCP (Alg. 4) [SILVER et al. 2010] est un algorithme de type *en ligne* de basé sur l'algorithme *Monte-Carlo Tree Search* (MCTS) c'est-à-dire que la politique est calculée pendant l'exécution de la mission. Avant d'exécuter une action, l'algorithme calcule une politique dans le temps imparti. POMCP fonctionne en échantillonnant un état  $s$  à partir de l'état de croyance initial  $b_0$ , et en simulant des séquences d'action-observation (par une procédure de *trial*) pour construire un arbre de nœuds de croyance. Chaque nœud d'arbre  $h$  représente un historique des paires action-observation depuis l'état de croyance initial (étant un nœud de croyance  $h$ ). POMCP calcule pour un nœud donné  $h$  de l'arbre le coût moyen observé pour

**Algorithme 3 : RTDP-bel**


---

```

1  $b \leftarrow b_0$ 
2 tant que  $b$  n'est pas le but faire
3    $s \leftarrow b$ 
4   pour chaque  $a \in A$  faire
5      $Q(b, a) = c(b, a) + \sum_{o \in \Omega} p(o|b, a)V(b_o^a)$ 
6     si  $b_o^a$  n'est pas dans la table de hachage alors
7        $V(b_o^a) = d(b_o^a)$ 
8    $a_{\text{best}} \leftarrow \arg \min_{a \in A} Q(b, a)$ 
9    $V(b) = Q(b, a_{\text{best}})$ 
10   $b_a \leftarrow$  exécute  $a_{\text{best}}$  in  $b$ 
11   $o \leftarrow b_a$ 
12   $b_a^o \leftarrow$  backup( $b_a, o$ )
13   $b \leftarrow b_a^o$ 

```

---

tous les *trials* qui ont commencé à partir de ce nœud. POMCP ne met pas à jour l'état de croyance après chaque paire action-observation, mais met à jour le coût moyen pour chaque nœud de l'arbre appelé, et garde en mémoire le nombre de fois où un nœud a été exploré  $N(h)$ , et le nombre de fois où une action donnée a été choisie  $N(ha)$  dans ce nœud. La figure 1.9 représente schématiquement l'évolution de l'historique durant l'optimisation.

Cette procédure (*trial*) permet à POMCP d'approcher la fonction de valeur et la politique pour un nœud donné (un état de croyance représenté par un historique). Pendant le calcul de la valeur et de la politique associée, les *trials* sont réalisés de manière gloutonne. L'action la plus prometteuse, suivant une heuristique donnée, est choisie dans chaque nœud de l'arbre. Plus précisément, dans chaque nœud, la stratégie de sélection d'action gloutonne est basée sur le même choix heuristique que UCT [Kocsis et al. 2006], la méthode dite UCB1 (Upper Confidence Bound 1) [Auer et al. 2002].

Cette stratégie de sélection des actions est basée sur une combinaison de deux caractéristiques, la  $Q$ -valeur de l'action et une mesure du degré d'exploration d'une action. La valeur  $Q$  de l'action peut être définie comme suit :

$$Q(b, a) = \left[ c(b, a) + \gamma \sum_{o \in \Omega} p(o|b, a)V(b_o^a) \right], \quad (1.26)$$

étant la valeur de l'exécution d'une action  $a$  dans l'état de croyance  $b$ , en supposant que la politique optimale sera exécutée après.  $V$  est alors le coût réel attendu pour atteindre l'objectif [KoloBOV 2012].

Parce que POMCP fonctionne en simulant des séquences d'action-observation, la  $Q$ -valeur de chaque action n'est pas calculée de cette façon, mais est estimée après chaque séquence d'action-observation, ce qui permet d'estimer la valeur de  $V(b) = \min_{a \in A} Q(b, a)$ .

La deuxième caractéristique de cette stratégie de sélection d'action gloutonne utilisée lors

**Algorithme 4 : POMCP**


---

```

1 Fonction POMCP( $h, b_0, \gamma$ )
2    $h \leftarrow b_0$ 
3   tant que qu'il reste du temps faire
4      $s \leftarrow \text{sampling } s \text{ from } b_0$ 
5      $Trial(h, s, 0)$ 
6    $a^* \leftarrow \arg \min_{a \in A} Q(b_0, a)$ 
7   retourner  $a^*$ 

8 Fonction Rollout( $s, h, depth, \gamma$ )
9   si  $\gamma^{depth} < \epsilon$  alors
10    retourner 0
11    $a \leftarrow \pi_{rollout}(h)$ 
12    $s', o \leftarrow \mathcal{G}(s, a)$  /* Générateur aléatoire */
13   retourner  $r(s', a) + \gamma Rollout(s', hao, depth + 1, \gamma)$ 

14 Fonction Trial( $h, s, depth$ )
15   si  $\gamma^{depth} < \epsilon$  alors
16    retourner 0
17   si  $h \notin T$  alors
18     pour chaque  $a \in A$  faire
19        $T(hao) \leftarrow (N_{init}(ha), V_{init}(ha), \emptyset)$ 
20     retourner  $Rollout(s, h, depth + 1, \gamma)$ 
21    $a \leftarrow \arg \min_{a \in A} \left( Q(s, a) - c \sqrt{\frac{\log N(h)}{N(ha)}} \right)$ 
22    $ha \leftarrow h + a$ 
23    $s', o \leftarrow \mathcal{G}(s, a)$ 
24    $hao \leftarrow ha + o$ 
25    $N(h) \leftarrow N(h) + 1$ 
26    $N(ha) \leftarrow N(ha) + 1$ 
27    $Q(h, a)' \leftarrow C(s, a) + \gamma Trial(hao, s', depth + 1)$ 
28    $Q(h, a) \leftarrow Q(h, a) + \frac{Q(h, a)' - Q(h, a)}{N(ha)}$ 
29    $V(h) \leftarrow \min_{a \in A} Q(h, a)$ 

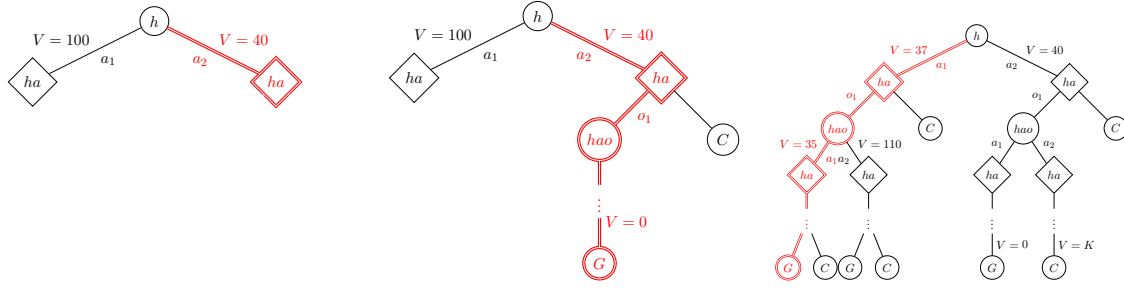
```

---

du calcul des politiques est une mesure (donnée par  $c \sqrt{\frac{\log N(h)}{N(ha)}}$ ) du degré d'exploration de l'action  $a$ , étant donné l'historique  $h$  (ou noeud de croyance). Cette mesure est utilisée pour équilibrer le choix d'action dans l'algorithme, qui sélectionnera l'action qui minimise  $Q(h, a) - c \sqrt{\frac{\log N(h)}{N(ha)}}$  (la stratégie UCB1). En d'autres termes, l'algorithme sélectionnera l'action qui minimise le regret de choisir une mauvaise action [KOCIS et al. 2006].

Le coefficient d'exploration  $c \geq 0$  est utilisé pour forcer l'algorithme à essayer des actions qui semblent a priori moins intéressantes pour éviter de tomber dans une politique locale optimale. Si  $c$  est élevé, l'algorithme essaiera plus souvent les actions qui semblent moins intéressantes, au contraire, si  $c$  est faible, l'algorithme explorera rarement différentes actions.

Dans la suite, nous présenterons une extension des POMDP, les MOMDP - *Mixed Observability Markov Decision Process* [S. C. ONG et al. 2010; ARAYA-LÓPEZ et al. 2010] qui factorise l'espace d'état entre les variables d'état qui sont complètement observables et les



(a) Arbre de l'historique au début de l'algorithme. (b) Arbre de l'historique après le premier essai. (c) Arbre de l'historique après  $N$  essais.

FIGURE 1.9 – Évolution de l'historique durant l'optimisation.

variables d'état qui sont partiellement observables.

## 1.6 MOMDP - *Mixed Observability Markov Decision Process*

Résoudre un POMDP est un processus coûteux et nécessite dans le cas de problème réel d'utiliser des algorithmes de résolutions approchés. Ce coût de résolution vient en partie du fait que les fonctions de transition d'état, d'observation et de récompense utilisent l'entièreté de l'espace d'état pour définir leurs domaines. Cependant, une extension des POMDP, appelés MOMDP - *Mixed Observability Markov Decision Process* [S. C. ONG et al. 2010; ARAYA-LÓPEZ et al. 2010] a été proposé.

Les MOMDP exploitent la structure du problème à résoudre pour factoriser l'espace d'état en deux ensembles de variables : les variables d'état totalement observables et les variables d'état partiellement observables. Dans ce modèle factorisé, il est considéré que certaines variables d'état peuvent être observées complètement et donc ne nécessitent pas d'être incluses dans le domaine de la fonction d'observation. Ce découpage permet un gain dans le temps de calcul très important, car la fonction de mise à jour de l'état de croyance n'a pas besoin d'inférer sur les variables d'état totalement observables et les opérations sur les  $\alpha$ -vecteurs traiteront avec des vecteurs de plus petites dimensions.

Dans les modèles proposés par [S. C. ONG et al. 2010] et [ARAYA-LÓPEZ et al. 2010], l'espace d'état est divisé en deux ensembles de variables d'état, les variables partiellement observables  $s_p \in S_p$  et les variables totalement observables  $s_v \in S_v$ . L'espace d'état complet est alors représenté par le couple  $(s_p, s_v)$  tel que  $|S| = |S_p| \times |S_v|$  représente le domaine complet de l'espace d'état. [ARAYA-LÓPEZ et al. 2010] propose de factoriser aussi l'espace d'observation en accord avec la factorisation de l'espace d'état. L'espace d'observation est alors défini en deux ensembles de variables, les variables d'observation sur les variables d'état partiellement observables  $o^{s_p} \in O^{S_p}$  et les variables d'observation sur les variables d'état totalement observables  $o^{s_v} \in O^{S_v}$ . Étant donné que la probabilité  $p(o^{s_v} | s_v) = 1$ , la différence entre les deux modèles MOMDP tient plus de la notation.

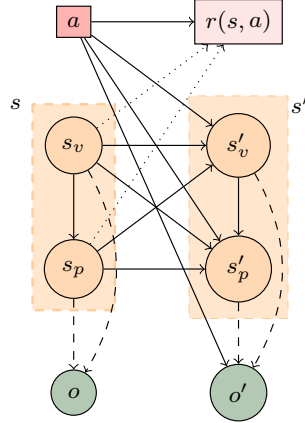


FIGURE 1.10 – Modèle factorisé de la transition d'état du MOMDP proposé par [S. C. ONG et al. 2010] pour une action  $a_t$ .

Le MOMDP de [S. C. ONG et al. 2010] est défini par un  $n$ -uplet  $(S_p, S_v, A, \Omega, T_p, T_v, O, R, (s_{v,0}, b_{p,0}))$  où :

- $S_p$  : est l'ensemble fini des variables discrètes d'état partiellement observables ;
- $S_v$  : est l'ensemble fini des variables discrètes d'état totalement observables ;
- $A$  : est l'ensemble fini des actions discrètes ;
- $\Omega$  : est l'ensemble fini des observations que l'agent peut recevoir ;
- $T_p$  :  $T_p(s_p, s_v, a, s'_p, s'_v) = p(s'_p | s_p, s_v, a, s'_v)$  comme étant la probabilité que la variable partiellement observable prennent la valeur  $s'_p$  sachant l'action  $a$  dans l'état  $(s_p, s_v)$  et qu'on arrive dans l'état  $s'_v$ .
- $T_v$  :  $T_v(s_p, s_v, a, s'_v) = p(s'_v | s_p, s_v, a)$  comme étant la probabilité que la variable totalement observable prennent la valeur  $s'_v$  sachant l'action  $a$  dans l'état  $(s_p, s_v)$ .
- $O$  :  $\Omega \times S_p \times A \rightarrow [0, 1]$  : est la fonction d'observation, où  $\forall o \in \Omega, \forall a \in A, \forall s'_v \in S_v$  et  $\forall s'_p \in S_p$ , on définit :

$$O(s'_p, s'_v, a, o) = p(o | s'_p, s'_v, a), \forall t \quad (1.27)$$

comme étant la probabilité que l'agent reçoive l'observation  $o$  dans l'état  $s'$  après avoir effectué l'action  $a$  ;

- $R$  :  $S_p \times S_v \times A \rightarrow \mathbb{R}$  est la fonction de récompense. Elle permet de quantifier l'utilité de l'action  $a$  dans l'état  $s$  par rapport au but de l'agent ;
- $(b_{p,0}, s_{v,0})$  : l'état de croyance initiale  $b_{p,0}$  conditionné par  $s_{v,0}$ .

En regardant la définition du MOMDP et le modèle de transition d'état (Figure 1.10), on constate l'avantage de ce modèle dans la représentation de l'état de croyance dont la distribution de probabilité sur les états partiellement observables  $s_p$  est conditionnée par la valeur de  $s_v$ . Il est donc seulement nécessaire de maintenir la distribution de probabilité sur les variables  $s_p$  notées  $b_p$ . Donc l'espace des états de croyances pour la variable d'état partiellement observable  $s_p$  conditionné par  $s_v$  est noté  $B_p$  tel que  $B_p(s_v) = \{(b_p, s_v), b_p \in B_p\}$ . La représentation par des MOMDP permet d'exploiter le fait que la dimension de l'état de croyance ne dépend que des variables partiellement observables. L'espace entier des états de

croyance  $B$  est alors défini comme  $B = \bigcup_{s_v \in S_v} B_p(s_v)$ . Il faut alors changer la fonction de mise à jour de l'état de croyance :

$$b_{p,a}^o(s'_p) = \lambda \sum_{s_p \in S_p} p(o|s'_p, s'_v, a) p(s'_p|s_p, s_v, a, s'_v) p(s'_v|s_v, s_p, a) b_p(s_p) \quad (1.28)$$

où  $\lambda$  est une constante de normalisation.

Ce modèle a pour objectif de réduire la complexité de la mise à jour de l'état de croyance d'un POMDP par la factorisation de l'espace d'état. Les MOMDP permettent d'obtenir un vrai gain de performance sur les algorithmes d'itération de la valeur approchée adaptés à ce modèle [S. C. ONG et al. 2010] (dont SARSOP), car l'espace des distributions de probabilité est plus petit et donc les  $\alpha$ -vecteurs ont une aussi une dimension plus petite.

## 1.7 Bilan

Dans ce chapitre, nous avons présenté le modèle POMDP ainsi que quelques algorithmes de résolution exacte ou approchée. Nous avons aussi présenté une extension des POMDP pour l'observabilité mixte les MOMDP. Les algorithmes approchés hors ligne tels que PBVI [PINEAU et al. 2003] et RTDP-bel [Hector GEFNER et al. 1998 ; BONET et Héctor GEFNER 2009] permettent d'obtenir des solutions pour des problèmes avec un espace d'état très grand. Il existe aussi de nombreux autres algorithmes de résolution approchée qui calculent des politiques quasi optimales telles que HSVI [SMITH et al. 2004], FSVI [SHANI, BRAFMAN et al. 2007] ou encore SARSOP [KURNIAWATI, HSU et al. 2008]. Les algorithmes de résolution en ligne tels que POMCP [SILVER et al. 2010] permettent aussi de résoudre ces problèmes en s'appuyant sur les méthodes hors ligne pour utiliser des heuristiques de recherches pertinentes, ou en se basant sur la simulation Monte-Carlo. De plus, beaucoup de ces méthodes sont applicables aux modèles factorisés proposés par [S. C. ONG et al. 2010 ; ARAYA-LÓPEZ et al. 2010] qui contribue encore aux gains de performance.

Dans le prochain chapitre, nous étudierons les différentes approches utilisées actuellement dans le domaine de la planification de chemin pour des véhicules autonomes sous l'incertitude de localisation. Nous verrons quelles méthodes sont utilisées pour planifier des chemins en tenant compte de la dégradation de la précision des capteurs en fonction de l'environnement.





# 2

## Méthodes existantes pour la planification de chemin sous incertitude de localisation et/ou d'exécution

### Sommaire

---

<b>2.1 Algorithmes de planification de chemins sous l'incertitude sur la localisation</b>	<b>31</b>
2.1.1 <i>Rapidly-exploring Random Belief Trees</i> - RRBT	32
2.1.2 Recherche de chemins et choix d'une stratégie de navigation et de guidage	34
2.1.3 Recherche de chemins dans l'espace des états de croyances	35
<b>2.2 Utilisation des POMDP pour le calcul de chemins</b>	<b>37</b>
2.2.1 Utilisation des POMDP pour la conduite autonome à travers la foule.	37
2.2.2 Planification de mouvement sous incertitude en utilisant itérativement des optimisation locale dans l'espace de croyance	38
<b>2.3 Conclusion et Bilan</b>	<b>39</b>

---

Ce chapitre a pour objectif de présenter les différents travaux réalisés sur la planification de chemins sûrs pour des véhicules autonomes, qui prennent en compte la dégradation de la précision de la localisation due à la non-disponibilité ou la précision dégradée des capteurs. Nous présenterons tout d'abord différentes méthodes pour prendre en compte l'incertitude sur la position du véhicule pour minimiser le risque de collision. Nous présenterons également des méthodes qui visent à introduire la disponibilité connue a priori des capteurs dans des modèles de planification tels que les POMDP.

### 2.1 Algorithmes de planification de chemins sous l'incertitude sur la localisation

Plusieurs travaux se sont penchés sur la question du calcul de chemin pour des véhicules autonomes en prenant en compte la disponibilité des capteurs ainsi que son impact sur l'incertitude sur la position du véhicule, car celle-ci a un impact direct sur le risque de collision

et donc sur le succès de la mission. Ces travaux sont très souvent basés sur des algorithmes de planification de chemin le plus court en distance parcourue.

### 2.1.1 *Rapidly-exploring Random Belief Trees* - RRBT

Le problème général de planification de mouvement qui consiste à essayer de trouver un chemin sûr et de distance minimale depuis un état de départ jusqu'à un objectif quelconque a fait l'objet d'études approfondies. En particulier, les méthodes de planification de chemin basées sur l'échantillonnage ont fait l'objet de quelques travaux au cours des dernières années. L'algorithme *Rapidly-exploring Random Tree (RRT)* [LAVALLE 1998] est basé sur la création d'un arbre de configurations ou états atteignables, par échantillonnage itératif de nouveaux états, puis en les liants vers des nœuds existants dans l'arborescence qui sont les plus proches. RRT a de nombreuses propriétés utiles, notamment probabilistes, sur l'exhaustivité et la décroissance exponentielle de la probabilité d'échec avec le nombre d'échantillons. Cet algorithme est un algorithme de planification de mouvement pour les problèmes déterministes qui ne traite d'aucune incertitude.

Toutefois [BRY et al. 2011] considèrent des problèmes de planification de chemin pour des véhicules en utilisant l'algorithme RRBT (une variante de RRT). L'hypothèse est que le véhicule ne reçoit des mesures que dans certaines zones spécifiques. Les zones de couverture du capteur sont connues à l'avance et hors de cette zone le véhicule ne peut pas corriger l'erreur sur l'estimation de la position et donc l'incertitude ne fera que croître. L'objectif dans ce problème est de trouver un chemin sans risque de collision et minimisant la distance parcourue (Figure 2.1). Dans ces travaux, le système est considéré comme non linéaire et partiellement observable (Chapitre 3). Le véhicule considéré est défini par sa dynamique et son capteur embarqué tel que :

$$x_t = f(x_{t-1}, u_{t-1}, w'_t), \quad w'_t \sim N(0, Q') \quad (2.1)$$

$$z_t = h(x_t, v'_t), \quad v'_t \sim N(0, R') \quad (2.2)$$

où  $x_t \in X$  est le vecteur d'état du véhicule,  $u_t \in U$  le vecteur d'entrée,  $w'_t$  une perturbation aléatoire,  $z_t$  le vecteur de la mesure et  $v'_t$  un bruit aléatoire sur la mesure. [BRY et al. 2011] incluent un filtre de Kalman afin d'estimer l'état à chaque pas de temps. Après utilisation du filtre de Kalman, la croyance sur l'état est représentée par une loi normale centrée sur un état estimé et dont la covariance est celle renvoyée par le filtre de Kalman.

RRBT se base sur RRT pour construire un graphe et chercher dans ce graphe un arbre dans l'espace des états de croyance. L'algorithme travaille sur un ensemble de sommets  $V$  et un ensemble d'arêtes  $E$  qui définissent un graphe. Chaque sommet  $v \in V$  est composé d'un état  $v.x$  et d'un ensemble de nœuds de croyance  $v.N$ . Chacun des nœuds de croyance  $n \in v.N$  possède la covariance sur l'incertitude d'estimation d'état  $n.\Sigma$  qui est issue du filtre de Kalman, la distribution sur tous les états estimés possibles  $n.\Lambda$ , un coût  $n.c$  et un nœud de croyance parent  $n.parent$ .

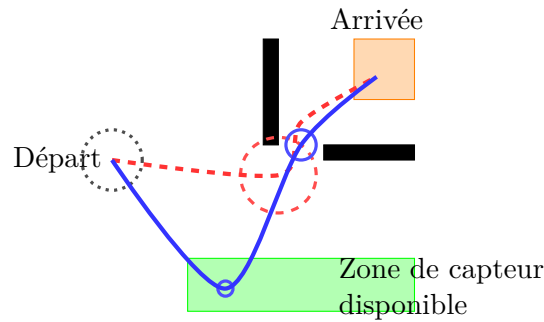


FIGURE 2.1 – Représentation de la différence entre deux chemins, le chemin en bleu passant par la zone où le capteur est disponible réduisant ainsi l'incertitude sur la position et permettant de passer entre les deux obstacles. Le deuxième chemin en rouge ne passant pas par la zone où le capteur est disponible. Le chemin en rouge subit alors un accroissement de l'incertitude sur la position ne permettant pas de passer entre les deux obstacles.[BRY et al. 2011]

Les nœuds de croyances correspondent à un chemin unique à travers le graphe pour atteindre le sommet  $v$  et dont ses variables membres ( $n.\Sigma$ ,  $n.\Lambda$ ,  $n.c$ ) sont les propriétés qui résultent du fait de suivre ce chemin. Chaque arête  $e \in E$  contient la trajectoire et les lois de contrôle qui relient les sommets concernés. L'algorithme utilise une liste de recherche  $L$  composée de nœuds de croyance qui vont garder une trace des chemins qui ont besoin d'être mis à jour.

**Déroulement de l'algorithme.** L'algorithme 5 représente l'algorithme RRBT. Le graphe est initialisé avec un seul sommet et une seule croyance sur l'état qui correspond à l'estimation de l'état initial (Lignes 1-3). Cette croyance formera la racine de l'arbre de recherche. À chaque itération de la boucle principale, le graphe d'état est mis à jour en échantillonnant un nouvel état suivant en considérant la dynamique du véhicule, puis en ajoutant des arêtes au sommet le plus proche et aux sommets proches avec la fonction *Connecter()*. Chaque fois qu'un sommet existant a une arête sortante qui est ajoutée, tous les nœuds de croyances à ce sommet sont ajoutés à la file d'attente. Le nouveau sommet est ajouté au graphe seulement s'il n'y a pas de collision lors de la propagation de l'état avec la fonction *Propager( $e, n$ )* (Ligne 8). Après que toutes les arêtes ont été ajoutées, la file d'attente est exhaustivement évaluée à l'aide d'une recherche uniforme des coûts (Lignes 18-23). Pour plus de détail sur l'algorithme, nous vous conseillons de vous référer à [BRY et al. 2011].

Ces travaux proposent donc un algorithme de recherche de chemin sûr et minimisant la distance parcourue. Cependant, cette méthode considère que la disponibilité des capteurs est fixe et connue dans l'environnement, et ne peut pas traiter la disponibilité probabiliste des capteurs. De plus, l'algorithme calcule un chemin déterministe et donc le chemin ne peut pas s'adapter pour la vraie disponibilité d'un capteur observé lors de l'exécution.

**Algorithme 5 : Algorithme RRBT**


---

```

1  $n.\Sigma \leftarrow \Sigma_0$ ;  $n.\Lambda \leftarrow 0$ ;  $n.c \leftarrow 0$ ;  $n.parent \leftarrow NULL$ ;
2  $v.x \leftarrow x_0$ ;  $v.N \leftarrow \{n\}$ ;
3  $V \leftarrow \{v\}$ ;  $E \leftarrow \{\}$ ;
4 while  $i < M$  do
5    $x' \leftarrow \text{Échantillonner}()$ ;
6    $v_{\text{PlusProche}} \leftarrow \text{PlusProche}(V, x')$ ;
7    $e_{\text{PlusProche}} \leftarrow \text{Connecter}(v_{\text{PlusProche}}.x, x')$ ;
8   if  $\exists v_{\text{PlusProche}}.n : \text{Propager}(e_{\text{PlusProche}}, n)$  then
9      $V \leftarrow V \cup v(x')$ ;
10     $E \leftarrow E \cup e_{\text{PlusProche}}$ ;
11     $E \leftarrow E \cup \text{Connecter}(x', v_{\text{PlusProche}}.x)$ ;
12     $L \leftarrow L \cup v_{\text{PlusProche}}.N$ ;
13     $V_{\text{proche}} \leftarrow \text{Proche}(V, v_{\text{proche}})$ ;
14    forall  $v_{\text{proche}} \in V_{\text{proche}}$  do
15       $E \leftarrow E \cup \text{Connecter}(v_{\text{proche}}.x, x')$ ;
16       $E \leftarrow E \cup \text{Connecter}(x', v_{\text{proche}}.x)$ ;
17       $L \leftarrow L \cup v_{\text{proche}}.N$ ;
18    while  $L \neq \emptyset$  do
19       $n \leftarrow \text{Pop}(L)$ ;
20      forall  $v_{\text{voisin}}$  de  $v(n)$  do
21         $n' \leftarrow \text{Propager}(e_{\text{voisin}}, n)$ ;
22        if  $\text{AjouterNoeud}(v_{\text{voisin}}, n')$  then
23           $L \leftarrow L \cup n'$ ;
24     $i \leftarrow i + 1$ ;

```

---

**2.1.2 Recherche de chemins et choix d'une stratégie de navigation et de guidage**

[WATANABE, DESSUS et al. 2014; WATANABE, VEILLARD et al. 2016] considèrent un problème dans lequel un véhicule (drone) doit se rendre d'un point de départ à un point d'arrivée en minimisant la distance et sans risque de collision. Il est considéré que le système embarqué du véhicule contient plusieurs modes de navigation (localisation) et de guidage ce qui donne lieu à différentes évolutions de l'erreur de localisation et d'exécution. Par exemple, deux modes de guidages ont été inclus, le guidage absolu qui suit des points de passage et le guidage relatif qui permet de guider le véhicule par rapport à son environnement (par exemple en suivant des murs). L'idée d'utiliser le mode de guidage relatif est que l'incertitude d'exécution du chemin ne dépend plus de l'erreur de localisation absolue, mais dépend de la précision du capteur pour percevoir l'environnement. En d'autres termes, ce mode de guidage relatif permet d'assurer la sécurité du vol des véhicules par rapport au risque de collision même en cas de dégradation importante de la précision de la localisation des drones due à la perte de signal GPS.

Leur modèle se base sur la carte de l'environnement et une carte de disponibilité pour chacun des capteurs embarqués. Ces cartes sont représentées sous forme de grille indiquant pour chaque cellule de la grille si le capteur est disponible / ou si l'espace est occupé de façon certaine ou non. Ce planificateur intègre une modélisation de la propagation de l'incertitude de la localisation par un filtre de Kalman en utilisant des capteurs disponibles. L'erreur de

l'exécution du chemin est propagée par le mode de guidage utilisé. Ce choix du mode de navigation et de guidage permet lors de la planification du chemin de sélectionner le capteur disponible et le mode de guidage le plus adapté afin d'assurer la sécurité de la mission.

[WATANABE, VEILLARD et al. 2016] propose une fonction de coût. Dans ces travaux, les auteurs ont développé la notion de *couloir d'incertitude*. Ce couloir représente toutes les positions probables du véhicule durant la transition entre deux points de l'environnement. Utiliser le volume du couloir d'incertitude comme fonction de coût permet de minimiser directement l'incertitude sur la position (perpendiculaire au chemin) et la distance parcourue sans devoir privilégier l'un ou l'autre de ces critères.

L'algorithme utilisé dans ces travaux pour calculer le chemin est l'algorithme A\* qui est appliqué dans un espace de recherche en 5 dimensions (position en 3D et les modes de navigation et de guidage). L'espace entre deux points est traversable si le couloir d'incertitude ne rentre pas en collision avec un obstacle.

Cet algorithme permet de calculer un chemin sans risque de collision et de distance minimale tout en choisissant le meilleur couple de modes de navigation et de guidage à chaque pas de temps. La disponibilité des capteurs dans les problèmes considérés dans [BRY et al. 2011; WATANABE, VEILLARD et al. 2016] est déterministe. Le défaut principal de ces travaux est de considérer que le déplacement du véhicule est déterministe malgré l'incertitude de localisation, alors que dans les problèmes réels le déplacement d'un véhicule contient aussi l'incertitude issue de la localisation.

### 2.1.3 Recherche de chemins dans l'espace des états de croyances

[SCHIRMER et al. 2017] ont aussi travaillé sur la planification de chemin sans risque de collision pour des véhicules sous l'incertitude de localisation. Comme dans les travaux précédents, l'imperfection du capteur embarqué est prise en compte et son impact sur le risque de collision aussi. L'idée est alors de maintenir une distribution de probabilité sur les positions possibles du véhicule appelé *état de croyance*  $b$ . Les états de croyances sont définis par une fonction gaussienne ayant pour moyenne  $\mu$  et covariance  $\Sigma$  tel que  $b \sim \mathcal{N}(\mu, \Sigma)$ . Cet état de croyance est mis à jour au fur et à mesure du temps à travers les actions des véhicules.

Cet espace de croyance continu a été discrétisé en ne considérant que des états de croyances centrés dans chaque cellule d'une grille préalablement créée. Afin de réduire encore plus le temps de calcul du chemin, la transition (l'évolution de l'incertitude sur la localisation) d'un état de croyance à un autre a été simulée en avance et retournée sous forme d'une carte de localisation. L'idée est de se baser sur une carte contenant pour chaque cellule l'approximation a priori de la covariance qui sera estimée après correction par filtre de Kalman.

L'algorithme de résolution pour ce problème est une adaptation de l'algorithme de recherche de [CENSI et al. 2008]. Cet algorithme fonctionne sur le principe de la recherche de chemin dans un graphe. Un nœud  $n$  dans ce graphe est un tuple  $\langle \mu, \Sigma, d \rangle$  qui encode l'évolution espérée de l'état de croyance sur le chemin depuis la position de départ jusqu'à la moyenne  $\mu$

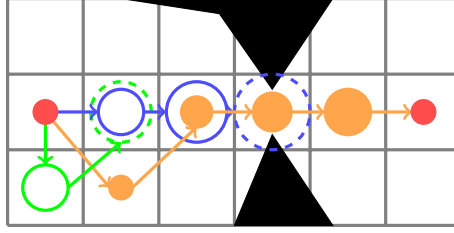


FIGURE 2.2 – Illustration de la dominance entre deux nœuds, le nœud vert pointillé est dominé par le nœud bleu, car il minimise la distance et l'incertitude. Afin de trouver la solution la plus sûre, la solution orange se doit d'être considérée, car elle est la seule dont l'incertitude sur la localisation (représentée par les cercles) ne rentre pas en collision.[SCHIRMER et al. 2017]

du nœud  $n$ ,  $d$  est la longueur de chemin jusqu'à la position  $\mu$ . La fonction  $\text{SUCCESSEUR}(n)$  génère tous les nœuds successeurs pour toutes les cellules adjacentes en utilisant la mise à jour de l'état de croyance du filtre de Kalman et en rejetant tous les états de croyances en collisions.

La fonction de coût  $c$  d'un nœud  $n$  est la somme des coûts associés à la distance de chemin et de l'incertitude de la localisation et définie comme :

$$c(n) = \text{SDistance}(n) + \text{SIncertitude}(n) \quad (2.3)$$

$$\text{SDistance}(n) = \text{DIJKSTRA}(n, \text{goal}) + d(n) - L_{\text{ideal}} \quad (2.4)$$

$$\text{SIncertitude}(n) = \text{D-Opt}(c_{\text{min}}) \times (\text{SDistance}(n) - d(n)) + \text{D-Opt}(n) \quad (2.5)$$

où  $\text{DIJKSTRA}(n, \text{goal})$  est la longueur du chemin le plus court et sans collision sans considérer l'incertitude entre le nœud actuel et le but.  $L_{\text{ideal}} = \text{DIJKSTRA}(n_0, \text{goal})$  est la longueur du chemin le plus court et sans collision sans considérer l'incertitude de la position de départ et jusqu'au but. La covariance minimale donnée par la carte de localisation  $\text{D-Opt}(\Sigma)$  est calculée par :

$$\text{D-Opt}(\Sigma) = \frac{1}{2}(\exp(\log \lambda_1(\Sigma) + \log \lambda_2(\Sigma))) \quad (2.6)$$

avec  $\lambda_{1,2}(\Sigma)$  représentant les valeurs propres de la matrice de covariance  $\Sigma$ . Cette méthode est proche de la notion du volume de couloir d'incertitude présentée dans la section précédente (Section 2.1.2), l'objectif de ces deux fonctions de coût étant de minimiser en même temps la distance et l'incertitude d'exécution du chemin.

Cette approche permet de calculer des chemins sans risque de collision de distance minimale. De plus la fonction de coût utilisée pourrait être directement utilisée dans notre problème. Cependant, cette méthode traite uniquement des problèmes de planification déterministe.

## 2.2 Utilisation des POMDP pour le calcul de chemins

Dans le chapitre (1), nous avons vu que les POMDP permettent de modéliser et résoudre des problèmes de décision séquentielle dans l'incertain sous observabilité partielle. Le POMDP étant un formalisme général, il n'a pas été spécialement conçu pour résoudre des problèmes de planification de chemins, qui peuvent s'avérer difficile dû aux espaces d'états et d'actions de grandes dimensions. Mais la recherche pour la résolution de POMDP avec de grands espaces d'état ou d'actions a beaucoup progressé et reste un domaine de recherche très actif [SEILER et al. 2015; KURNIAWATI et YADAV 2016; YEE et al. 2016; SUNBERG et al. 2018]. Nous présenterons ci-dessous quelques-uns de ces travaux.

### 2.2.1 Utilisation des POMDP pour la conduite autonome à travers la foule

L'utilisation des POMDP pour la planification de chemins sous incertitude n'est pas très répandue en particulier à cause de la dimension de l'espace d'état dans ce type d'application. Cependant, [BAI, CAI et al. 2015] proposent de résoudre le problème de la navigation d'un véhicule autonome à travers une foule en se basant sur l'algorithme de résolution des POMDP en ligne, *DESPOT* [SOMANI et al. 2013]. Le choix de cet algorithme de résolution en ligne a été basé sur son efficacité reconnue permettant de prendre en compte la dynamique de l'environnement comme l'apparition soudaine de nouveaux obstacles.

L'espace d'état est alors très important, car la position dynamique de chaque personne (obstacle) dans la foule doit être prise en compte. Pour diminuer la complexité, [BAI, CAI et al. 2015] propose une méthode de planification à deux niveaux basée sur les POMDP (Figure 2.3). Le premier niveau relatif au contrôle est le planificateur de chemin déterministe, qui va planifier l'orientation des roues du véhicule en fonction de sa position dans l'environnement pour arriver jusqu'au but. Le deuxième niveau est modélisé avec un POMDP et va contrôler la vitesse du véhicule. De cette façon, l'espace d'action est réduit par rapport à un espace d'actions contenant aussi la direction de mouvement du véhicule, ce qui réduit la complexité de résolution du problème. Le planificateur dans ce problème est donc composé de trois modules distincts, le planificateur de chemin, le planificateur de vitesse et le module gardant à jour l'état de croyance. De plus, cette méthode considère un véhicule qui est modélisé par un capteur et un contrôleur (Figure 2.3) ce qui permet de considérer de façon simple la dynamique du véhicule.

**Planificateur de vitesse** Le planificateur de vitesse est basé sur un POMDP et contrôle l'accélération du véhicule. L'espace d'action du POMDP est discret et contient trois actions : accélérer, maintenir et décélérer. L'observation dans ce problème est modélisée comme un vecteur contenant la position du véhicule, sa vitesse et la position de tous les piétons. Il n'y a pas d'incertitude sur ces observations, seule la destination des piétons n'est pas observable et doit être inférée. Le véhicule reçoit une récompense quand il se rapproche du but, mais reçoit des pénalités quand un piéton est trop près, quand il utilise les actions accélérer et décélérer afin de minimiser l'énergie.

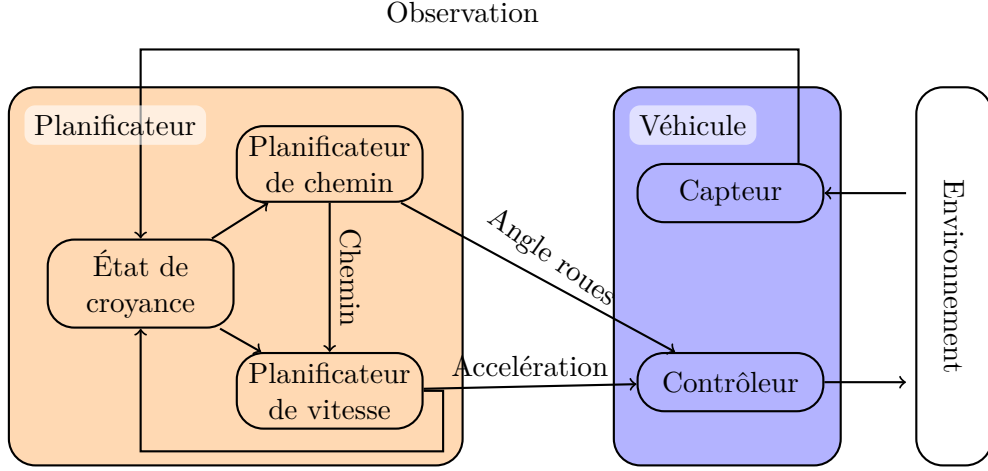


FIGURE 2.3 – Modèle d'un planificateur POMDP sur deux niveaux.

La trajectoire réalisée par le véhicule peut être sous-optimale, car le problème d'optimisation est divisé en deux niveaux en cascade. Il nous semble donc nécessaire dans notre problème de prendre en compte la planification du chemin directement dans le POMDP pour obtenir des chemins optimaux.

### 2.2.2 Planification de mouvement sous incertitude en utilisant itérativement des optimisation locales dans l'espace de croyance

[BERG et al. 2012] présentent une approche pour la planification de chemins en intégrant le modèle de mouvement du véhicule en boucle fermée (GNC) dans un modèle POMDP. Ils proposent de résoudre ce problème en calculant un optimum local à partir d'une trajectoire initiale.

Dans ce problème, ils définissent l'état du véhicule  $x_t$ , les observations  $z_t$  comme étant les mesures du capteur, la fonction de transition  $f$  et la fonction d'observation  $h$  de la façon suivante :

$$x_{t+1} = f(x_t, u_t, w_{t+1}), \quad w_{t+1} \sim N(0, Q) \quad (2.7)$$

$$z_{t+1} = h(x_t, v_{t+1}), \quad v_{t+1} \sim N(0, R) \quad (2.8)$$

où  $w_{t+1}$  représente l'erreur d'exécution et  $v_{t+1}$  le bruit sur la mesure.

L'état de croyance initial est alors représenté par une fonction gaussienne de la façon suivante  $b_0 = (\hat{x}_0, \Sigma_0)$ , avec  $\hat{x}_0$  l'état estimé initiale et  $\Sigma_0$  l'incertitude initiale sur l'état. Dans ce problème, la mise à jour de l'état de croyance est faite à l'aide d'un filtre de Kalman étendu.

Étant donnée la complexité pour calculer la politique optimale d'un POMDP, [BERG et al. 2012] ont fait le choix de calculer une solution optimale localement. Pour cela, ils considèrent un chemin nominal donné calculé a priori sur lequel ils appliquent l'itération sur la valeur en



partant de l'état estimé final. Cela permet de calculer un nouveau chemin (ou trajectoire) et ce processus est répété jusqu'à ce qu'un optimum local soit trouvé.

La méthode proposée permet de prendre en compte la dynamique du véhicule ainsi que l'incertitude sur la position en fonction de la disponibilité des capteurs directement dans un modèle POMDP. Cependant, l'utilisation de l'algorithme d'itération sur la valeur permet uniquement de calculer des solutions sous optimales. Néanmoins, la méthode pourrait être reprise dans nos travaux afin de propager l'influence de la disponibilité de capteurs sur l'état de croyance.

## 2.3 Conclusion et Bilan

Dans ce chapitre, nous avons vu des travaux qui ont pris en compte la dynamique d'un véhicule et la disponibilité des capteurs embarqués pour planifier un chemin sûr et efficace par des algorithmes tels que RRBT [ACHTELIK et al. 2014; BRY et al. 2011] et A\* [WATANABE, DESSUS et al. 2014; WATANABE, VEILLARD et al. 2016]. Cependant, ces algorithmes ne peuvent pas intégrer la stochasticité de l'environnement telle la disponibilité probabiliste des capteurs embarqués que nous considérons dans cette thèse. Les POMDP ont pour objectifs de permettre de résoudre des problèmes de planification dans l'incertain sous observabilité partielle. Malgré les avancées scientifiques, la résolution reste encore difficile et en particulier pour des problèmes avec de grands espaces d'états et d'actions, ce qui est le cas des applications robotiques. [BAI, CAI et al. 2015] a proposé un planificateur basé sur deux niveaux pour réduire la dimension du problème. Cette architecture permet de réduire la complexité en résolvant deux sous-problèmes, mais ne permet pas d'obtenir des chemins dynamiques en fonction de la position des personnes dans la foule. Et donc la solution devient sous optimale. À notre connaissance peu d'approches ont considéré dans le même problème :

1. l'intégration du modèle de mouvement du véhicule en boucle fermée (GNC) dans un modèle POMDP. L'intégration du module GNC dans le modèle de planification est une piste que nous avons choisi d'explorer dans cette thèse afin de propager l'influence de la disponibilité de capteurs sur l'état de croyance.
2. la prise en compte de la dégradation de la précision et/ou la disponibilité des capteurs en fonction de la position du véhicule dans l'environnement pendant la planification, la disponibilité de chaque capteur pouvant être considérée comme probabiliste en fonction de l'environnement, ce que les algorithmes de planification de chemins déterministes ne peuvent pas prendre en compte.

Dans cette thèse, nous avons choisi de nous attaquer à ces deux enjeux qui ont été peu traités dans la littérature. Tout d'abord, dans le chapitre 3 nous définirons un modèle de mouvement de véhicule en bouclant avec le module GNC qui servira pour le reste de cette thèse. Ensuite dans le chapitre 4, nous présenterons une première modélisation du problème sous la forme d'un MOMDP. Ce MOMDP intégrera le modèle de mouvement du véhicule avec le module GNC dans la fonction de transition ceci afin de s'attaquer au premier problème évoqué précédemment. De plus, nous intégrons dans ce modèle un ensemble de connaissance a priori sur la disponibilité probable pour chaque capteur embarqué. Finalement dans le chapitre

5, nous proposons une autre approche qui prend en compte les limitations de la première afin de résoudre plus efficacement le problème particulier de planification de chemins que nous étudions.

## Partie II

Planification de chemins efficaces et sûrs  
par la prise en compte de la dégradation de  
la précision de la localisation et de la  
disponibilité des capteurs



# 3

## Modèle de transition GNC pour représenter le mouvement d'un véhicule

### Sommaire

---

<b>3.1</b>	<b>Définition de l'architecture du système</b>	<b>43</b>
<b>3.2</b>	<b>Modèle de transition GNC</b>	<b>44</b>
3.2.1	Modèle de transition d'état du véhicule	44
3.2.2	L'estimateur d'état du module de navigation	45
3.2.3	Le module de guidage	47
3.2.4	La fonction de densité de probabilité d'état	48
3.2.5	Fréquence de fonctionnement	50
<b>3.3</b>	<b>Bilan</b>	<b>50</b>

---

Ce chapitre a pour objectif de présenter un modèle de mouvement d'un drone qui nous servira de cas pratique étudié pour le problème de planification de chemin sûr en minimisant le temps de vol. L'objectif d'intégrer le modèle de mouvement du véhicule en boucle fermée avec le module du GNC présenté dans ce chapitre dans un planificateur est d'être capable de propager l'influence de la disponibilité des capteurs et l'incertitude sur l'erreur de localisation ainsi que l'erreur d'exécution de la trajectoire.

### 3.1 Définition de l'architecture du système

Comme il a été déjà mentionné, la planification de chemin pour des véhicules autonomes est un domaine de recherche très actif [BRECHTEL et al. 2013; BRY et al. 2011]. Cependant la sécurité du chemin dépend fortement de la précision et de la disponibilité des capteurs embarqués [WATANABE, DESSUS et al. 2014; ACHTELIK et al. 2014]. Le modèle de transition GNC peut être représenté par un système composé de plusieurs parties comme le modèle de mouvement du véhicule, les capteurs embarqués et le système de guidage, navigation et contrôle de vol qui est difficilement intégrables directement dans un modèle POMDP.

Le module de navigation permet d'estimer l'état dont la position du véhicule en utilisant les mesures reçues des capteurs. Le nombre de capteurs embarqués dépend des cas d'applications. Dans nos travaux nous considérons qu'il y a au minimum une centrale inertielle embarquée (IMU) dont la disponibilité et la précision ne dépendent pas de l'environnement. Par contre, ses mesures d'accélération et de vitesse angulaire sont biaisées. Ce biais d'accélération entraîne une dérive dans l'estimation de la vitesse et de la position. Il est nécessaire d'utiliser d'autres capteurs tels que le GPS pour corriger le biais de l'IMU. On définit le mode de navigation par le choix de capteurs à utiliser pour cette correction. Ce mode de navigation devra être un des éléments à spécifier dans notre modèle de planification.

L'état estimé par le module de navigation est utilisé par le module de guidage (pilotage) afin de calculer des commandes pour réaliser la trajectoire. C'est pourquoi l'incertitude sur la localisation influence directement la trajectoire exécutée.

## 3.2 Modèle de transition GNC

Comme expliqué précédemment, le modèle de transition GNC du système est constitué du modèle de mouvement du véhicule, du modèle des capteurs embarqués et du système de guidage, navigation et contrôle. Dans cette section, nous présenterons le modèle développé pour un drone et utilisé dans les travaux de cette thèse.

### 3.2.1 Modèle de transition d'état du véhicule

Dans cette thèse, pour simplifier, nous considérons uniquement le mouvement translationnelle du véhicule en supposant que son altitude est contrôlée à part en fonction du chemin choisi. L'état du drone noté  $x$  à estimer est défini par la position, la vélocité dans le repère inertiel local et le biais de l'accéléromètre dans le repère du drone :

$$x = \begin{bmatrix} \mathcal{X} \\ \mathcal{V} \\ b_a \end{bmatrix} \quad (3.1)$$

Nous modélisons le mouvement du véhicule en fermant la boucle avec le module contrôlé par une cinématique translationnelle avec une accélération en entrée :

$$x_{k+1} = \Phi x_k + B a_k + v_{k+1} \quad (3.2)$$

où  $a_k$  est l'accélération et  $v_k \sim N(0, Q)$  le bruit du processus de transition, qui modélise

l'erreur lors de la transition et

$$\Phi = \begin{bmatrix} I & \Delta t I & 0 \\ 0 & I & 0 \\ 0 & 0 & I \end{bmatrix}, B = \begin{bmatrix} \frac{\Delta t^2}{2} I \\ \Delta t I \\ 0 \end{bmatrix}$$

L'équation (3.2) définit la densité de probabilité  $p(x_{k+1}|x_k, a_k)$  que le véhicule se trouve dans l'état  $x_{k+1}$  depuis l'état  $x_k$  et ayant reçu le vecteur de contrôle (l'accélération)  $a_k$ . Cette densité de probabilité est obtenue comme une loi normale multidimensionnelle :

$$X_{k+1} \sim \mathcal{N}(\Phi x_k + B a_k, Q) \quad (3.3)$$

### 3.2.2 L'estimateur d'état du module de navigation

La valeur réelle de l'état du véhicule n'est pas accessible, elle doit donc être estimée par le module de navigation en utilisant les mesures des capteurs embarqués disponibles. L'estimateur considéré est basé sur le filtre de Kalman (étendu) et fonctionne en deux étapes :

1. La prédiction avec la mesure de l'accéléromètre
2. La correction avec la mesure d'un autre capteur (si disponible)

Nous définissons :

$\hat{x}_k^-$	l'état prédit à temps k
$\tilde{x}_k^- = x_k - \hat{x}_k^-$	l'erreur de l'état prédit
$P_k^- = E[\tilde{x}_k^- \tilde{x}_k^{-T}]$	la covariance de l'erreur de prédiction
$\hat{x}_k$	l'état estimé à temps k
$\tilde{x}_k = x_k - \hat{x}_k$	l'erreur de l'état estimé
$P_k = E[\tilde{x}_k \tilde{x}_k^T]$	la covariance de l'erreur d'estimation

#### 3.2.2.1 Prédiction avec l'accéléromètre

La mesure de l'accéléromètre  $a_{\text{IMU}_k}$  est utilisée pour propager l'état estimé de  $k$  à  $k+1$ . Cette mesure est l'accélération non gravitationnelle biaisée et bruitée dans le repère du drone :

$$a_{\text{IMU}_k} = R_{BI_k}(a_k - g) + b_{a_k} + \xi_{\text{IMU}_k} \quad (3.4)$$

où  $R_{BI_k}$  est une matrice de rotation du repère de l'inertie au repère du drone, que nous assumons connu par l'estimateur d'altitude,  $g$  le vecteur de gravité et  $\xi_{\text{IMU}} \sim N(0, R_{\text{IMU}})$  est l'erreur sur la mesure de l'accéléromètre.

Par l'équation (3.4), l'accélération réelle dans le repère inertiel local est décrite de la façon

suivante :

$$a_k = R_{BI_k}^T (a_{IMU_k} - b_{a_k} - \xi_{IMU_k}) + g \quad (3.5)$$

L'accélération estimée  $\hat{a}_k$  est calculée avec la mesure  $a_{IMU}$  et le biais estimé  $\hat{b}_a$  :

$$\hat{a}_k = R_{BI_k}^T (a_{IMU_k} - \hat{b}_{a_k}) + g \quad (3.6)$$

En se basant sur la fonction de transition d'état (Éq. 3.2), l'état estimé noté  $\hat{x}_k$  à un pas de temps  $k$  est propagé de la façon suivante :

$$\begin{aligned} \hat{x}_{k+1}^- &= \Phi \hat{x}_k + B \hat{a}_k \\ &= \Phi \hat{x}_k + B \left( R_{BI_k}^T (a_{IMU_k} - \hat{b}_{a_k}) + g \right) \end{aligned} \quad (3.7)$$

Alors l'erreur sur la prédiction de l'état est donnée par :

$$\begin{aligned} \tilde{x}_{k+1}^- &= x_{k+1} - \hat{x}_{k+1}^- \\ &= (\Phi x_k + B a_k + v_{k+1}) - (\Phi \hat{x}_k + B \hat{a}_k) \\ &= \Phi (x_k - \hat{x}_k) + B (a_k - \hat{a}_k) + v_{k+1} \\ &= \Phi \tilde{x}_k - B R_{BI_k}^T (\tilde{b}_{a_k} + \xi_{IMU_k}) + v_{k+1} \\ &= (\Phi - \Delta \Phi_k^a) \tilde{x}_k + v_{k+1} - B R_{BI_k}^T \xi_{IMU_k} \end{aligned} \quad (3.8)$$

avec  $\Delta \Phi_k^a = B R_{BI_k}^T \begin{bmatrix} 0 & 0 & I \end{bmatrix}$ .

Puis on calcule la covariance de l'erreur associée :

$$P_{k+1}^- = (\Phi - \Delta \Phi_k^a) P_k (\Phi - \Delta \Phi_k^a)^T + Q + \tilde{R}_{IMU_k} \quad (3.9)$$

où,  $\tilde{R}_{IMU_k} = B R_{BI_k}^T R_{IMU} R_{BI_k} B^T$ . Cependant, pour simplifier nous considérons le cas où  $R_{IMU} = \sigma_{IMU}^2 I$  afin que  $\tilde{R}_{IMU} = B R_{IMU} B^T$  reste constant pour tout  $k$ .

### 3.2.2.2 Correction avec d'autres capteurs embarqués

Comme il a été expliqué précédemment, il est possible de corriger le biais dans la mesure de la centrale inertielle et limiter la dérive de l'état estimé en utilisant d'autres capteurs. Quand le  $n$ -ième capteur  $S_n$  embarqué est disponible à  $t_{k+1}$ , l'état prédit (Éq. 3.7) peut être corrigé en utilisant la mesure  $z_{S_{n,k+1}}$  du capteur tel que le GPS.

$$z_{S_{n,k+1}} = h_{S_n}(x_{k+1}) + \xi_{S_{n,k+1}} \quad (3.10)$$



où  $\xi_{S_n}$  est le bruit sur la mesure  $\xi_{S_n} \sim N(0, R_{S_n})$ , qui est spécifique au capteur embarqué. Alors, la mesure du capteur peut être utilisée pour corriger l'état prédit :

$$\begin{aligned}\hat{x}_{k+1} &= \hat{x}_{k+1}^- + K_{S_{n_{k+1}}} \left( z_{S_{n_{k+1}}} - h_{S_n}(\hat{x}_{k+1}^-) \right) \\ &= \hat{x}_{k+1}^- + K_{S_{n_{k+1}}} \left( h_{S_n}(x_{k+1}) - h_{S_n}(\hat{x}_{k+1}^-) + \xi_{S_{n_{k+1}}} \right) \\ &\simeq \hat{x}_{k+1}^- + K_{S_{n_{k+1}}} \left( H_{S_{n_{k+1}}} \tilde{x}_{k+1}^- + \xi_{S_{n_{k+1}}} \right)\end{aligned}\quad (3.11)$$

où  $H_{S_{n_{k+1}}}$  est la matrice de mesure :

$$H_{S_{n_{k+1}}} = \left. \frac{\partial h_{S_n}(x)}{\partial x} \right|_{x=\hat{x}_{k+1}^-} \quad (3.12)$$

Le gain de Kalman  $K_{S_{n_{k+1}}}$  sur le résidu de la mesure est le gain qui minimise la covariance d'erreur d'estimation  $P_{k+1}$ . Il est donnée par  $K_{S_{n_{k+1}}} = P_{k+1}^- H_{S_n, k+1}^T \left( H_{S_n, k+1} P_{k+1}^- H_{S_n, k+1}^T + R_{S_n} \right)^{-1}$ . L'erreur estimée et la covariance associée devient :

$$\begin{aligned}\tilde{x}_{k+1} &= \left( I - K_{S_{n_{k+1}}} H_{S_n, k+1} \right) \tilde{x}_{k+1}^- - K_{S_{n_{k+1}}} \xi_{S_{n_{k+1}}} \\ P_{k+1} &= \left( I - K_{S_{n_{k+1}}} H_{S_n, k+1} \right) P_{k+1}^- \end{aligned}\quad (3.13)$$

Il suffit ainsi de concaténer le vecteur de mesures (3.10), si plusieurs capteurs sont disponibles en même temps. En revanche, si aucun des capteurs embarqués n'est disponible alors l'estimation de l'état à  $t_{k+1}$  est celui prédit :

$$\begin{aligned}\hat{x}_{k+1} &= \hat{x}_{k+1}^- \\ \tilde{x}_{k+1} &= \tilde{x}_{k+1}^- \\ P_{k+1} &= P_{k+1}^- \end{aligned}\quad (3.14)$$

### 3.2.3 Le module de guidage

Un ensemble d'actions sera défini dans le modèle de planification de chemin, de telle façon que chaque action représente la direction de déplacement désirée pour le drone. Étant donnée une direction désirée  $u_{ref}$  et une vitesse de référence  $\mathcal{V}_{ref}$ , la loi de guidage linéaire suivante peut être appliquée pour obtenir la consigne de guidage pour l'accélération :

$$\mathcal{V}_{ref} = \mathcal{V}_{ref} u_{ref} \quad (3.15)$$

Avec la vitesse de référence  $\mathcal{V}_{ref}$  :

$$a_k = -K_d(\hat{\mathcal{V}}_k - \mathcal{V}_{\text{ref}}) \quad (3.16)$$

où  $K_d > 0$  est le gain du contrôle et  $\hat{\mathcal{V}}_k$  la vitesse estimée du véhicule à chaque instant  $t_k$ .

Le prochain état  $x_{k+1}$  peut être obtenu en insérant la loi de guidage (Éq. 3.16) dans la fonction de transition du modèle de transition d'état (Éq. 3.2).

$$\begin{aligned} x_{k+1} &= \Phi x_k + B(-K_d \begin{bmatrix} 0 & I & 0 \end{bmatrix} \hat{x}_k) + v_{k+1} \\ &= (\Phi - B\tilde{K}_d)x_k + B\tilde{K}_d\tilde{x}_k + v_{k+1} \\ &= (\Phi - \Delta\Phi^{\mathcal{V}})x_k + \Delta\Phi^{\mathcal{V}}\tilde{x}_k + v_{k+1} \end{aligned} \quad (3.17)$$

avec  $\Delta\Phi^{\mathcal{V}} = BK_d \begin{bmatrix} 0 & I & 0 \end{bmatrix}$  et  $\tilde{x}_k = x_k - \hat{x}_k$  l'erreur d'estimation de l'état à chaque instant  $t_k$ . Nous obtenons alors que  $x_{k+1}$  suit une distribution normale :

$$\begin{aligned} x_{k+1} &\sim N((\Phi - \Delta\Phi^{\mathcal{V}})x_k, \Delta\Phi^{\mathcal{V}}P_k\Delta\Phi^{\mathcal{V}T} + Q) \\ &= N(\bar{x}_{k+1}^a, \tilde{Q}_{k+1}) \end{aligned} \quad (3.18)$$

où la matrice de covariance  $\tilde{Q}_{k+1}$  est une fonction de l'estimation de covariance  $P_k$  calculée dans le module de navigation.

### 3.2.4 La fonction de densité de probabilité d'état

Étant donné un état initial  $x(t_0) = x_0$  et la covariance initiale des erreurs d'estimation  $P_0$ , nous avons  $X_0 \sim N(x_0, P_0)$ . Pour une direction, un mode de navigation, il est possible d'obtenir la distribution sur le prochain état  $x_1$  sachant  $x_0$  :

$$p(x_1|x_0, a) \sim N(\bar{x}_{1|0}^a, \tilde{Q}_1) \quad (3.19)$$

En même temps, la covariance de l'erreur sur l'état estimé  $P_1$ , est mise à jour en utilisant le mode de navigation (capteur embarqué) sélectionné. La distribution conditionnelle de l'état à  $t_k$  ( $k > 1$ ) sachant  $x_0$  et en répétant la même action, peut être obtenue itérativement de la façon suivante.

$$p(x_k|x_0, a) = \int p(x_k|x_{k-1}, a)p(x_{k-1}|x_0, a)dx_{k-1} \sim N(\bar{x}_{k|0}, \Sigma_k) \quad (3.20)$$

où  $p(x_k|x_{k-1}, a) \sim N(\bar{x}_k|x_{k-1}^a, \tilde{Q}_k)$ . En parallèle, le processus de filtrage de Kalman du mode de navigation choisi est répété  $k$  fois pour obtenir  $P_k$ . En faisant l'hypothèse que  $\tilde{Q}_k$  et  $P_k$  ne dépendent pas de l'état  $x_{k-1}$ , l'intégration ci-dessus peut être ré-écrite par une distribution

normale (la notation  $a$  est omis) :

$$\begin{aligned}
 p(x_k|x_0) &= \int p(x_k|x_{k-1})p(x_{k-1}|x_0)dx_{k-1} \\
 &= \frac{1}{(2\pi)^N |\tilde{Q}_k|^{\frac{1}{2}} |\Sigma_{k-1}|^{\frac{1}{2}}} \int_{x_{k-1}} \exp\left(-\frac{1}{2}\left((x_k - \bar{x}_{k|k-1})^T \tilde{Q}_k^{-1}(x_k - \bar{x}_{k|k-1}) \right. \right. \\
 &\quad \left. \left. + (x_{k-1} - \bar{x}_{k-1|0})^T \Sigma_{k-1}^{-1}(x_{k-1} - \bar{x}_{k-1|0})\right)\right) dx_{k-1} \\
 &= \frac{1}{(2\pi)^N |\tilde{Q}_k|^{\frac{1}{2}} |\Sigma_{k-1}|^{\frac{1}{2}}} \int_{x_{k-1}} \exp\left(-\frac{1}{2}\left((x_k - (\Phi - \Delta\Phi^V)x_{k-1})^T \tilde{Q}_k^{-1}(x_k - (\Phi - \Delta\Phi^V)x_{k-1}) \right. \right. \\
 &\quad \left. \left. - \frac{1}{2}\left((x_{k-1} - \bar{x}_{k-1|0})^T \Sigma_{k-1}^{-1}(x_{k-1} - \bar{x}_{k-1|0})\right)\right)\right) dx_{k-1} \\
 &= \frac{1}{(2\pi)^N |\tilde{Q}_k|^{\frac{1}{2}} |\Sigma_{k-1}|^{\frac{1}{2}}} \int_{x_{k-1}} \exp\left(-\frac{1}{2}\left((x_{k-1} - (\Phi - \Delta\Phi^V)^{-1}x_k)^T (\Phi - \Delta\Phi^V) \right. \right. \\
 &\quad \tilde{Q}_k^{-1}(\Phi - \Delta\Phi^V)(x_{k-1} - (\Phi - \Delta\Phi^V)^{-1}x_k) \\
 &\quad \left. \left. - \frac{1}{2}\left((x_{k-1} - \bar{x}_{k-1|0})^T \Sigma_{k-1}^{-1}(x_{k-1} - \bar{x}_{k-1|0})\right)\right)\right) dx_{k-1} \\
 &= \frac{1}{(2\pi)^N |\tilde{Q}_k|^{\frac{1}{2}} |\Sigma_{k+1}|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}\left((\Phi - \Delta\Phi^V)^{-1}x_k - x_{k-1|0}\right)^T \right. \\
 &\quad \left. ((\Phi - \Delta\Phi^V)^{-1}\tilde{Q}_k((\Phi - \Delta\Phi^V)^T + \Sigma_{k-1})^{-1}((\Phi - \Delta\Phi^V)^{-1}x_k - x_{k-1|0})\right) \\
 &\quad \int_{x_{k-1}} \exp\left(-\frac{1}{2}(x_{k-1} - \mu)^T P^{-1}(x_{k-1} - \mu)\right) dx_{k-1} \\
 &= \frac{|P|^{\frac{1}{2}}}{(2\pi)^{\frac{N}{2}} |\tilde{Q}_k|^{\frac{1}{2}} |\Sigma_{k+1}|^{\frac{1}{2}}} \exp\left(\frac{1}{2}(x_k - \bar{x}_{k|0})^T \Sigma_k^{-1}(x_k - \bar{x}_{k|0})\right) \\
 &= \frac{1}{(2\pi)^{\frac{N}{2}} |\Sigma_k|^{\frac{1}{2}}} \exp\left(\frac{1}{2}(x_k - \bar{x}_{k|0})^T \Sigma_k^{-1}(x_k - \bar{x}_{k|0})\right) \\
 &= N(\bar{x}_{k|0}, \Sigma_k), \quad k > 1
 \end{aligned} \tag{3.21}$$

avec :

$$\begin{aligned}
 P^{-1} &= (\Phi - \Delta\Phi^V)^T \tilde{Q}_k^{-1}(\Phi - \Delta\Phi^V) + \Sigma_{k-1}^{-1} \\
 \mu &= P((\Phi - \Delta\Phi^V)^T \tilde{Q}_k^{-1}x_k + \Sigma_{k-1}^{-1}\bar{x}_{k-1|0}) \\
 \bar{x}_{k|0} &= (\Phi - \Delta\Phi^V)\bar{x}_{k-1|0} \\
 \Sigma_k &= (\Phi - \Delta\Phi^V)\Sigma_{k-1}(\Phi - \Delta\Phi^V)^T + \tilde{Q}_k
 \end{aligned} \tag{3.22}$$

où  $\bar{x}_{1|0} = (\Phi - \Delta\Phi^V)x_0$ ,  $\tilde{\Sigma}_1 = \tilde{Q}_1$  et  $\tilde{Q}_k$  est obtenue par (Éq. 3.18).

Étant donné que cette dérivation de la distribution de l'état devient plus compliquée dans le cas où l'on a une dépendance de  $\tilde{Q}_k$  et donc  $P_k$  avec l'état  $x_{k-1}$  (par exemple en cas de mesures non linéaire à l'état), nous avons alors considéré que ces matrices peuvent être

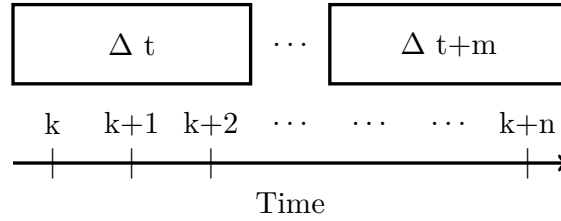


FIGURE 3.1 – Représentation de la différence de la fréquence de fonctionnement entre le système GNC et le planificateur.

approximées par celles qui ont été évaluées à l'état espéré  $\bar{x}_{k-1|0}$ .

### 3.2.5 Fréquence de fonctionnement

Le modèle de transition GNC cherche à représenter le mouvement du véhicule en boucle fermée pour lequel nous souhaitons planifier un chemin. Le module GNC embarqué fonctionne avec une fréquence élevée. Cependant, pour un planificateur basé sur les POMDP, par exemple il n'est pas souhaitable qu'il ait une fréquence de fonctionnement élevé. Car, pour ce problème de planification, ceci implique d'augmenter l'horizon pour atteindre l'objectif et se confronter davantage au problème de la malédiction de la dimension [BELLMAN 1957]. Il est donc contre-productif de faire fonctionner le planificateur à la même fréquence que le module GNC, car ceci augmente très fortement la complexité de la résolution.

Nous considérons alors que le système de transition GNC présenté dans ce chapitre fonctionne à une fréquence plus élevée que le planificateur. Pour différencier les unités de temps entre le système de transition GNC et le planificateur, nous noterons une unité de temps du GNC  $k$  et une *époque* du planificateur  $t$ . Nous considérons qu'une époque du planificateur représente plusieurs unités de temps du système de transition du GNC (Figure 3.1).

## 3.3 Bilan

Dans ce chapitre, nous avons présenté un modèle de transition GNC pour un ensemble de capteurs embarqués choisi par le mode de navigation. Nous avons dérivé l'influence du choix du capteur sur l'erreur d'exécution de la trajectoire  $\Sigma_k$ . Dans le chapitre suivant, nous présenterons un premier modèle de planification basé sur les MOMDP, afin de résoudre le problème de planification de chemin sûr en minimisant le temps de vol en considérant ce modèle de transition GNC dans le modèle de planification.

# 4

## Proposition d'un modèle MOMDP et des algorithmes de résolution avec représentation d'état de croyance par des modèles de mélange gaussiens

### Sommaire

---

<b>4.1</b>	<b>Connaissances a priori</b>	<b>52</b>
<b>4.2</b>	<b>Modèle de planification basé MOMDP</b>	<b>54</b>
4.2.1	Le modèle de planification MOMDP proposé	54
4.2.2	Représentation de l'état de croyance par un modèle de mélange gaussien	59
4.2.3	Fonction de coût	63
4.2.4	Fonction de valeur	64
<b>4.3</b>	<b>Algorithmes de résolution</b>	<b>65</b>
4.3.1	Initialisation de la fonction de valeur	65
4.3.2	LRTDP-bel	66
4.3.3	MCBTS	69
<b>4.4</b>	<b>Expérimentations</b>	<b>71</b>
4.4.1	Setup des expérimentations	71
4.4.2	Résultats	72
<b>4.5</b>	<b>Bilan</b>	<b>76</b>

---

Dans ce chapitre, nous allons présenter la première approche proposée pour résoudre le problème de planification de chemin sûr avec minimisation du temps de trajet pour des drones dans des milieux encombrés. D'une part, nous proposons un modèle se basant sur les MOMDP [S. C. ONG et al. 2010], qui sont une extension des POMDP. L'utilisation d'un MOMDP permet de prendre en compte la dynamique de l'environnement et ainsi gérer l'incertitude sur la disponibilité des capteurs tout en diminuant le temps de calcul de la mise à jour de l'état de croyance en factorisant l'espace d'état avec les variables non observables et les variables totalement observables. D'autre part, nous avons choisi de représenter les états de croyances par des modèles de mélange gaussien, afin d'exploiter les caractéristiques du modèle de transition GNC. Cette représentation permet d'utiliser la fonction de coût déjà proposée par [WATANABE, DESSUS et al. 2014], qui optimise à la fois l'incertitude sur la position et le temps de vol sans favoriser un de ces deux critères. Nous verrons qu'avec cette fonction de coût, nous n'obtenons plus une fonction de valeur linéaire. Cela nous oblige donc à utiliser et à développer des algorithmes de résolution qui n'exploitent pas cette particularité de la fonction de valeur.

#### 4.1 Connaissances a priori

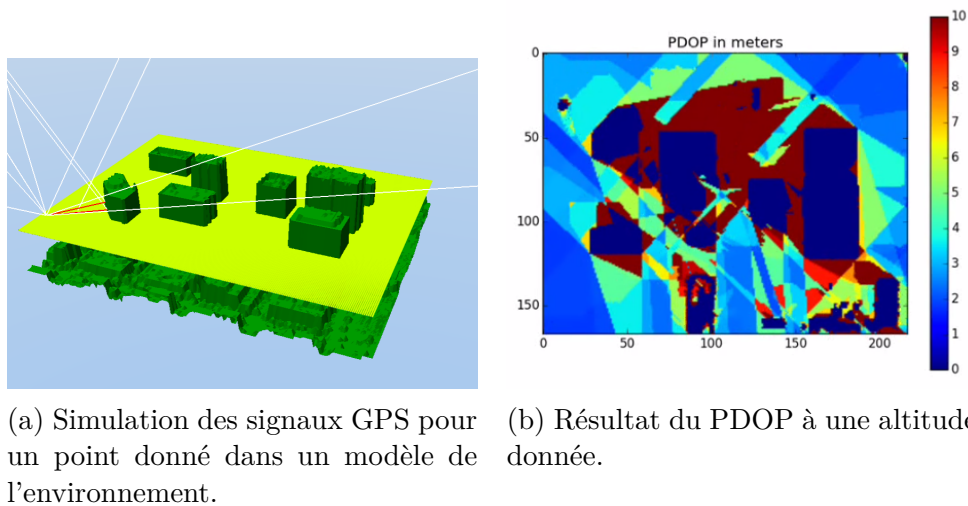


FIGURE 4.1 – Exemple du calcul du PDOP du GPS pour un environnement donné.

Pour pouvoir prendre en compte lors de la planification la disponibilité a priori de capteurs, il est nécessaire d'avoir une information sur cette disponibilité en amont. La disponibilité des capteurs peut dépendre de la position du véhicule dans l'environnement. En effet, les capteurs de localisation tels que le GPS peuvent subir une dégradation de la précision près des bâtiments due à l'effet de multi-trajet et/ou l'occlusion des signaux. D'autres capteurs, notamment visuels, ont besoin d'amères visibles et reconnaissables. La détection des amères

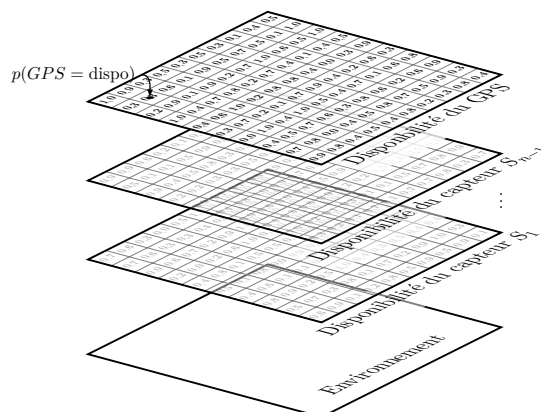


FIGURE 4.2 – Représentation de l'ensemble des cartes de probabilités sur la disponibilité, ainsi d'une carte de l'environnement.

est sensible à la texture et de l'environnement et aux conditions de luminosité.

Cependant, il est possible pour ces capteurs d'anticiper la dégradation de la précision si l'environnement est connu. Par exemple, pour le GPS la dégradation de la précision est défini par une métrique appelée PDOP (Position Dilution of Precision), qui correspond à l'écart type de l'erreur espéré sur la position mesurée. Il est possible de prédire cette dilution de la précision avec un simulateur GPS pour une géolocalisation, une date, une heure et un environnement spécifique. La figure 4.1a montre la simulation des signaux de chaque satellite GPS pour calculer la dilution de précision pour un point de l'environnement. La figure 4.1b est un exemple de la carte PDOP résultant de ces simulations pour un ensemble des points à une altitude. Il est alors possible d'utiliser cette carte pour générer la carte de disponibilité du GPS sous forme d'une grille notée  $G$  en fixant un seuil d'erreur maximum acceptable  $\epsilon$ . Succinctement, le GPS est considéré comme étant disponible si l'erreur sur la position  $\tilde{\mathcal{X}}_{\text{GPS}}$  est inférieure à ce seuil. En partant de la valeur donnée sur la PDOP, une probabilité de disponibilité du GPS  $p(\text{GPS}|c_i)$  pour chaque cellule  $c_i \in G$  peut être calculée par :

$$p(\text{GPS}|c_i) = p(\|\tilde{\mathcal{X}}_{\text{GPS}}(c_i)\| < \epsilon), \quad \text{étant donné } \tilde{\mathcal{X}}_{\text{GPS}}(c_i) \sim N(0, PDOP^2(c_i)) \quad (4.1)$$

Nous obtenons alors une carte sur la probabilité de disponibilité du GPS. Il est possible de calculer une carte sur la probabilité de disponibilité pour certains autres capteurs.

Dans la suite de ces travaux, nous avons considéré que nous disposions d'un ensemble de cartes de probabilités de disponibilité discrétisé sous forme de grille pour chacun des capteurs embarqués, ainsi que d'une carte de l'environnement ou carte d'occupation (Figure 4.2).

## 4.2 Modèle de planification basé MOMDP

Dans ce chapitre, nous présentons un modèle pour notre problème de planification basé sur les MOMDP. L'objectif est de proposer un modèle de planification et un algorithme de résolution qui permet de calculer un chemin sûr, c'est-à-dire sans risque de collision et le plus court en distance/temps. Pour atteindre cet objectif, nous considérons le modèle de transition GNC présenté précédemment (Chapitre 3) pour prendre en compte les connaissances a priori de disponibilités des différents capteurs embarqués et propager ses influences sur la distribution d'état. Ces connaissances a priori peuvent être considérées comme des observations dans un POMDP. Pour cela, nous avons proposé une nouvelle architecture de MOMDP. Nous verrons dans ce chapitre, comment nous avons intégré cette connaissance a priori dans le modèle de planification MOMDP en le liant avec le modèle de transition GNC.

### 4.2.1 Le modèle de planification MOMDP proposé

Nous définissons notre modèle MOMDP comme un tuple  $(\mathcal{S}_v, \mathcal{S}_h, \mathcal{A}, \Omega, \mathcal{T}, \mathcal{O}, \mathcal{C}, b_0, \gamma)$  où :

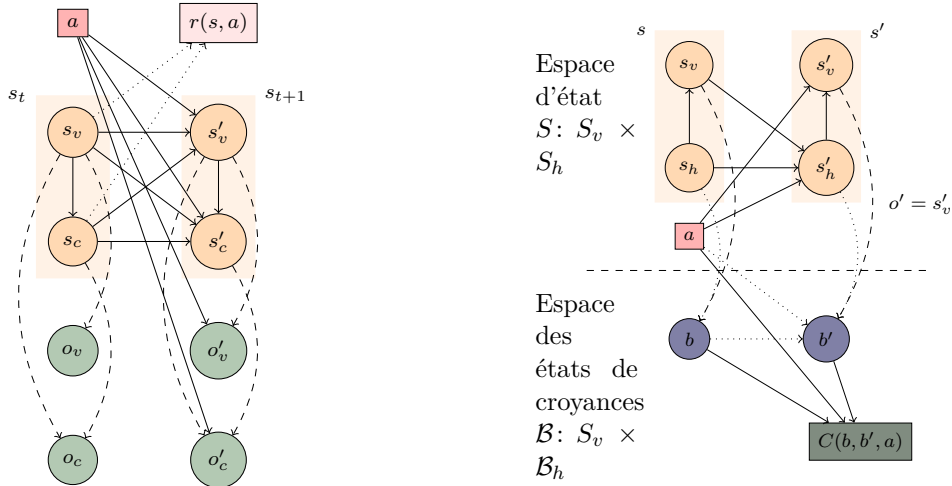
- $\mathcal{S}_v$  est l'espace borné d'états totalement observables.
- $\mathcal{S}_h$  est l'espace borné d'états continus non observables.
- $\mathcal{A}$  est l'ensemble fini des actions
- $\Omega$  est l'ensemble fini des observations
- $\mathcal{T}(s_v, s_h, a, s'_v, s'_h) \rightarrow [0, 1]$  la fonction de transition composée de deux fonctions :
  - $T_{\mathcal{S}_h} : \mathcal{S}_h \times \mathcal{S}_v \times \mathcal{A} \times \mathcal{S}_h \rightarrow [0; 1]$  une fonction de transition :  $T_{\mathcal{S}_h}(s_h, s_v, a, s'_h) = p(s'_h | s_h, s_v, a)$ .
  - $T_{\mathcal{S}_v} : \mathcal{S}_v \times \mathcal{S}_h \rightarrow [0; 1]$  une fonction de transition :  $T_{\mathcal{S}_v}(s'_v, s'_h) = p(s'_v | s'_h)$  ;
- $\mathcal{O} : \Omega \times \mathcal{S}_v \rightarrow [0, 1]$  la fonction d'observation :

$$O(o, s'_v) = p(o | s'_v) = \begin{cases} 1 & o = s'_v \\ 0 & \text{sinon} \end{cases} \quad (4.2)$$

- $\mathcal{C} : \mathcal{B} \times \mathcal{B} \times \mathcal{A} \rightarrow \mathbb{R}$  : la fonction de coût, où  $\mathcal{B}$  est l'espace des états de croyances défini sur  $\mathcal{S} = \mathcal{S}_v \times \mathcal{S}_c$
- $b_0$  l'état de croyance initial  $b_0 = (s_{v_0}, b_{h_0})$

Il est à noter que l'ensemble des observations  $\Omega$  est égal à  $\mathcal{S}_v$  et par conséquent la fonction d'observation  $\mathcal{O}$  est déterministe parce que  $o = s'_v$  (Éq. 4.2). Mais il est important de remarquer que cela n'est pas suffisant pour supprimer l'incertitude sur l'état caché (voir schéma Figure





(a) Modèle de transition d'un MOMDP [ARAYA-LÓPEZ et al. 2010]. (b) Modèle de transition du MOMDP proposé.

FIGURE 4.3 – Différence entre un des modèles des MOMDP classiques et le modèle du MOMDP proposé et utilisé dans ces travaux.

4.3b), car la dépendance Bayésienne dans notre modèle change par rapport aux modèles proposés par [S. C. ONG et al. 2010] et [ARAYA-LÓPEZ et al. 2010] (Figure 4.3a). Notamment, l'état visible  $s'_v$  dépend de l'état caché  $s'_h$ , alors que  $s'_h$  dépend de  $s_v$  et  $s_h$ . Donc, étant donné  $p(o|s'_v)$ , nous supposons que cette information  $o = s'_v$  permettrait de réduire l'incertitude sur état caché de façon indirecte.

#### 4.2.1.1 Définition de l'espace d'état

Dans ce modèle MOMDP proposé, l'espace d'état  $\mathcal{S}$  est caractérisé par deux sous-espaces  $\mathcal{S}_v$  et  $\mathcal{S}_h$ , tel que  $|\mathcal{S}| = |\mathcal{S}_v| \times |\mathcal{S}_h|$ . L'espace d'état entier  $\mathcal{S} = \mathcal{S}_v \times \mathcal{S}_h$  est partiellement observable et nous choisissons de factoriser en fonction des variables totalement observables et non observables.

**Définition de l'état visible** L'état visible  $s_v \in \mathcal{S}_v$  est défini comme un tuple contenant les variables d'états booléens totalement observables sur la disponibilité des capteurs, une variable booléenne sur la détection d'une collision et la matrice de covariance sur l'erreur de localisation  $P$  calculé par le filtre de Kalman (Section 3.2.2). Il est nécessaire de conserver cette matrice dans l'état, car elle se propage en fonction de l'action choisie. L'état visible est alors défini de la façon suivante :

$$s_v = \{F_{S_1}, \dots, F_{S_N}, F_{\text{Col}}, P\} \quad (4.3)$$

où  $F_{S_i}$  est la disponibilité du capteur  $S_i$ . Dans le reste de cette thèse, nous supposons que la variable booléenne sur la détection de collision  $F_{\text{Col}}$  est totalement observable par des mesures,

par exemple une force de contact.

$$F_{S_i} = \begin{cases} 0 & \text{Pas disponible} \\ 1 & \text{Disponible} \end{cases} \quad (4.4)$$

$$F_{\text{Col}} = \begin{cases} 0 & \text{Pas collision} \\ 1 & \text{Collision} \end{cases} \quad (4.5)$$

**Définition de l'état caché** Nous définissons les variables d'état continu non observables de la façon suivante :

$$s_h = [\mathcal{X}^T \quad \mathcal{V}^T \quad \beta]^T \quad (4.6)$$

Cette définition correspond à celle de l'état du véhicule  $x$  dans le modèle de transition GNC (Éq. 3.1). Il est nécessaire de conserver (en plus de la position  $\mathcal{X}$ ) la vitesse  $\mathcal{V}$  et le biais  $\beta$  sur l'accélération dans les variables d'état, car elles sont nécessaires pour la fonction de transition et donc pour calculer l'état suivant. Il est à noter que  $\mathcal{X}^T = [x \ y \ z]$  est le vecteur correspondant à la position du véhicule et il est défini sur un espace non observable continu et borné pour le planificateur.

#### 4.2.1.2 Définition de l'espace d'action

Contrairement à l'espace d'état du véhicule, l'espace d'action défini dans ce modèle est discret. Une action  $a \in A$  est définie de la façon suivante :

$$a = (\{\mathcal{V}_{\text{ref}}\}, m_n) \quad (4.7)$$

où  $\mathcal{V}_{\text{ref}}$  est la vitesse de référence dans le module de guidage (Éq. 3.16) et  $\{\mathcal{V}_{\text{ref}}\}$  définit un ensemble fini de  $\mathcal{V}_{\text{ref}}$  possibles.  $m_n \in \mathcal{M}_n$  définit le mode de navigation sélectionné (choix du capteur). Les modes de navigation possibles à une époque  $t$  dépendent des capteurs embarqués disponibles à cette époque.

#### 4.2.1.3 Espace d'observation et fonction d'observation

Comme il a été expliqué, l'ensemble des observations  $\Omega$  est considéré égal à  $\mathcal{S}_v$  et par conséquent la fonction d'observation est déterministe (Éq. 4.2). Contrairement au cas standard d'un POMDP, nous supposons que l'agent ne reçoit aucune observation directe ou même imprécise sur l'état  $s_h$ . En revanche, nous connaissons la dynamique de propagation de l'erreur d'exécution en fonction du choix du mode de navigation. Nous avons fait ce choix, car cela permet d'éviter de considérer les mesures des capteurs (qui ne sont pas accessibles au moment de la planification) au niveau de la planification et donc éviter de travailler avec un espace d'observation (mesures) continue sachant que les POMDP ont une complexité exponentielle en nombre d'observations. De plus, il est possible de raisonner sur les états de

croyances qui représentent l'erreur d'exécution estimée. Cette méthode permet d'avoir un nombre d'observations égales au nombre d'états visibles (capteurs disponibles et matrice  $P$  associée). Même si le problème reste difficile dû aux états  $s_h$  continus, nous réduisons quand même la complexité en réduisant le nombre d'observations possible.

Cependant, il y a une contrepartie à ne pas considérer les mesures des capteurs dans la planification. En effet, c'est toujours un fonctionnement en boucle ouverte et quand on exécute la politique nous ne réduisons jamais l'incertitude.

#### 4.2.1.4 Fonction de transition

Sur la base de la factorisation de l'espace d'état, nous définissons la fonction de transition du MOMDP par deux fonctions :

- $T_{S_h}$  une fonction de transition :

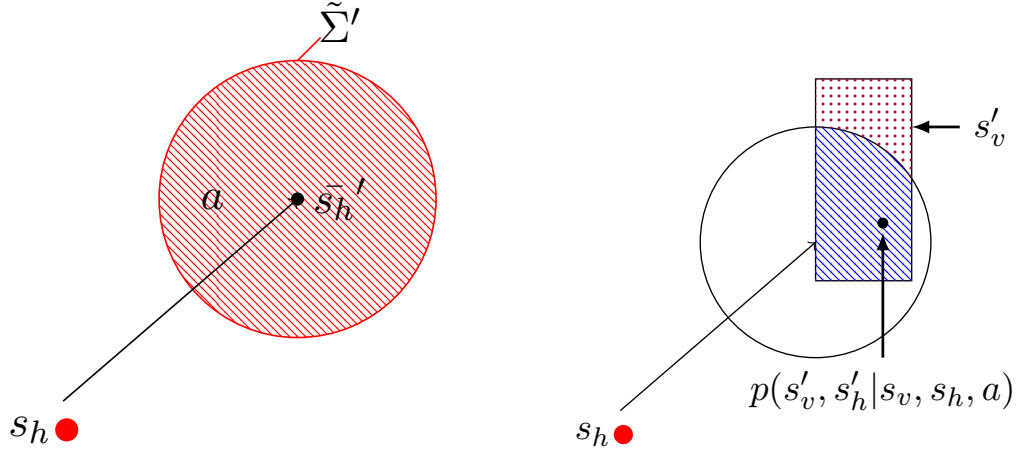
$$T_{S_h}(s_h, s_v, a, s'_h) = p(s'_h | s_h, s_v, a) \sim N(\bar{s}'_h, \tilde{\Sigma}'(s_v)) \quad (4.8)$$

Cette fonction de transition correspond à la fonction de densité de probabilité de l'état définie dans le modèle de transition GNC (Éq. 3.21). Elle permet de calculer la probabilité de passer dans l'état  $s'_h$  depuis l'état précédent  $s = (s_h, s_v)$  en exécutant l'action  $a$ . Pour rappel, une époque de planification représente plusieurs pas de temps du module de transition GNC (Section 3.2.5). La figure 4.4a illustre cette fonction de transition.

- $T_{S_v}$  est la fonction de transition telle que  $T_v(s'_h, s'_v) = p(s'_v | s'_h)$ . Elle représente la transition de  $s'_v$  et dépend des cartes de disponibilité des capteurs et des obstacles. Concrètement  $p(s'_v | s'_h)$  est le produit des probabilités sur la disponibilité (ou non) de chaque capteur embarqué. ce qui nous donne :

$$p(s'_v | s'_h) = p(F'_{Col} | s'_h) \prod_{i=1}^{|S_v \setminus P_{col}|} p(F'_{S_i} | s'_h) \quad (4.9)$$

où  $|S_v \setminus P_{col}|$  est le nombre de capteurs embarqués.  $P$  n'est pas utilisé dans le calcul de  $T_{S_v}$  parce que la transition de  $P$  est déterministe pour un mode de navigation choisi.



(a) Propagation de  $s'_h$  depuis  $s_h$  avec l'action  $a : T_{S_h}$  (b) Réduction du prochain état atteignable en fonction de  $s'_v : T_{S_v}$

FIGURE 4.4 – Illustration des deux fonctions de transitions :  $T_{S_h}$  and  $T_{S_v}$

Il est alors possible de définir la fonction de transition du modèle comme :

$$\begin{aligned}
 T(s_v, s_h, a, s'_v, s'_h) &= T_{S_h}(s_h, s_v, a, s'_h) \times T_{S_v}(s'_h, s'_v) \\
 &= p(s'_v | s'_h) p(s'_h | s_h, s_v, a) \\
 &= p(s'_v, s'_h | s_v, s_h, a)
 \end{aligned} \tag{4.10}$$

La figure 4.4b illustre cette fonction.

#### 4.2.1.5 État de croyance et mise à jour

Dans ce modèle, il est considéré exclusivement les états de croyance atteignables depuis l'état de croyance initial  $b_0 = (s_{v_0}, b_{s_{h_0}})$ . L'état de croyance  $b_{s_h}$  est une fonction de densité de probabilité calculée par le modèle de transition GNC et mise à jour avec l'observation sur les disponibilités capteurs embarqués. Nous considérons que l'état de croyance initial  $b_{s_{h_0}}$  est une loi normale avec un état moyen et une erreur de localisation initiale. Cet état de croyance est délimité par une ellipse de confiance. En factorisant l'espace d'état en accord avec notre modèle, nous obtenons :

$$b_t = (s_{v_t}, b_{s_{h_t}}), \quad \text{avec } b_{s_{h_t}}(s_{h_t}) = p(s_{h_t} | b_{t-1}, s_{v_t}, a_{t-1}), \quad \text{étant donnée } b_0 \tag{4.11}$$

La mise à jour de l'état de croyance se fait après chaque action et observation et en accord avec la fonction de transition et d'observation. La mise à jour se fait à l'aide de deux

sous-fonctions correspondant aux deux fonctions de transitions  $T_{S_h}$  et  $T_{S_v}$ .

1. À partir d'un état de croyance  $b_{s_h}$  et d'une action  $a$ , nous utilisons le modèle de transition du GNC qui va propager l'état de croyance vers le prochain état de croyance.

$$b_a = p(s'_h|b, a) = \int_{s_h \in S_h} p(s'_h|s_h, s_v, a) b(s_h) ds_h \quad (4.12)$$

2. La deuxième fonction est le calcul de la probabilité de  $s'_v$ .

$$\begin{aligned} p(s'_v|b, a) &= \int_{s'_h \in S_h} p(s'_v|s'_h) p(s'_h|b, a) ds'_h \\ &= \sum_{i=0}^{|G|} p(s'_v|s'_h \in c_i) \int_{s'_h \in c_i} p(s'_h|b, a) ds'_h \\ &= \sum_{i=0}^{|G|} p(s'_v|s'_h \in c_i) p(s'_h \in c_i|b, a) \end{aligned} \quad (4.13)$$

où  $c_i$  est la  $i$ -ème cellule de la carte de disponibilité du capteur et  $|G|$  est le nombre de cellules de la carte.

Le nouvel état de croyance  $b_{s'_h}$  est alors calculé en fonction de l'état complètement observable  $s'_v$  de la façon suivante :

$$b_{s'_h}(s'_h) = p(s'_h|b, a, s'_v) = \frac{p(s'_h, s'_v|b, a)}{p(s'_v|b, a)} = \frac{p(s'_v|s'_h) p(s'_h|b, a)}{p(s'_v|b, a)} \quad (4.14)$$

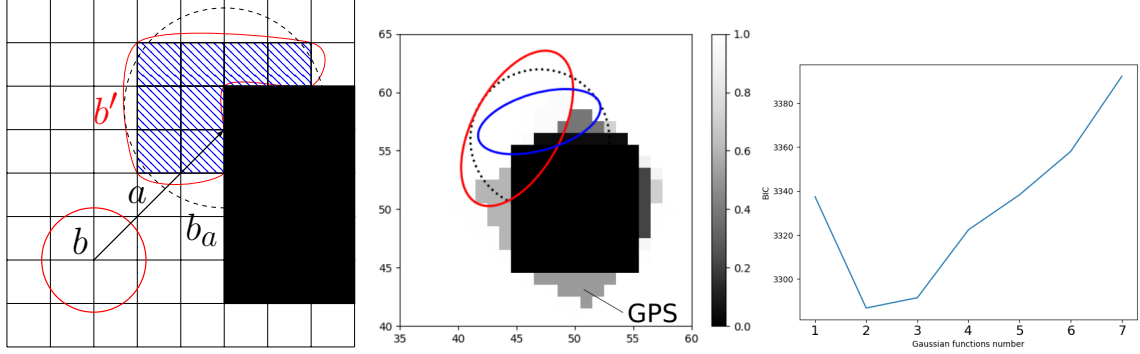
L'état de croyance après mise à jour  $b'$  est défini de la façon suivante :

$$b' = (s'_v, b_{s'_h}) \quad (4.15)$$

## 4.2.2 Représentation de l'état de croyance par un modèle de mélange gaussien

La figure 4.5a représente la mise à jour d'un état de croyance. Le futur état de croyance  $b_a$  est calculé après l'action  $a$  depuis l'état de croyance initiale  $b$ . Si  $b$  est une fonction gaussienne,  $b_a$  calculée et retournée par la fonction de transition GNC est aussi gaussienne. Alors, dans cet exemple (Figure 4.5a), une observation  $s_v$  est perçue telle que la probabilité  $p(s_v|b_a) > 0$  seulement dans les cellules bleues. En utilisant cette observation et la fonction de mise à jour de l'état de croyance (Éq. 4.14), l'état de croyance  $b_a$  est mis à jour pour obtenir  $b'$  (Éq 4.15). La forme de ce nouvel état de croyance n'est plus gaussien dû à la grille de définition des probabilités. Par conséquent, les futurs états de croyance ne seront plus gaussiens.

Cependant, utiliser la représentation des états de croyances représentés par des fonctions



(a) Transition de l'état de croyance  $b$  à  $b'$  pendant la mise à jour. (b) GMM avec  $N_g = 2$  obtenu par apprentissage. (c) Score BIC pour différentes valeurs de  $N_g$ .

FIGURE 4.5 – Exemple de résultat de l'algorithme Expectation Maximization pour l'apprentissage d'un modèle de mélange gaussien pour approximer l'état de croyance  $b'$ .

gaussiennes permet des simplifications algorithmiques. Les propriétés d'une fonction gaussienne permettent à l'algorithme de résolution d'appliquer le modèle de transition, et en particulier cette représentation facilite le calcul de la fonction de coût qui est basée sur l'incertitude (voir section 4.2.3). C'est pourquoi, dans ces travaux nous avons proposé d'utiliser des algorithmes d'apprentissage automatique pour obtenir un modèle de mélange gaussien (*Gaussian Mixture Model* - GMM) pour approximer le nouvel état de croyance  $b'$ .

Dans ce contexte, nous proposons d'appliquer l'algorithme Expectation-Maximization [DEMPSTER et al. 1977] (décrit dans l'Alg. 6) pour apprendre un modèle de mélange gaussien de l'état de croyance mis à jour.

Cet algorithme apprend un GMM avec un nombre donné de fonctions gaussiennes  $N_g$  qui représente au mieux l'état de croyance. Un GMM est défini de la façon suivante :

$$GMM = \sum_{i=1}^{N_g} w(g_i)g_i(s), \quad g_i(s) = N(\mu_i, \Sigma_i) \quad (4.16)$$

où  $g_i$  est une fonction gaussienne.

Dans un GMM, chaque fonction gaussienne a une proportion attribuée  $0 < w(g_i) < 1$  pour  $1 \leq i \leq N_g$ . Le nombre de fonctions gaussiennes  $N_g$  influence la précision de la représentation, mais aussi sa complexité. Plus  $N_g$  est grand, plus la représentation sera précise, mais cela augmente aussi la complexité du modèle. La valeur idéale de  $N_g$  est inconnue, c'est pourquoi il est nécessaire de comparer les GMM appris pour différents  $N_g$  en définissant une métrique.

Tout d'abord, l'algorithme tire aléatoirement un nombre suffisant d'échantillons  $s_h$  (Ligne 2). Un échantillon est un état caché  $s_h$  de l'état de croyance  $b'$ . Après avoir obtenu les échantillons, pour chaque  $N_g$  de 1 à  $N_g^{max}$ , le GMM est initialisé et entraîné (Ligne 5) en utilisant l'algorithme EM. C'est une méthode itérative permettant de trouver une estimation a pos-

teriori de la vraisemblance maximale de modèles statistiques. Il fonctionne en deux étapes : l'étape *Expectation* (E) et l'étape *Maximization* (M). L'étape (E) (Lignes 17 à 21), calcule pour chaque échantillon  $s_h$  la probabilité  $r_{s_h,c}$  que l'échantillon appartienne à la gaussienne  $g_c$  :

$$r_{s_h,c} = \frac{w(g_c) \times g_c(s_h)}{\sum_{i=1}^{N_g} w(g_i) \times g_i(s_h)} \quad (4.17)$$

L'étape (M) (Lignes 22 à 27) consiste à estimer les nouvelles valeurs de chaque gaussienne de la façon suivante :

$$\begin{aligned} m_c &= \sum_{s_h \in S_h} r_{s_h,c} \\ w(g_c) &= \frac{m_c}{m} \\ \mu_{g_c} &= \frac{1}{m_c} \sum_{s_h \in S_h} r_{s_h,c} s_h \\ \Sigma_{g_c} &= \frac{1}{m_c} \sum_{s_h \in S_h} r_{s_h,c} (s_h - \mu_{g_c})^T (s_h - \mu_{g_c}) \end{aligned} \quad (4.18)$$

où  $m_c$  est la somme des probabilités des échantillons que contient  $g_c$ ,  $w(g_c)$  la nouvelle proportion sachant  $m$  la somme des probabilités de tous les échantillons,  $\mu_{g_c}$  et  $\Sigma_{g_c}$  respectivement la nouvelle moyenne et covariance de la fonction gaussienne.

Les étapes sont ensuite répétées jusqu'à un nombre maximum d'itérations fixé ou jusqu'à ce que le gain de vraisemblance  $L$  soit inférieur à un seuil  $\epsilon$ .

Pour décider quel modèle représente le mieux l'état de croyance, nous utilisons la métrique BIC - *Bayes Information Criterion* [SCHWARZ 1978]. En effet, cette métrique a entre autres comme propriété de pénaliser la complexité du modèle appris, c'est-à-dire le nombre de fonctions gaussiennes  $N_g$  par rapport au gain de précision de l'approximation. En effet, le BIC est donné par :

$$BIC = -2 \ln L + N_g \ln(n) \quad (4.19)$$

où  $L$  est la vraisemblance du GMM calculé qui représente la précision d'approximation du modèle appris et  $n$  le nombre d'échantillons. Ainsi, l'algorithme va sélectionner le GMM qui minimise le score BIC (Lignes 6-10). Cependant, le nombre maximum  $N_g^{\max}$  à tester ainsi que le nombre d'échantillons à tirer est dépendant du problème. Il est donc nécessaire de tester empiriquement en amont le nombre le plus approprié. La figure 4.5 montre un exemple de l'apprentissage d'un modèle de mélange gaussien. L'état de croyance  $b_a$  (représenté par un cercle pointillé en noir) est mis à jour après avoir reçu l'observation  $F_{\text{Col}} = 0$  et  $F_{\text{GPS}} = 1$  pour obtenir  $b'$ . L'algorithme 6 est appliqué pour apprendre le modèle qui représentera au mieux ce nouvel état de croyance. L'algorithme retourne un modèle de mélange gaussien composé de

**Algorithme 6 : Expectation-Maximization**


---

```

1 Fonction  $EM(b)$ 
2    $gmm \leftarrow \emptyset$ ;  $bic \leftarrow \infty$ ;  $sample \leftarrow \text{Sampling}(b)$ 
3   for  $i \leftarrow 1$  to  $N_g^{\max}$  by 1 do
4      $gmm\_tmp \leftarrow \text{Initialize}(i)$ 
5      $gmm\_tmp \leftarrow \text{Train}(gmm\_tmp, sample)$ 
6      $bic\_tmp \leftarrow \text{BIC}(gmm\_tmp)$ 
7     if  $bic\_tmp < bic$  then
8        $bic \leftarrow bic\_tmp$ 
9        $gmm \leftarrow gmm\_tmp$ 
10    end
11  end
12  return  $gmm$ 
13 Fonction  $\text{Train}(gmm, samples)$ 
14    $L \leftarrow \infty$ 
15   for  $i \leftarrow 1$  to  $maxIter$  by 1 do
16      $L\_tmp \leftarrow L$ 
17     foreach  $s_h \in samples$  do
18       foreach  $g_c \in gmm$  do
19          $r_{s_h,c} = \frac{w(g_c) \times g_c(s_h)}{\sum_{i=1}^{N_g} w(g_i) \times g_i(s_h)}$ 
20       end
21     end
22     foreach  $g_c \in gmm$  do
23        $m_c = \sum_{s_h \in samples} r_{s_h,c}$ 
24        $w(g_c) = \frac{m_c}{m}$ 
25        $\mu_{g_c} = \frac{1}{m_c} \sum_{s_h \in samples} r_{s_h,c} s_h$ 
26        $\Sigma_{g_c} = \frac{1}{m_c} \sum_{s_h \in S_h} r_{s_h,c} (s_h - \mu_{g_c})^T (s_h - \mu_{g_c})$ 
27     end
28      $L \leftarrow L(samples, gmm)$ 
29     if  $|L - L\_tmp| < \epsilon$  then
30       return  $gmm$ 
31     end
32  end
33  return  $gmm$ 

```

---

deux gaussiennes  $N_g = 2$  (en rouge et en bleu) avec chacune une proportion  $w(g_i)$  respective de 0.42 et 0.58. La figure 4.5c montre qu'un GMM avec  $N_g = 2$  minimise le score BIC parmi tous ceux calculés.



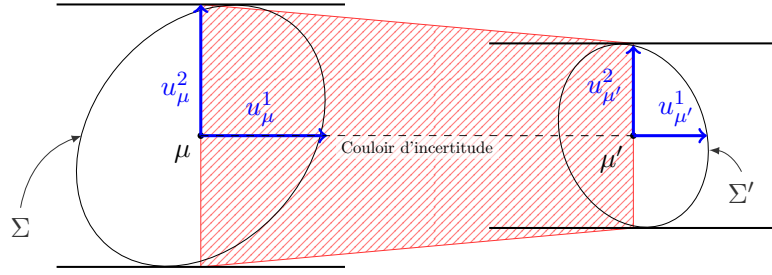


FIGURE 4.6 – Exemple d'un couloir d'incertitude en deux dimensions entre deux états.

### 4.2.3 Fonction de coût

Nous rappelons que l'objectif de ces travaux est de proposer un planificateur de chemin sans risque de collision qui minimise la distance. Pour éviter de favoriser l'un de ces critères d'une façon artificielle, nous avons basé notre fonction de coût sur la notion de couloir d'incertitude proposé dans [WATANABE, VEILLARD et al. 2016](Section 2.1.2). Intuitivement, le couloir d'incertitude est créé par une séquence d'ellipsoïdes de confiance. De cette façon, plus l'incertitude est grande pendant le chemin, plus les ellipsoïdes seront volumineux et le volume du couloir dépendra directement de la longueur du chemin et de l'incertitude sur la position du drone.

Comme les états de croyances sont représentés par des fonctions gaussiennes il est possible d'adapter la notion de couloir d'incertitude proposé par [WATANABE, VEILLARD et al. 2016]. Nous notons  $U(g, g')$  le volume entre deux fonctions gaussiennes sur la position 3D, avec  $g = N(\mu, \Sigma), g' = N(\mu', \Sigma')$  :

$$U(g, g') = \frac{\pi}{6} \|\mu - \mu'\| \cdot \left( u_{\mu'}^2 u_{\mu}^3 + u_{\mu}^2 u_{\mu'}^3 + 2(u_{\mu}^2 u_{\mu'}^3 + u_{\mu'}^2 u_{\mu}^3) \right) \quad (4.20)$$

où  $\|\mu - \mu'\|$  représente la distance euclidienne entre les deux ellipsoïdes approximés à  $3\sigma$ , et  $u^2, u^3$  sont les vecteurs de l'ellipse perpendiculaire au vecteur de direction de  $\mu$  à  $\mu'$  (Figure 4.6).

La fonction de coût entre deux états de croyances est définie de la façon suivante :

$$\mathcal{C}(b, b', a) = K \times F'_{col} + \sum_{g \in b} \sum_{g' \in b'} U(g, g') w(g) w(g'). \quad (4.21)$$

où  $b$  est l'état de croyance qui est un GMM,  $b'$  le prochain état de croyance appris avec l'algorithme EM (Alg. 6) et  $K$  une pénalité en cas de collision. La figure 4.7 illustre le fonctionnement de la fonction de coût de transition entre deux GMM.

Cette fonction de coût est très différente des fonctions de récompense généralement utilisées pour les POMDP. Classiquement, une fonction de récompense est définie de la façon suivante  $R : S \times A \rightarrow \mathbb{R}$ , où  $R(s, a)$  dépend directement de l'état courant  $s$  et de l'action  $a$  effectuée. Ce qui donne une fonction  $R(b, a)$  comme étant la moyenne de sorte que

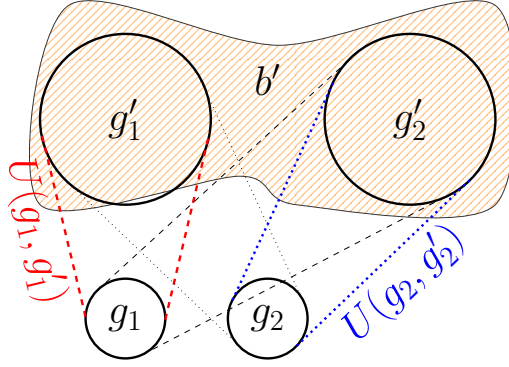


FIGURE 4.7 – Exemple du calcul du coût de transition entre deux états de croyances représentés par un modèle de mélange gaussien.

$R(b, a) = \sum_s R(s, a)b(s)$ . Cette moyenne est une fonction linéaire par rapport à  $b$ . Cette linéarité permet de représenter la récompense d'une action  $a$  par un hyperplan ( $\alpha$ -vecteur) sur l'espace des états de croyance (Chapitre 1). Lors du calcul de la valeur d'un état de croyance, cet hyperplan est projeté par l'opérateur de Bellman, qui est lui-même un opérateur linéaire, résultant ainsi dans un autre hyperplan (Chapitre 1). Quand l'opérateur de maximisation est appliqué sur l'espace d'état de croyance, la fonction de valeur devient PWLC. Cependant, dans notre modèle le coût espéré n'est plus une fonction linéaire et par conséquent la fonction de valeur n'est plus linéaire par morceaux. En plus, la fonction de coût dépend de l'état de croyance  $b$  et de l'action  $a$ , mais aussi de l'état de croyance  $b'$ . Cette différence a pour conséquence d'empêcher l'utilisation d'un nombre important d'algorithmes de la littérature telle que SARSOP [KURNIAWATI, HSU et al. 2008], qui exploitent cette propriété de la fonction de valeur classique d'un POMDP.

#### 4.2.4 Fonction de valeur

La fonction de valeur  $V^\pi(b)$  est définie comme le coût total espéré (pondéré par le temps en utilisant  $\gamma$ ) que l'agent va recevoir depuis  $b_0$  en suivant la politique  $\pi$  [KAELBLING et al. 1998; CASSANDRA et al. 1997]. Elle est définie de la façon suivante :

$$V^\pi(b) = \mathbb{E}^\pi \left[ \sum_{t=0}^{\infty} \gamma^t \mathcal{C}(b_t, b_{t+1}, \pi(b_t)) | b_0 = b \right] \quad (4.22)$$

Il est à noter que la fonction de coût que nous avons défini précédemment (Éq. 4.21) dépend de la transition entre deux états de croyances. L'incertitude sur l'état est liée à la matrice de covariance sur l'erreur d'exécution  $\tilde{\Sigma}$ , qui est affectée par le mode de navigation.

La politique optimale  $\pi^*$  est définie par la fonction de valeur optimale  $V^{\pi^*}$  :

$$V^{\pi^*}(b) = \min_{\pi \in \Pi} \mathbb{E} \left[ \sum_{t=0}^{\infty} \gamma^t [\mathcal{C}(b_t, b_{t+1}, \pi(b_t))] | b_0 = b \right] \quad (4.23)$$

Si l'on ouvre la somme dans l'Équation 4.23, on retrouve l'opérateur de Bellman, ce qui permet d'appliquer la programmation dynamique pour calculer la valeur optimale :

$$\begin{aligned} V^{\pi^*}(b) &= \min_{\pi \in \Pi} \mathbb{E} \left[ \sum_{t=0}^{\infty} \gamma^t [\mathcal{C}(b_t, b_{t+1}, \pi(b_t))] | b_0 = b \right] \\ &= \min_{\pi \in \Pi} \mathbb{E} \left[ \mathcal{C}(b_0, b_1, \pi(b_0)) + \sum_{t=1}^{\infty} \gamma^t \mathcal{C}(b_t, b_{t+1}, \pi(b_t)) | b_0 = b \right] \\ &= \min_{\pi \in \Pi} \sum_{s_v \in S_v} p(s_v | b, \pi(b)) \left( \mathcal{C}(b, b_{\pi(b)}^{s_v}, \pi(b)) + \gamma \mathbb{E} \left[ \sum_{t=0}^{\infty} \gamma^t \mathcal{C}(b_t, b_{t+1}, \pi(b_t)) | b_0 = b_{\pi(b)}^{s_v} \right] \right) \\ &= \min_{a \in A} \sum_{s_v \in S_v} p(s_v | b, a) \left( \mathcal{C}(b, b_a^{s_v}, a) + \gamma \mathbb{E} \left[ \sum_{t=0}^{\infty} \gamma^t \mathcal{C}(b_t, b_{t+1}, \pi(b_t)) | b_0 = b_a^{s_v} \right] \right) \\ &= \min_{a \in A} \sum_{s_v \in S_v} p(s_v | b, a) (\mathcal{C}(b, b_a^{s_v}, a) + \gamma V(b_a^{s_v})) \\ &= \min_{a \in A} \left[ \sum_{s_v \in S_v} p(s_v | b, a) (\mathcal{C}(b, b_a^{s_v}, a) + \gamma V(b_a^{s_v})) \right] \end{aligned} \quad (4.24)$$

Dans le cas d'une résolution approchée, quand la valeur (Éq. 4.24) converge pour tous les états de croyances dits atteignables (avec une erreur inférieure à  $\epsilon$ ), il est possible d'extraire la politique (partielle) optimisée associée [KURNAWATI, HSU et al. 2008].

### 4.3 Algorithmes de résolution

Étant donnée notre fonction de coût (Éq. 4.21) et la fonction de valeur qui en découle (Éq. 4.24), nous avons dû utiliser des algorithmes de résolution qui n'utilisent pas la propriété classique des fonctions de valeur (PWLC). Nous avons donc proposé deux algorithmes, LRTDP-bel adapté de *Labeled Real Time Dynamic Programming* (LRTDP) et MCBTS qui est une adaptation de POMCP.

#### 4.3.1 Initialisation de la fonction de valeur

La fonction de valeur d'un (PO)MDP provient de l'application successive de l'opérateur de Bellman par la programmation dynamique.

LRTDP est un algorithme pour lequel la valeur initiale d'un état de croyance doit être une borne inférieure dans le cas d'un problème de minimisation afin de préserver la convergence de l'algorithme et pour respecter la propriété de contraction de cet opérateur.

#### Définition 4.1

**Fonction de valeur avec une borne inférieure monotone.** Une fonction de valeur  $V$  d'un MDP a une borne inférieure monotone sur  $V^*$  si et seulement si pour tout état  $s \in \mathcal{S}$ ,  $V(s) \leq \min_{a \in \mathcal{A}} Q^V(s, a)$ . La  $Q$ -valeur  $Q^V(s, a)$  étant la valeur à une époque de l'action  $a$  dans l'état  $s$ . [KOLOBOV 2012]

Intuitivement, une fonction de valeur avec une borne inférieure monotone ne peut croître que par le résultat de la mise à jour de Bellman. Cette monotonie d'une heuristique admissible a une conséquence importante pour la convergence de l'algorithme.

Cependant, dans le cas de l'algorithme MCBTS, il est possible d'utiliser n'importe quelle valeur pour initialiser un état de croyance dû au fonctionnement de l'algorithme.

Nous pouvons donc utiliser la même fonction d'initialisation des valeurs pour les deux algorithmes. Ainsi l'initialisation de l'état de croyance dans nos algorithmes se fait par l'exploration du chemin  $A^*$  le plus court sur la carte de l'environnement (discretisée sous forme d'une grille). L'erreur d'exécution  $\tilde{\Sigma}$  est propagée le long de ce chemin pour le mode de navigation dont la disponibilité est la plus probable.

### 4.3.2 LRTDP-bel

Le premier algorithme de résolution, que nous proposons LRTDP-bel est basé sur les algorithmes *Real-Time Dynamic Programming* (RTDP) [BARTO et al. 1995], LRTDP [BONET et Hector GEFNER 2003] et RTDP-bel [BONET et Héctor GEFNER 2009], qui n'ont pas pour prérequis que la fonction de valeur du problème soit une représentation de la fonction de valeur basée sur des  $\alpha$ -vecteurs. Nous proposons dans cet algorithme d'évaluer directement l'état de croyance (et non pas les états comme dans RTDP-bel) tout en explorant l'amélioration de la convergence introduite par l'algorithme LRTDP.

LRTDP introduit une mécanique pour identifier les états  $\epsilon$ -consistants et les marquer comme résolus pour ne plus les explorer afin de définir comme critère d'arrêt de l'algorithme la détection de la convergence de l'état de croyance initiale.

Notons  $R^V(s)$  la valeur résiduelle, c'est-à-dire la différence entre la valeur de l'état  $s$  et le résultat de la  $Q$ -valeur de  $s$  pour l'action gloutonne  $a$ .

#### Définition 4.2

Pour un algorithme RTDP initialisé avec une borne inférieure monotone, la fonction de valeur actuelle  $V$  est  $\epsilon$ -consistante dans un état  $s$  et restera  $\epsilon$ -consistante dans cet état si  $R^V(s) < \epsilon$  et  $R^V(s') < \epsilon$  pour tous les états successeurs  $s'$  de  $s$  dans le graphe glouton  $G_s^V$ . [KOLOBOV

2012]

LRTDP travaille sur des MDP, c'est pourquoi il est nécessaire d'adapter certaines fonctions et définitions. Tout d'abord, nous définissons le résiduel comme étant :

$$R(b) = \left| V(b) - \min_{a \in A} \sum_{s_v \in S_v} p(s_v|b, a) (\mathcal{C}(b, b_a^{s_v}, a)) + \gamma V(b_a^{s_v}) \right| \quad (4.25)$$

Puis, comme pour LRTDP, nous considérons que l'état de croyance a convergé si la définition suivante est vérifiée.

**Théorème 4.1**

*Pour un algorithme LRTDP initialisé avec une borne inférieure monotone, la fonction de valeur actuelle  $V$  est  $\epsilon$ -consistant dans un état de croyance  $b$  et restera  $\epsilon$ -consistant dans cet état de croyance si  $R^V(b) < \epsilon$  et  $R^V(b')$   $< \epsilon$  pour tous les états descendants  $b'$  de  $b$  dans le graphe glouton  $G_b^V$ . [KOLOBOV 2012]*

La valeur de  $b$  est garantie de rester  $\epsilon$ -consistant indéfiniment à partir du moment où l'application de l'opérateur de Bellman ne peut pas changer la valeur de  $V(b)$  ou de ses descendant. En effet, la valeur de  $V(b)$  est déterminé uniquement par les valeurs de ses descendants dans  $G_b^V$ . Par conséquent, dans le cas où le graphe  $G_b^V$  ne change pas,  $V(b)$  peut uniquement croître d'une valeur supérieur à  $\epsilon$  dans le cas où certains de ses descendants  $b'$  croissent d'une valeur supérieur à  $\epsilon$ . Or cela contredirait les prémisses du théorème, qui dit que les descendant d'un état de croyance  $\epsilon$ -consistant sont eux-même  $\epsilon$ -consistant.

Notre adaptation de LRTDP montré dans l'algorithme 7 prend en entrée un état de croyance initial  $b_0$  et un paramètre  $\epsilon$ . Tout le temps que  $b_0$  n'a pas convergé, l'algorithme va continuer à simuler une politique gloutonne (trial). Le fonctionnement d'un trial dans notre algorithme est similaire à LRTDP à quelques exceptions près. En effet, en plus de travailler sur des états de croyances au lieu des états, la mise à jour de l'état de croyance se fait en utilisant l'algorithme Expectation-Maximization (Alg. 6) pour apprendre un modèle de mélange gaussien qui approxime l'état de croyance suivant. Quand l'objectif est atteint, l'algorithme vérifie si toutes les valeurs de l'état de croyances ont convergé. Cette vérification est faite à l'aide de l'algorithme Check-Solved montré dans l'algorithme 8.

**Algorithme 7 : LRTDP-Bel**


---

```

1 Fonction LRTDP-BEL( $b_0, \epsilon$ )
2   while  $b_0$  not solved do
3     | LRTDP-TRIAL( $b_0, \epsilon$ )
4   | return  $\pi_{b_0}^*$ 
5 Fonction LRTDP-TRIAL( $b_0, \epsilon$ )
6   visited  $\leftarrow \emptyset$ ;  $b \leftarrow b_0$ 
7   while  $b$  not solved do
8     | visited  $\leftarrow b$ 
9     | if  $b \notin \text{Goal}$  then
10    |    $a_{\text{best}} \leftarrow \arg \min_{a \in A} Q^{V_i}(b, a)$ 
11    |    $V_i(b) \leftarrow Q^{V_i}(b, a_{\text{best}})$ 
12    |    $b_a \leftarrow \text{execute } a_{\text{best}} \text{ in } b$ 
13    |    $s_v \leftarrow \text{sample } s_v \text{ from } p(s_v | b_a)$ 
14    |    $b_a^{s_v} \leftarrow \text{EM}(b_a, s_v)$ 
15    |    $b \leftarrow b_a^{s_v}$ 
16   | while visited  $\neq \emptyset$  do
17   |    $b \leftarrow \text{pop}(\text{visited})$ 
18   |   if !CHECK-SOLVED( $b, \epsilon$ ) then
19   |     | break

```

---

**Algorithme 8 : Check-Solved**


---

```

1 Fonction CHECK-SOLVED( $b'_0, \epsilon$ )
2   retValue  $\leftarrow \text{true}$ 
3   open  $\leftarrow \emptyset$ 
4   closed  $\leftarrow \emptyset$ 
5   open  $\leftarrow b'_0$ 
6   while open  $\neq \emptyset$  do
7     |  $b \leftarrow \text{open}$ 
8     | closed  $\leftarrow b$ 
9     | if  $R^{V_i}(b) > \epsilon$  then
10    |   | retVal  $\leftarrow \text{false}$ 
11    |   | continue
12    |    $a_{\text{best}} \leftarrow \arg \min_{a \in A} Q^{V_i}(b, a)$ 
13    |   foreach  $s'_v \in S_v$  and  $p(s'_v | b, a_{\text{best}}) > 0$  do
14    |     | if  $b'$  is not solved and  $b' \notin \text{open} \cup \text{closed}$  then
15    |       |   |  $\text{open} \leftarrow b'$ 
16   | if retVal == true then
17   |   | foreach  $b \in \text{closed}$  do
18   |     |   | label  $b$  solved
19   | else
20   |   | while closed  $\neq \emptyset$  do
21   |     |   |  $b \leftarrow \text{closed}$ 
22   |     |   |  $V_i(b) \leftarrow \min_{a \in A} Q^{V_i}(b, a)$ 
23   | return retVal

```

---

### 4.3.3 MCBTS

L'algorithme POMCP [SILVER et al. 2010] est un algorithme fonctionnant sur le principe de l'algorithme Monte-Carlo Tree Search (MCTS) adapté pour les environnements partiellement observables. POMCP fonctionne en tirant aléatoirement des états  $s$  dans l'état de croyance courant, aussi appelé nœud ou historique, et simule des séquences de paires action-observation (procédure appelée *trial*) pour construire un arbre et calculer la récompense moyenne pour un nœud basé sur la récompense moyenne des nœuds fils. L'algorithme garde en mémoire le nombre de fois qu'un nœud a été exploré  $N(h)$  et le nombre de fois qu'une action a été choisie  $N(ha)$  dans ce nœud. Comme pour l'algorithme UCT [KOC SIS et al. 2006], il applique la stratégie de sélection d'action UCB1 qui est basée sur la combinaison de deux caractéristiques : une approximation de la  $Q$ -valeur et une mesure (donnée par  $c\sqrt{\frac{\log N(h)}{N(ha)}}$ ) qui renseigne à quel point une action a bien été explorée. Cette mesure permet de gérer le ratio entre exploration et exploitation. Pour plus de détail sur l'algorithme, veuillez vous référer à la section 1.5.3.

Cependant, dû aux particularités du problème traité dans cette thèse et du fait que le problème est orienté but, nous proposons un nouvel algorithme MCBTS - *Monte-Carlo Belief Tree Search* (Alg. 9) basé sur MCTS et POMCP. MCBTS est un algorithme de type hors ligne. L'algorithme prend en entrée l'état de croyance initiale (Ligne 3) et lance un *trial* jusqu'à ce qu'un état de croyance final soit atteint : le but est atteint (Ligne 7) ou une collision est détectée (ligne 10). Si aucun de ces deux cas est détecté, alors l'algorithme continue à explorer. Si l'état de croyance n'a jamais été exploré alors on initialise chaque état de croyance fils et ils sont ajoutés à l'arbre (Lignes 13 - 16). De plus contrairement à l'algorithme classique, aucun état n'est tiré aléatoirement puisque nous exploitons directement les états de croyances sous forme de fonctions gaussiennes. La prochaine action est choisie en utilisant la stratégie UCB1 (Ligne 20). Après avoir exécuté l'action, on tire aléatoirement une observation et l'état de croyance est mis à jour en utilisant l'algorithme EM (Alg. 6). Ensuite un nouveau *trial* est lancé récursivement avec le nouvel état de croyance.

---

**Algorithme 9 : MCBTS**


---

```

1 Fonction MCBTS( $b_0, c, \gamma$ )
2   while !Timeout do
3     | Trial( $b_0, c, \gamma$ )
4   end
5    $a^* \leftarrow \arg \min_{a \in A} Q(b_0, a)$ 
6 Fonction Trial( $b, c, \gamma$ )
7   if  $b \in \textit{Goal}$  then
8     | return 0
9   end
10  if  $F_{Col} == 1$  then
11    | return  $K$ 
12  end
13  if  $b \notin T$  then
14    | foreach  $a \in A$  do
15      | foreach  $s_v \in S_v$  do
16        |  $T(b_a^{s_v}) \leftarrow (N_{\textit{init}}(b_a^{s_v}), V_{\textit{init}}(b_a^{s_v}), \emptyset)$ 
17      | end
18    | end
19  end
20   $\bar{a} \leftarrow \arg \min_{a \in A} \left( Q(b, a) - c \sqrt{\frac{\log N(b)}{N(b_{\bar{a}})}} \right)$ 
21   $s_v \sim b_{\bar{a}}$ 
22   $b_a^{s_v} \leftarrow \textit{update}(b_{\bar{a}}, s_v)$ 
23   $N(b) \leftarrow N(b) + 1$ 
24   $N(b_{\bar{a}}) \leftarrow N(b_{\bar{a}}) + 1$ 
25   $Q(b, \bar{a})' \leftarrow C(b, b_a^{s_v}, \bar{a}) + \gamma \textit{Trial}(b_a^{s_v}, c, \gamma)$ 
26   $Q(b, \bar{a}) \leftarrow Q(b, \bar{a}) + \frac{Q(b, \bar{a})' - Q(b, \bar{a})}{N(b_{\bar{a}})}$ 
27   $V(b) \leftarrow \min_{a \in A} Q(b, a)$ 

```

/\* Générateur aléatoire \*/  
/\* Calcule du GMM \*/

---



## 4.4 Expérimentations

### 4.4.1 Setup des expérimentations

**Capteurs :** Nous avons utilisé le modèle du véhicule défini dans le chapitre 3, en ne considérant que deux capteurs embarqués : l'IMU et le GPS.

**Environnement :** Les deux algorithmes LRTDP et MCBTS ont été testés avec un benchmark de test pour la navigation de drones<sup>1</sup> [METTLER et al. 2010] avec les cartes d'obstacles simples et la carte compliqué de San Diego (Figure 4.9). Pour chacune de ces cartes, nous avons utilisé les cartes de probabilités de disponibilité du GPS calculée par le simulateur GPS OKTAL disponible à l'ONERA/DEMR (Figure 4.9). Les tests ont été effectués sur deux cartes différentes, la carte avec un seul cube *Cube* et la carte avec deux cubes *CubeBaffle*.

**Condition initiale :** L'état de croyance initiale était  $b_0 = (s_{v_0}, b_{s_{h_0}} = (\bar{s}_{h_0}, \tilde{\Sigma}_0))$ , avec :

- $s_{h_0} = [5, 5, 7, 0, 0, 0, 0, 0, 0]$
- $\tilde{\Sigma}_0 = \text{diag}(1, 1, 4, 0.01, 0.01, 0.04, 0.01, 0.01, 0.01)$
- $s_{v_0} = [1, 1, 0, P]$ ,  $P = \tilde{\Sigma}_0$ .

Nous définissons le but en  $(90, 90, 7)$  et le coût de collision  $K = 1000$ . Le but est considéré atteint si une fonction gaussienne de moyenne  $(90, 90, 7)$  et de covariance  $\tilde{\Sigma}_0$  a plus de 90% de chance de se trouver dans l'état de croyance actuel.

**Actions :** Pour rappel, une action est composée d'une vitesse de référence et d'un mode de navigation (choix du capteur). L'ensemble des  $\mathcal{V}_{\text{ref}}$  est composé de 10 vitesses de référence qui peuvent être représentées sous forme de directions (Figure 4.8).

**Algorithmes :** Pour obtenir des résultats représentatifs, nous avons effectué 1000 simulations pour chaque politique calculée par l'algorithme LRTDP modifié et MCBTS. Le paramètre pour l'algorithme LRTDP modifié est  $\gamma = 0.9$ . Les paramètres pour l'algorithme MCBTS sont  $\gamma = 0.9$ ,  $\text{Timeout} = 180\text{min}$  et  $c = 200$ .

**Métriques :** Nous considérons les métriques suivantes pour évaluer la qualité des politiques :

- Le taux de succès : Rapport du nombre de simulations qui ont atteint l'objectif sur le nombre total de simulations.
- Coût moyen : Moyenne des coûts totaux des simulations qui ont atteint l'objectif. C'est-à-dire la moyenne des volumes entre les états de croyances des simulations qui ont atteint l'objectif et des pénalités de collisions.

---

1. Le benchmark de test est disponible à l'adresse : [www.aem.umn.edu/people/mettler/projects/AFDD/AFDDwebpage.htm](http://www.aem.umn.edu/people/mettler/projects/AFDD/AFDDwebpage.htm)

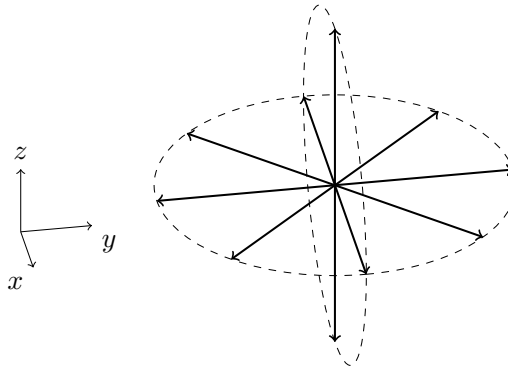


FIGURE 4.8 – Ensemble des vitesses de références.

À noter que sortir de la carte est considéré comme une collision.

#### 4.4.2 Résultats

La moyenne des résultats obtenus est condensée dans le tableau 4.1. En termes de performance, le taux de succès est presque similaire pour chaque algorithme, quelle que soit la carte. Cependant, le coût moyen de chaque chemin est différent entre les deux algorithmes. Nous constatons que MCBTS est plus performant que LRTDP pour minimiser le coût du chemin. Pour la carte à deux obstacles la différence est plus petite que pour la carte avec un seul obstacle. MCBTS réduit le coût respectivement de 9.45% et de 30.54%.

La figure 4.10 présente les résultats des trajectoires simulées sur la carte « CubeBaffle » pour la politique calculée avec LRTDP et MCBTS. Les figures 4.10a et 4.10c représentent les chemins simulés en suivant les politiques calculées et les figures 4.10b et 4.10d représentent les états de croyances approximés par GMM d'un des chemins simulés. Comme pour la carte précédente, la figure 4.11 présente les résultats des politiques calculées par les deux algorithmes sur la carte « Cube ».

Il est à noter que les chemins simulés des politiques obtenues pour les deux algorithmes sont presque similaires. Ceci est dû au fait que les cartes soient simples et que les chemins les plus surs et plus courts sont relativement faciles à calculer. Cependant, nous pouvons observer que les trajectoires simulées par la politique calculée avec MCBTS sont plus lisses. Pour la carte avec un seul obstacle, MCBTS a mieux anticipé la disponibilité des capteurs ainsi que le risque de collision que LRTDP. Ce résultat pouvait être espéré, car LRTDP favorise plus d'optimisations locales lors du calcul de la politique par un choix glouton d'action. Alors que MCBTS avec le choix d'action d'UCB1 explore plus en amont de la recherche arborescente. La représentation en deux dimensions donne une bonne idée des chemins simulés, mais il est aussi intéressant de les analyser en trois dimensions (Figure 4.12). La figure 4.12a représente les chemins obtenus avec la politique calculée par LRTDP et la figure 4.12b ceux obtenus avec la politique calculée par MCBTS. Cependant, MCBTS calcule une politique qui monte

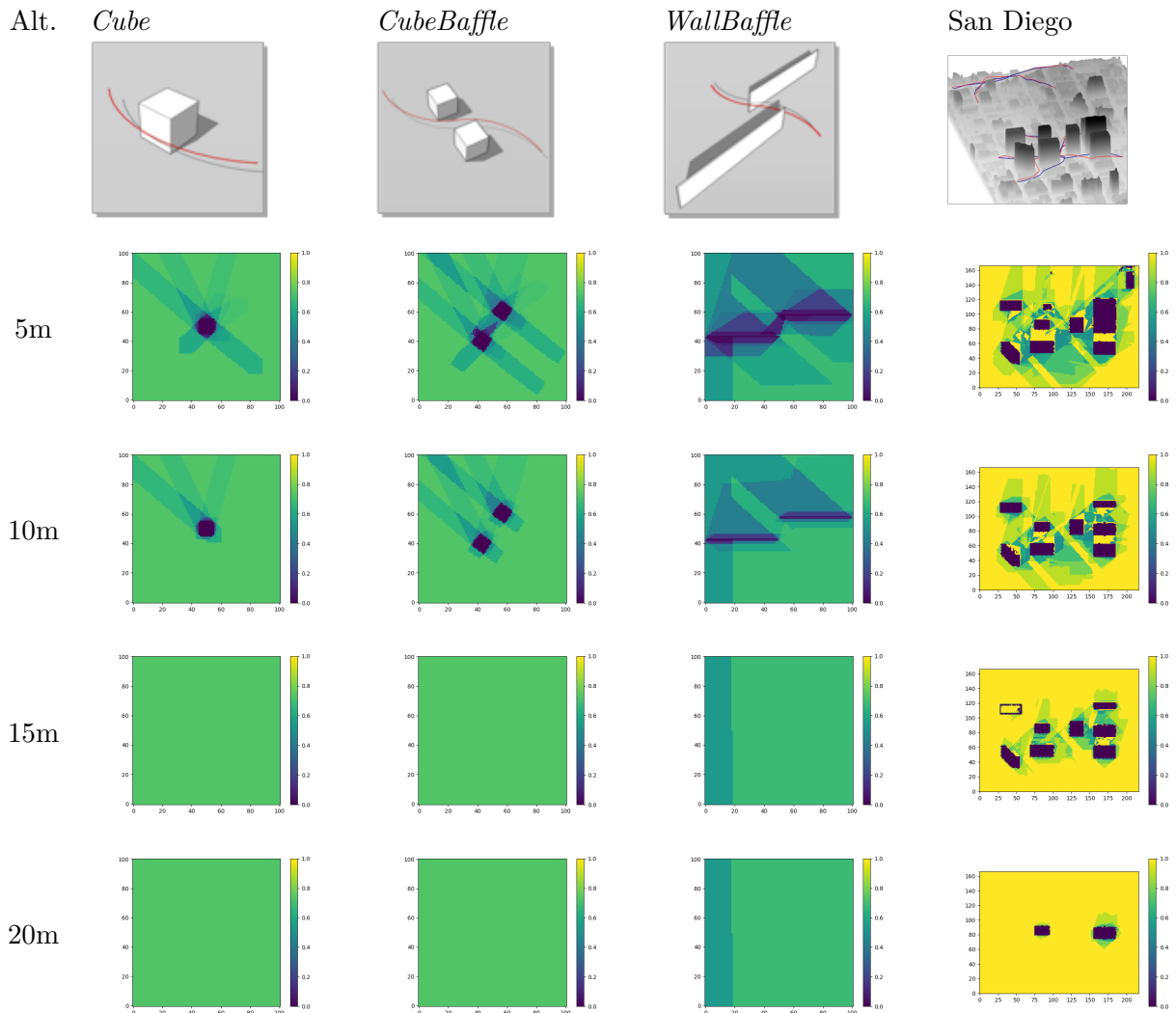
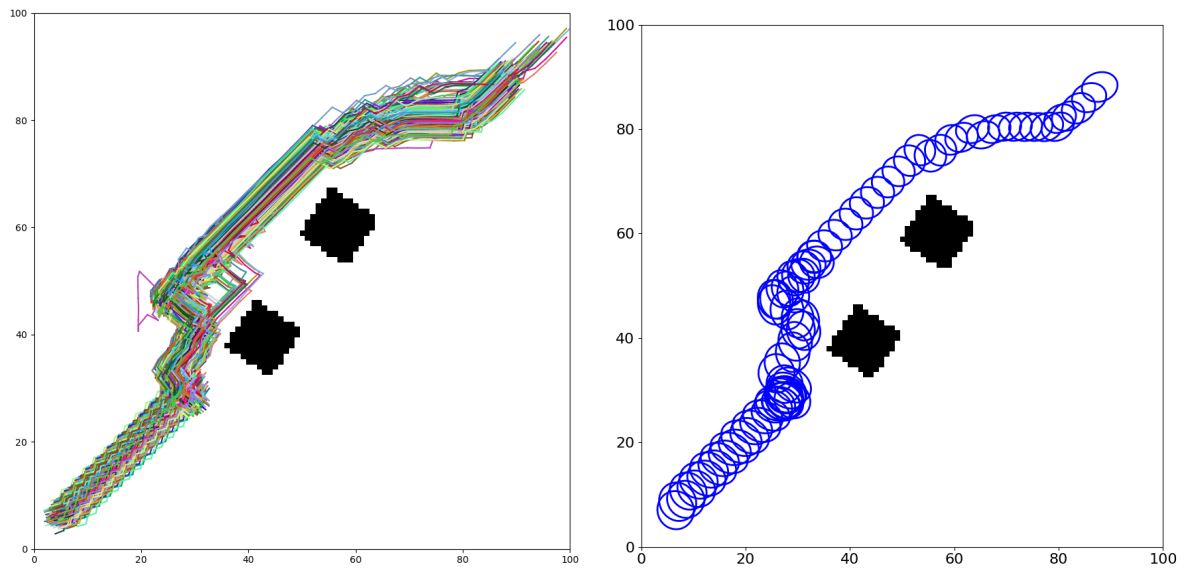


FIGURE 4.9 – Exemple de cartes proposées sur le benchmark de test ainsi que des exemples de cartes de probabilités de disponibilité du GPS.

en hauteur au niveau du premier obstacle. Cela peut s'expliquer par la plus grande probabilité que le GPS soit disponible (Figure 4.9) et donc le choix limiter l'incertitude et donc le risque de collision. Les chemins restent concentrés dans les états de croyances qui sont un peu plus larges en hauteur due à une plus faible précision du GPS sur cet axe. La carte étant limitée en hauteur, les algorithmes n'ont pas pu calculer des politiques permettant de passer par dessus les obstacles en évitant des collisions.

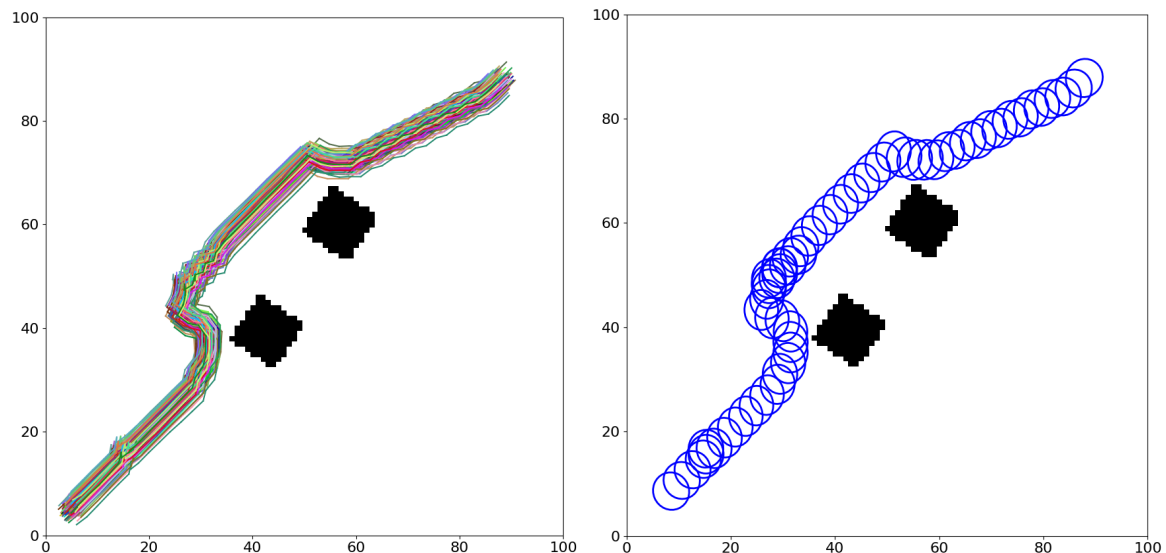
	CubeBaffle		Cube	
	LRTDP	MCBTS	LRTDP	MCBTS
Taux de succès	96%	95%	96%	96%
Coûts moyens	3393.84	3072.67	5892.91	4093.24
Coûts moyens en %	0%	-9.47%	0%	-30.54%

TABLE 4.1 – Comparaison des performances entre les deux algorithmes proposés basés sur LRTDP et MCBTS.



(a) Chemins simulés avec la politique calculé par LRTDP sur la carte « Cube baffle ».

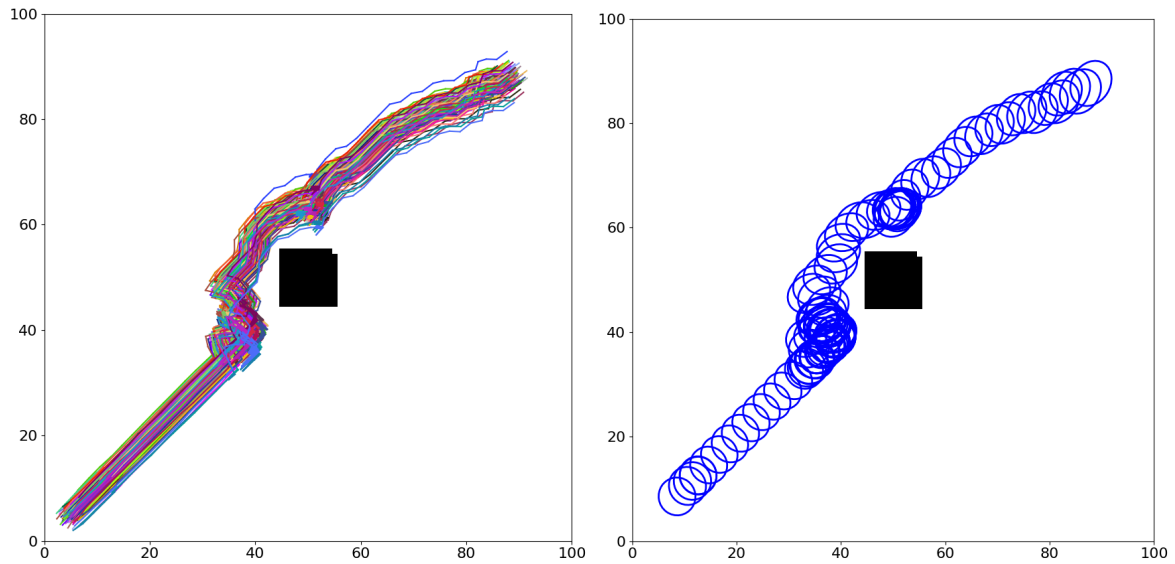
(b) Représentation des états de croyances des chemins simulés par LRTDP.



(c) Chemins simulés avec la politique calculé par MCBTS sur la carte « Cube baffle ».

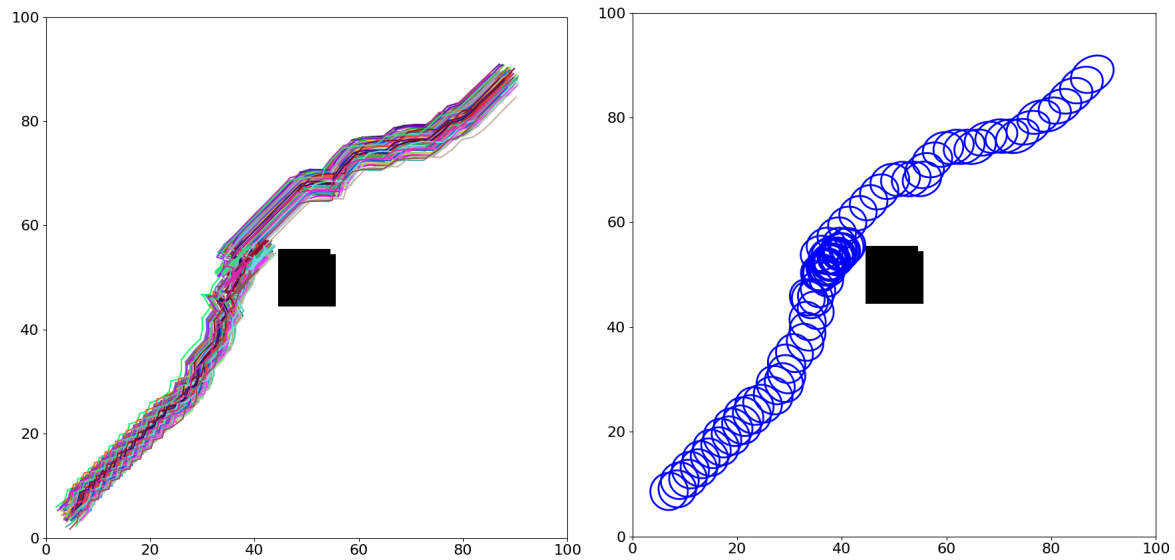
(d) Représentation des états de croyances des chemins simulés par MCBTS.

FIGURE 4.10 – Chemins simulés sur la carte avec deux obstacles « CubeBaffle ».



(a) Chemin simulé avec la politique calculé par LRTDP sur la carte « Cube ».

(b) Représentation des états de croyances des chemins simulés par LRTDP.



(c) Chemins simulés avec la politique calculé par MCBTS sur la carte on « Cube ».

(d) Représentation des états de croyances des chemins simulés par MCBTS.

FIGURE 4.11 – Chemins simulés sur la carte avec un obstacle « Cube ».

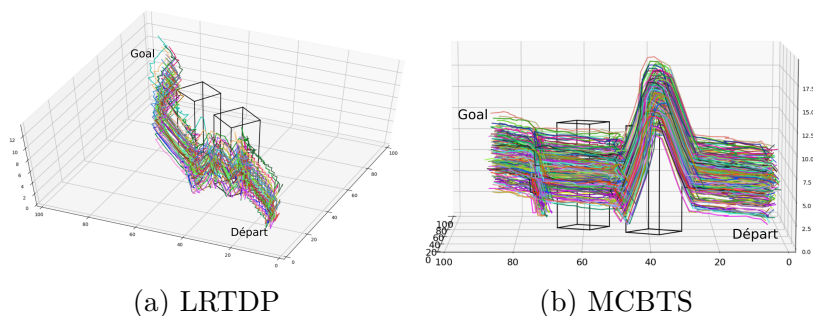


FIGURE 4.12 – Représentation en 3D des chemins simulés sur la carte « CubeBaffle ».

## 4.5 Bilan

Dans ce chapitre, nous avons proposé un modèle basé sur les MOMDP qui intègre un modèle de transition GNC comme fonction de transition d'état. L'intégration du modèle GNC permet de propager l'incertitude de l'exécution de la trajectoire qui dépend du choix / disponibilité du capteur. Cependant, les connaissances a priori étant représentées sous forme de carte de probabilité de disponibilité des capteurs compliquent la mise à jour de l'état de croyance. La probabilité de chaque observation étant valide sur des cases de la grille, cela déforme la loi normale renvoyée par la fonction de transition basée sur le GNC qui casse la propriété gaussienne de l'incertitude. C'est pourquoi nous avons proposé d'utiliser l'algorithme *Expectation-Maximization* afin d'approximer les états de croyances par des modèles de mélanges gaussiens afin de pallier à ce problème. Cette représentation des états de croyances permet d'utiliser la notion de couloir d'incertitude afin de calculer le coût de transition entre deux états de croyances en tenant compte de l'incertitude. La fonction de coût permet de pénaliser l'incertitude sur l'état du véhicule et le temps de vol sans favoriser un de ces critères plus que l'autre, car elle calcule le volume entre les états de croyances qui dépend de l'erreur d'exécution cumulée et de la distance parcourue. Toutefois, contrairement aux fonctions de coût classiquement utilisées dans la littérature qui permettent d'obtenir une fonction de valeur linéaire par morceaux et convexe, la fonction de coût que nous proposons ne permet plus d'obtenir cette propriété sur la fonction de valeur. Cela empêche l'utilisation des algorithmes de résolutions de littérature qui utilisent cette propriété.

C'est pourquoi dans ce chapitre nous avons présenté deux algorithmes qui s'appuient sur des algorithmes de la littérature qui n'explorent pas cette propriété pour prendre en compte les particularités de notre modèle. LRTDP-Bel se base sur l'utilisation d'une recherche heuristique gloutonne et a une propriété permettant de vérifier la convergence de certains états de croyances. Nos algorithmes prennent en compte notre apprentissage de modèle de mélange gaussien.

Par la suite, nous avons présenté les résultats que nous avons obtenus sur deux cartes simples faisant partie d'un benchmark de test pour la navigation de drone en milieux encombrés. Ces expérimentations ont été faites en considérant deux capteurs embarqués l'INS et le GPS. L'algorithme proposé MCBTS permet d'obtenir des chemins plus lisses qui anticipent

mieux les obstacles. Ces contributions ont donné lieu à des publications [DELAMER et al. 2017a ; DELAMER et al. 2017b].

Cependant, le modèle du MOMDP et les algorithmes proposés permettent d'obtenir des stratégies de navigation pour des drones autonomes sur des cartes simples. Sur des cartes plus complexes et réalistes comme la carte de San Diego disponible sur le benchmark de test (Figure 4.9), les algorithmes ne parviennent pas à calculer en temps raisonnable une politique acceptable. Une des raisons possibles est que dans des environnements complexes, le temps de calcul des modèles de mélange gaussien devient plus important. En effet sur les cartes simples le GPS est disponible sur une très grande partie de la carte avec une probabilité élevée. Or sur une carte complexe, la disponibilité du GPS est plus incertaine ce qui augmente le nombre d'observations différentes pouvant être constatées après une transition et donc le nombre d'états de croyances atteignables. Or la probabilité d'une observation est compliquée à calculer (Éq. 4.13), car il est nécessaire de calculer la densité de probabilité de l'observation pour chaque cellule composant le GMM. De plus, même si la représentation par modèle de mélange gaussien permet de simplifier la mise à jour de l'état de croyance, le nombre de fonctions gaussiennes  $N_g$  la représentant peut croître rapidement en fonction des observations perçues dans un environnement complexe. C'est probablement la principale raison de l'impossibilité de calculer une politique sur la carte de San Diego.

Dans le prochain chapitre, nous présentons un nouveau modèle toujours basé sur le MOMDP pour pallier le problème rencontré avec celui-ci avec des environnements plus complexes. Ce deuxième modèle ne calcule plus directement la transition des états de croyance. Les états de croyances ne sont plus représentés par des modèles de mélange gaussien, mais par des ensembles de particules. L'idée est alors d'utiliser un algorithme plus proche du POMCP originel, cela permettrait d'éviter d'appliquer l'apprentissage de GMM et d'estimer la valeur moyenne des particules évoluant selon le modèle de transition basé sur le GNC. Nous rappelons que la valeur moyenne de ces particules approche (par la loi des grands nombres) la valeur de l'état de croyance qu'elles représentent. De plus, la probabilité de chaque observation pour un état caché  $s_h$  est directement donnée par les cartes de probabilités.





# 5

## Évolution du modèle MOMDP et algorithme de résolution

### Sommaire

---

<b>5.1</b>	<b>Présentation du nouveau modèle MOMDP</b>	<b>80</b>
5.1.1	Fonction de coût	80
5.1.2	Fonction de valeur	81
<b>5.2</b>	<b>Algorithme de résolution</b>	<b>81</b>
5.2.1	Fonction heuristique et initialisation de la valeur	84
5.2.2	Élagage de l'arbre de recherche	84
<b>5.3</b>	<b>Expérimentations</b>	<b>87</b>
5.3.1	Setup des expérimentations	87
5.3.2	Résultats	88
<b>5.4</b>	<b>Bilan</b>	<b>96</b>

---

Ce chapitre a pour objectif de présenter le deuxième modèle de planification toujours basé sur un MOMDP pour résoudre le problème de planification d'une stratégie de navigation. Ce nouveau modèle est une évolution du modèle proposé dans le chapitre précédent pour pallier les limitations discutées. Ce modèle-ci ne calculera plus le nouvel état de croyance sous forme d'un modèle de mélange gaussien. La fonction de coût du précédent modèle utilisait cette approximation de l'état de croyance pour calculer directement le coût qui représente l'erreur d'exécution et la distance entre les états de croyance. Nous proposons donc une nouvelle fonction de coût plus simple qui est calculée à partir du temps de vol et d'une pénalité de collision. Nous proposons aussi un algorithme de résolution appelé POMCP-GO basé sur POMCP. Nous comparons les performances de l'algorithme proposé par rapport à POMCP afin de montrer son efficacité dans la résolution du problème de planification présenté dans cette thèse.

## 5.1 Présentation du nouveau modèle MOMDP

Le modèle MOMDP présenté dans ce chapitre n'a pour différence importante que la fonction de coût. C'est pourquoi nous détaillerons uniquement la nouvelle fonction de coût.

Nous définissons notre modèle MOMDP comme un tuple  $(\mathcal{S}_v, \mathcal{S}_h, \mathcal{A}, \mathcal{O}, \mathcal{T}, \mathcal{R}, b_0, \gamma)$  où :

- $\mathcal{S}_v$  est l'espace borné d'états totalement observable.
- $\mathcal{S}_h$  est l'espace borné d'états continus non observable.
- $\mathcal{A}$  est l'ensemble fini des actions
- $\Omega$  est l'ensemble fini des observations
- $\mathcal{T}(s_v, s_h, a, s'_v, s'_h) \rightarrow [0, 1]$  est la fonction de transition composée de deux fonctions :
  - $T_{\mathcal{S}_h} : \mathcal{S}_h \times \mathcal{S}_v \times \mathcal{A} \times \mathcal{S}_h \rightarrow [0; 1]$  une fonction de transition :  $T_{\mathcal{S}_h}(s_h, s_v, a, s'_h) = p(s'_h | s_h, s_v, a)$ .
  - $T_{\mathcal{S}_v} : \mathcal{S}_v \times \mathcal{S}_h \rightarrow [0; 1]$  une fonction de transition :  $T_{\mathcal{S}_v}(s'_v, s'_h) = p(s'_v | s'_h)$ ;
- $\mathcal{O} : \Omega \times \mathcal{S}_v \rightarrow [0, 1]$  est la fonction d'observation :

$$O(o, s'_v) = p(o | s'_v) = \begin{cases} 1 & o = s'_v \\ 0 & \text{sinon} \end{cases} \quad (5.1)$$

- $\mathcal{C} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$  : la fonction de coût où  $\mathcal{S} = \mathcal{S}_v \times \mathcal{S}_c$

### 5.1.1 Fonction de coût

La fonction de coût est le principal changement de ce modèle. En effet, puisque nous ne considérons plus que les états de croyances sont des modèles de mélange gaussien, il n'est plus possible de calculer un volume du couloir d'incertitude entre deux GMM. L'objectif étant toujours de minimiser le risque de collision (pour des raisons de sûreté) et le temps de vol (pour des raisons d'efficacité), la fonction de coût est définie de la façon suivante :

$$C(s_t) = \begin{cases} K - t \times f = K_t & \text{si } s_t \text{ en collision} \\ 0 & \text{si } s_t \text{ est un état but} \\ f & \text{sinon} \end{cases} \quad (5.2)$$

où  $f$  est le temps de vol constant pour une action  $a$  à une époque  $t$  et  $K$  une pénalité fixe en cas de collision. Quand une collision est détectée, le coût est une pénalité fixe, de laquelle on soustrait le temps de vol total depuis l'état de croyance initiale jusqu'à la collision. Ceci est fait pour éviter de pénaliser davantage l'agent si la collision se produit plus tard. En d'autres

termes, si le coût d'un trajet entier est défini par la somme des coûts des actions, alors la fonction de coût pénalise de manière identique toutes les trajectoires qui finissent en collisions.

### 5.1.2 Fonction de valeur

La fonction de coût ayant changé dans ce modèle, il est important de redéfinir la fonction de valeur. La fonction de valeur  $V^\pi(b_0)$  est définie comme le coût total espéré que l'agent va payer depuis  $b_0$  en suivant la politique  $\pi$ . La fonction de valeur est définie comme :

$$V^\pi(b) = \mathbb{E} \left[ \sum_{t=0}^{\infty} \gamma^t [\mathcal{C}(b_t, \pi(b_t))] \mid b_0 = b \right] \quad (5.3)$$

où  $\mathcal{C}(b, \pi(b) = a) = \sum_{s \in \mathcal{S}} \mathcal{C}(s, a)b(s)$  est le coût total espéré d'une action  $a$  dans l'état de croyance  $b$  pour le cas des espaces d'états discrets.

La politique optimale  $\pi^*$  est définie par la fonction de valeur optimale  $V^{\pi^*}$  comme suit :

$$V^{\pi^*}(b) = \arg \min_{\pi \in \Pi} \mathbb{E} \left[ \sum_{t=0}^{\infty} \gamma^t [\mathcal{C}(b_t, \pi(b_t))] \mid b_0 = b \right] \quad (5.4)$$

En ouvrant la somme (Éq. 5.4, cf. Chap. 4), on retrouve l'équation de Bellman, qui permet l'application de la programmation dynamique, par exemple :

$$V(b) = \min_{a \in \mathcal{A}} \left[ \mathcal{C}(b, a) + \gamma \sum_{s_v \in \mathcal{S}_v} p(s_v \mid b, a) V(b_a^{s_v}) \right] \quad (5.5)$$

Quand la valeur (Éq. 5.5) converge pour tous les états de croyances, il est possible d'extraire la politique optimale. Cependant, le calcul de cette politique et de la fonction de valeurs est connu pour être un problème de décision difficile (et même indécidable [MADANI et al. 1999]), en particulier quand l'espace d'état est continu. De récents algorithmes permettent néanmoins d'approcher la solution en cherchant une politique partielle en ne considérant que les états de croyances atteignables [KURNIAWATI, HSU et al. 2008; BONET et Héctor GEFNER 2009; SILVER et al. 2010]. Ces algorithmes sont capables de calculer des solutions sous-optimales dans un temps raisonnable.

## 5.2 Algorithme de résolution

Garder et mettre à jour les états de croyance peut être une étape difficile et coûteuse sur le plan informatique en raison de la complexité potentielle de la fonction de mise à jour des états de croyance. En particulier parce que dans notre modèle l'espace d'état caché est continu et que l'espace d'état entièrement observable est discret, le calcul de la distribution

de probabilité sur  $\mathcal{S}_h$ , et la correction par  $s_v$  demanderait un effort informatique coûteux (cf. Chap. 4).

Une solution est d'utiliser comme pour le modèle précédent (Chap. 4) des algorithmes qui n'ont pas besoin de maintenir et de mettre à jour l'état de croyance à chaque étape de décision comme l'algorithme POMCP [SILVER et al. 2010].

D'autre part RTDP-bel est un algorithme qui se propose de résoudre principalement des POMDP orientés but. Pour cela, l'algorithme effectue des *trials* qui vont jusqu'au but en choisissant l'action selon un critère glouton, c'est-à-dire en sélectionnant l'action qui minimise le coût. Pour les états de croyances qui n'ont pas déjà été visités, une valeur basée sur une fonction heuristique leur est affectée.

Cependant, aucun de ces algorithmes ne permet de répondre à l'entièreté de nos besoins. C'est pourquoi nous avons proposé un algorithme POMCP-GO (*POMCP-Goal Oriented*) s'inspirant des algorithmes précédents pour résoudre notre problème (Alg. 10). Dans ce chapitre, pour mieux étudier les propriétés de l'algorithme proposé, nous utilisons une configuration hors-ligne. Donc, tous les *trials* commenceront toujours depuis l'état de croyance initial, c'est-à-dire la racine de l'arbre d'exploration (Ligne 5). Étant donné que le problème est un problème orienté but nous proposons qu'un *trial* continue jusqu'à ce qu'il rencontre un état terminal, c'est-à-dire si l'objectif est atteint ou si une collision est détectée. Si le but est atteint alors l'agent reçoit un coût de 0 (Ligne 9) et si une collision est détectée un coût de collision est retourné (Ligne 11) comme définit par l'équation 5.2. Sachant que mettre à jour les états de croyances est une opération coûteuse, nous nous sommes inspiré de l'algorithme POMCP afin de ne pas représenter les états de croyances directement, mais de les représenter sous forme de nœud historique pour lesquels la valeur sera mise à jour au fur et à mesure des *trials*. C'est pourquoi quand un nouveau nœud  $h$  est exploré, un nœud fils  $ha$  est créé et initialisé pour chaque action  $a$  (Ligne 16). Le nœud fils ayant la plus petite valeur est utilisé pour initialiser la valeur du nœud  $h$  (Ligne 17). Ainsi, contrairement à POMCP nous n'utilisons pas la procédure de *rollout* pour estimer la valeur initiale d'un nœud. À la place, nous utilisons une valeur heuristique pour initialiser les valeurs de chaque nouveau nœud  $ha$  comme pour RTDP-bel. À chaque étape l'action qui sera exécutée est choisie avec la stratégie de sélection UCB1 (Ligne 18), le nouvel état suivant est calculé en utilisant la fonction de transition GNC et l'état visible  $s'_v$  est tiré aléatoirement selon les probabilités données par les cartes de disponibilités des capteurs. L'algorithme continue ensuite d'explorer à partir du nouveau nœud  $h'$ . [KELLER et al. 2013] propose une approche similaire pour les MDP.

**Algorithme 10 : POMPC-GO**


---

```

1 Fonction POMCP-GO( $h, b_0, c, \gamma$ )
2    $h \leftarrow b_0$ 
3   while  $nbTrial < Nb_{max}$  do
4      $s_h \leftarrow \text{sampling } s_h \text{ from } b_0$ 
5     Trial( $h, s_h, s_v, c, 0$ )
6      $nbTrial += 1$ 
7    $a^* \leftarrow \arg \min_{a \in A} Q(b_0, a)$ 
8 Fonction Trial( $h, s_h, s_v, c, t$ )
9   if  $s_h \in G$  then
10    return 0
11  if  $F_{Col} == 1$  then
12    return  $K - t \times f_t$ 
13  if  $h \notin Tree$  then
14    for  $a \in A$  do
15       $ha \leftarrow h + a$ 
16       $Tree(ha) \leftarrow (N_{init}(ha), V_{init}(ha), \emptyset)$ 
17     $V(h) \leftarrow \arg \min_{a \in A} V(ha)$ 
18   $\bar{a} \leftarrow \arg \min_{a \in A} (Q(s_h, a) - c \sqrt{\frac{\log N(h)}{N(ha)}})$ 
19   $ha \leftarrow h + \bar{a}$ 
20   $s'_h, s'_v \sim \mathcal{G}(h, \bar{a})$ 
21   $hao \leftarrow ha + s'_v$ 
22   $N(h) \leftarrow N(h) + 1$ 
23   $N(ha) \leftarrow N(ha) + 1$ 
24   $Q(h, \bar{a})' \leftarrow C(s_h, a) + Trial(hao, s'_h, s'_v, c, t + 1)$       /* Fonction de coût voir Éq. 5.2 */
25   $Q(h, \bar{a}) \leftarrow Q(h, \bar{a}) + \frac{Q(h, \bar{a})' - Q(h, \bar{a})}{N(ha)}$ 
26   $V(h) \leftarrow \min_{a \in A} Q(h, a)$ 

```

---

### 5.2.1 Fonction heuristique et initialisation de la valeur

Nous choisissons dans notre algorithme d'initialiser la valeur des nouveaux nœuds. Cette valeur initiale bien qu'elle puisse être choisie aléatoirement [SILVER et al. 2010], puisqu'elle sera mise à jour rapidement, guide la recherche pour UCB1 dans les premières explorations. C'est pourquoi choisir une bonne estimation de la valeur initiale de l'état de croyance est important.

Dans ces travaux, nous avons décidé d'utiliser l'algorithme Dijkstra [DIJKSTRA 1959] pour estimer la distance entre une position et la position de l'objectif. Pour cela, nous avons pris la grille d'occupation (environnement) déterministe pour laquelle nous avons créé un graphe où chaque cellule est un nœud. La cellule contenant l'objectif devenant le nœud but. Les arêtes de ce graphe sont pondérées par la distance estimée pour passer d'une cellule à une autre. Une fois ce graphe créé il est possible de calculer avec l'algorithme de Dijkstra pour chaque nœud la distance minimale, qui est divisée par la vitesse de référence pour obtenir le temps du vol estimé pour atteindre le but. Ces calculs sont réalisés a priori et le résultat est stocké afin d'être seulement consulté lors de l'optimisation de la politique.

Ces valeurs sont alors utilisées par l'algorithme pour initialiser la valeur de chaque nouveau nœud. Quand un nouveau nœud dans l'arbre  $h$  est exploré pour chaque action  $a \in A$ , un nouveau nœud  $ha$  est créé. En utilisant la fonction de transition GNC, il est possible de calculer l'état moyen  $\bar{s}_h$  de la loi normale  $b_a$ . La cellule contenant cet état sera alors le nœud du graphe de Dijkstra qui servira à initialiser la valeur  $V_{\text{init}}(ha)$ .

### 5.2.2 Élagage de l'arbre de recherche

Étant donnée la complexité du problème, la taille de l'arbre de recherche associée est exponentielle en nombre d'époques  $t$  tel que  $(|A| \times |\Omega|)^t$ . C'est pourquoi pour éviter d'être confrontés à des problèmes de mémoire nous avons proposé une méthode pour élaguer certaines branches de l'arbre de recherche pendant l'optimisation. La problématique principale est d'élaguer les nœuds qui ne mèneront pas à une solution optimale sans élaguer ceux qui pourraient mener à une solution optimale.

Nous considérons donc pour un nœud  $h$  le nœud  $ha^*$  qui représente la meilleure action à un instant  $t$  tel que  $a^* = \arg \min_{a \in A} Q(h, a)$ . Nous proposons la condition d'élagage pour un nœud  $ha$  représentant une action  $a \in |A \setminus a^*|$  :

$$\left( Q(h, a) - c\sqrt{\frac{\log(N(h) + \rho)}{N(ha)}} \right) > \left( Q(h, a^*) - c\sqrt{\frac{\log(N(h) + \rho)}{N(ha^*) + \rho}} \right) \quad (5.6)$$

Supposons que pour  $\rho$  visites supplémentaires du nœud  $ha^*$ , la  $Q$ -valeur de la meilleure action  $Q(h, a^*)$  ne change pas. Si (Éq. 5.6) est satisfaite alors la stratégie UCB1 ne fera pas choisir le nœud  $ha$  même au bout de  $\rho$  prochains trials. Dans ce cas, nous décidons d'élaguer ce nœud, car nous ne le considérons plus intéressant.

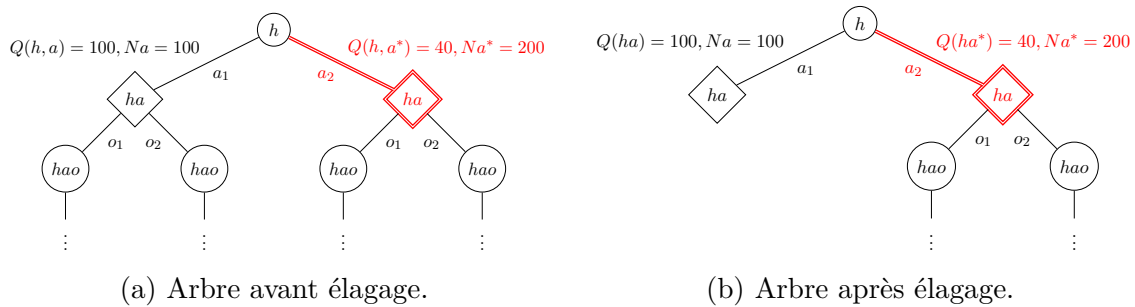


FIGURE 5.1 – Élagage de l’arbre de recherche.

Néanmoins, durant l’élagage ce sont les nœuds fils de  $ha$  qui sont supprimés afin de garder les valeurs de  $ha$ . Garder ces valeurs permet d’éviter que la stratégie UCB1 fasse explorer immédiatement ce nœud qu’il considérerait comme nouveau. De plus, il est important d’avoir réalisé un minimum de *trials* afin d’avoir assez d’information sur les nœuds (i.e.,  $Q(h, a)$  suffisamment représentatif de la vraie valeur de  $ha$ ). Dans la suite de ces travaux nous avons fixé un seuil  $minTrial = 100$  en dessous duquel ( $N(ha) < minTrial$ ) un nœud ne peut pas être élagué et  $\rho = 100$ . L’algorithme 11 et la figure 5.1 illustrent l’élagage d’un arbre de recherche.

Pour analyser le risque qu’un nœud élagué, soit de nouveau visité, nous avons vérifié de façon empirique cette méthode sur un exemple simple. Nous avons utilisé carte *WallBaffle* de la plate-forme de test avec l’ensemble de 10 actions. La figure 5.2 représente l’évolution du nombre de nœuds stockés lors de l’optimisation sur un exemple simple, avec et sans notre technique d’élagage de l’arbre. L’élagage a un impact important sur le nombre de nœuds gardés en mémoire lors de l’optimisation. En effet, en utilisant l’élagage nous constatons que au bout de 10000 *trials* le nombre de nœuds est diminué de moitié. De plus, après 100 000 *trials* nous n’avons pas constaté de nœuds élagués ayant été de nouveau visités. Donc la technique proposée permet dans ce problème d’économiser de l’espace mémoire sans affecter la performance d’optimisation.

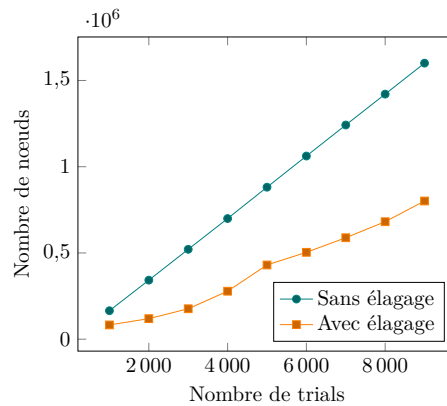


FIGURE 5.2 – Comparaison de l’évolution du nombre de nœud avec ou sans élagage.

---

**Algorithme 11 : POMPC-GO avec élagage**


---

```

1 Fonction POMPC-GO( $h, b_0, c, \gamma$ )
2    $h \leftarrow b_0$ 
3   while  $nbTrial < Nb_{max}$  do
4      $s_h \leftarrow$  sampling  $s_h$  from  $b_0$ 
5     Trial( $h, s_h, s_v, c, 0$ )
6     Prune( $h, c, \rho, minTrial$ )
7    $a^* \leftarrow \underset{a \in A}{\text{arg min}} Q(b_0, a)$ 
8 Fonction Prune( $h, c, \rho, minTrial$ )
9   if  $h$  is leaf then
10    return
11  foreach  $ha \in h$  do
12    if  $\left( Q(h, a) - c\sqrt{\frac{\log(N(h)+\rho)}{N(ha)}} \right) > \left( Q(h, a^*) - c\sqrt{\frac{\log(N(h)+\rho)}{N(ha^*)+\rho}} \right)$  and  $N(ha) > minTrial$  then
13      foreach  $hao \in ha$  do
14        Remove  $hao$ 
15    else
16      for  $hao \in ha$  do
17        Prune( $hao, c, \rho, minTrial$ )

```

---



### 5.3 Expérimentations

Dans ce chapitre, nous avons proposé une modélisation du problème sous forme d'un MOMDP ainsi qu'un algorithme de résolution POMCP-GO. Nous avons comparé l'algorithme proposé dans ce chapitre avec l'algorithme POMCP [SILVER et al. 2010]. Pour pouvoir comparer ces algorithmes, POMCP a été utilisé de façon hors-ligne, c'est-à-dire que chaque trial a commencé dans l'état de croyance initiale  $b_0$ . De plus, pour initialiser les nouveaux nœuds (état de croyance) nous avons utilisé la même fonction heuristique (Section 5.2.1) à la place d'appliquer la procédure *rollout*.

#### 5.3.1 Setup des expérimentations

**Les cartes** Nous avons utiliser deux cartes de la plate-forme de test (Figure 4.9) :

- *WallBaffle* : La carte contenant deux murs.
- *CubeBaffle* : La carte contenant deux cubes.

**Les métriques** Nous avons utilisé les métriques suivantes pour évaluer la qualité des politiques calculées :

- **La valeur de l'état de croyance initiale  $V(b_0)$  lors de l'optimisation.** Nous utilisons cette métrique afin de vérifier la convergence en valeur de la politique pour l'état de croyance initiale.
- **La valeur de l'état de croyance initiale  $V(b_0)$  simulée.** La valeur de  $b_0$  lors de l'optimisation est biaisée, car c'est la moyenne des coûts de tous les *trials* et donc ne représente pas la valeur réelle de  $b_0$ . La valeur de  $b_0$  simulée est calculée par la valeur moyenne de toutes les trajectoires réalisées en simulations en suivant la politique.
- **Le taux de succès.** Le taux de succès est le rapport entre le nombre de simulations qui arrive dans l'état but sur le nombre total de simulations effectué (en %).
- **Le temps de vol moyen.** Le temps de vol moyen permet de vérifier la minimisation du temps de vol.

**Les capteurs :** Les capteurs embarqués sont l'INS disponible tout le temps et le GPS dont la disponibilité probable varie selon la position dans l'environnement.

**Les actions :** Nous avons défini deux ensembles d'actions, le premier ensemble  $A_- = \{Nord, Sud, Ouest, Est\}$  composé de quatre directions de mouvement en 2D et l'ensemble d'action appelé « réduit » (noté  $A^-$ ) utilisant 10 directions (Figure 4.8).

**Conditions initiales** Certaines conditions initiales sont communes à toutes les cartes : Les autres conditions sont différentes en fonction de la carte :

- *WallBaffle* : Pour tous les tests réalisés sur cette carte l’objectif se trouve à la position  $(50, 80, 5)$  et l’état de croyance initial est une loi normale centrée en  $(50, 20, 5)$  sachant que les dimensions de la carte sont  $100 \times 100 \times 20$ , une unité étant équivalent à 2 mètres.
- *CubeBaffle* : Pour tous les tests réalisés sur cette carte l’objectif se trouve à la position  $(50, 80, 5)$  et l’état de croyance initial est une loi normale centrée en  $(60, 40, 5)$  sachant que les dimensions de la carte sont  $100 \times 100 \times 20$ , une unité étant équivalent à 2 mètres.

**Paramètres** Les algorithmes POMCP-GO et POMCP nécessitent un préérilage. La valeur des paramètres a été choisie après avoir effectué une série de tests empiriques :

- Le nombre maximum de *trials* a été fixé à  $Nb_{max} = 100000$ .
- Le coefficient d’exploration  $c$  pour la stratégie de sélection d’action UCB1 a été fixé à  $c = 100$ . Cette valeur a été choisie après avoir étudié empiriquement l’impact de plusieurs valeurs. Cette étude est donnée dans l’annexe A.
- La durée d’une époque de planification a été fixée à 4 secondes. Le choix de la durée de l’époque a été étudié dans l’annexe B.
- La pénalité de collision a été fixée à  $K = 450$ .
- Le nombre de simulations effectué afin d’évaluer chaque politique a été fixé à 1000. Le nombre de simulations devait être suffisamment grand pour être représentatif, c’est pourquoi nous avons étudié le choix du nombre de simulations dans l’annexe C.

## 5.3.2 Résultats

### 5.3.2.1 Carte *CubeBaffle*

Dans un premier temps, nous avons testé l’algorithme sur un problème réduit en deux dimensions avec l’ensemble d’action  $A_-$  sur la carte *CubeBaffle*.

Les résultats sont présentés dans les figures 5.3 et 5.4, les colonnes de gauche représentant les résultats avec l’algorithme proposé POMCP-GO et les colonnes de droite représentant les résultats de l’algorithme POMCP. L’évolution de la valeur de l’état de croyance initiale  $b_0$  montre que la valeur de  $b_0$  semble atteindre une même valeur pour les deux algorithmes à la fin des *trials*. Néanmoins, la valeur de  $b_0$  est en moyenne beaucoup plus faible avec l’algorithme POMCP (environ 69) qu’avec l’algorithme POMCP-GO (environ 96). Cependant, quand nous examinons la valeur de  $b_0$  lorsque les politiques sont simulées, nous constatons qu’avec POMCP-GO la valeur est en moyenne de 88 contre 113. La différence entre les valeurs de  $b_0$  lors de l’optimisation et lors des simulations pour POMCP-GO s’explique par le biais de l’algorithme. Dans l’algorithme, les  $Q$ -valeurs  $Q(h, a)$  des actions sont des moyennes de tous les *trials* qui commencent par  $a$  et non une moyenne des *trials* commençant par  $a$  et suivant la politique optimale. Étant donné que l’algorithme utilise la stratégie UCB1 pour sélectionner l’action à explorer lors des *trials*, l’algorithme va continuer à explorer même quand une politique sans risque de collision a été trouvée. C’est pourquoi la valeur de  $b_0$  optimisée

sera souvent supérieure à la valeur simulée. En revanche, la différence pour POMCP vient du fait que l'algorithme n'a toujours pas convergé. La valeur de l'optimisation est plus faible que la valeur simulée, car POMCP arrête les *trials* quand un nouveau nœud est rencontré et donc POMCP n'a toujours pas rencontré de nœuds but ou collision, alors la valeur retournée est la valeur de la fonction heuristique qui est optimiste.

Cela se confirme avec le taux de succès (Figure 5.4) qui montre qu'en moyenne l'algorithme POMCP donne un moins bon taux de succès 85% contre 96%. De plus, l'importante variation de la valeur de l'état de croyance initiale entre les différentes politiques se retrouve dans la variation du taux de succès. Nous constatons aussi que pour POMCP ce sont les politiques ayant le temps de vol moyen le plus faible qui ont le taux de succès le plus élevé. Les chemins parcourus (Figure 5.5) pour la politique ayant le plus de succès pour chaque algorithme confirment que POMCP a calculé une politique qui passe entre les deux obstacles et en minimisant le temps de vol alors que dans le cas de POMCP-GO la politique évite les risques de collisions en contournant les obstacles. Étant donné que POMCP arrête un *trial* quand il rencontre un nouveau nœud, l'algorithme va avoir à tendance à optimiser d'abord les nœuds proches de  $b_0$  et ensuite optimiser les nœuds plus éloignés. C'est pourquoi la fonction heuristique va guider POMCP vers des nœuds qui minimisent le temps de vol, car la fonction heuristique ne donne pas d'information sur le risque de collision. Une fois que POMCP rencontrera une collision, il aura tendance à optimiser l'action qui a mené à cette collision plutôt que d'explorer un autre chemin. Alors que POMCP-GO lors d'un *trial* explore jusqu'à rencontrer un état terminal (collision ou but). Ainsi s'il rencontre une collision en essayant de passer entre les obstacles, il va avoir tendance à explorer ensuite un chemin différent au lieu d'optimiser le chemin précédent.

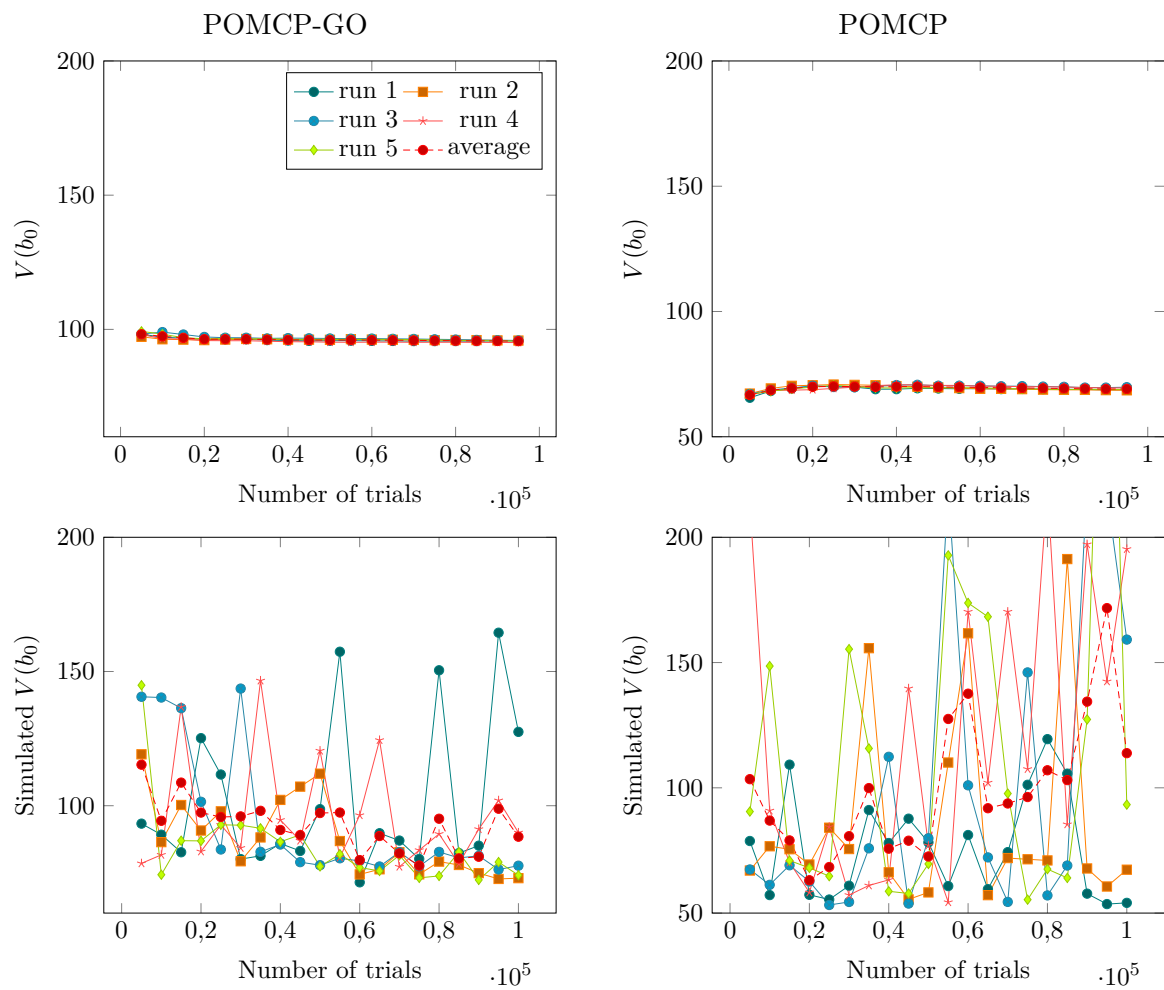


FIGURE 5.3 – Évolution de la fonction de valeur de la planification lors de l'optimisation (en haut) et lors des simulations (en bas) sur la carte *CubeBaffle* pour l'ensemble d'actions  $A_-$ .

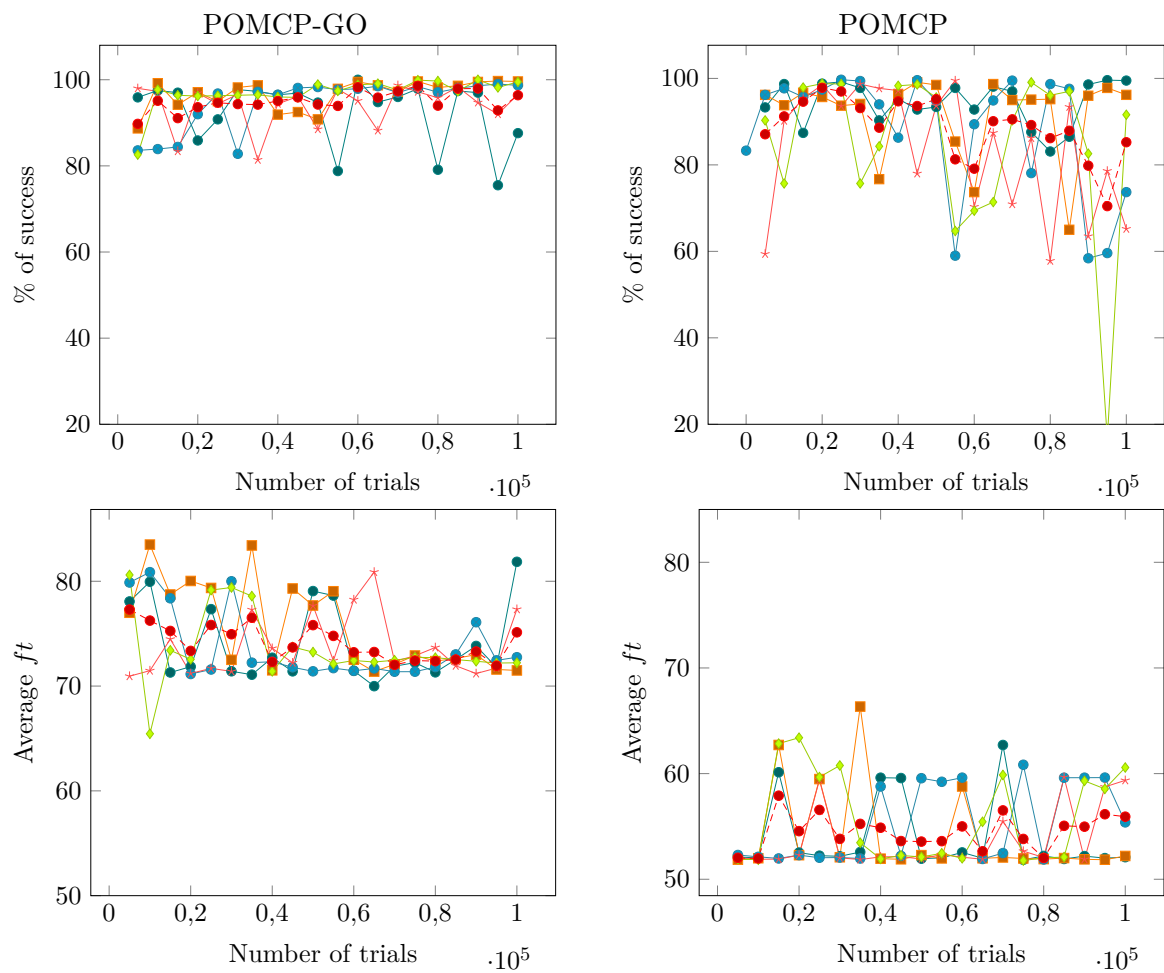
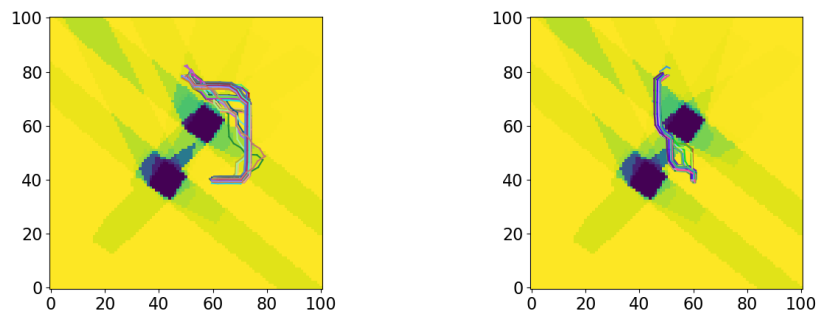


FIGURE 5.4 – Résultats de la planification avec le taux de succès (en haut) et le temps du vol moyen (en bas) sur la carte *CubeBaffle* pour l'ensemble d'actions  $A_-$ .



(a) Chemins simulé (POMCP-GO). (b) Chemins simulé (POMCP).

FIGURE 5.5 – Exemples des chemins simulés à partir des différentes politiques calculées pour l'algorithme POMCP-GO (run 2) et POMCP (run 1).

### 5.3.2.2 Carte *WallBaffle*, ensemble d'actions $A^-$

Nous comparons les deux algorithmes sur la carte *WallBaffle* avec l'ensemble réduit des actions  $A^-$  avec 10 directions de mouvement en 3D.

Les résultats sont présentés dans les figures 5.6 et 5.7. Nous constatons que les résultats pour l'évolution de la valeur de  $b_0$  sont similaires aux résultats sur le problème réduit. En effet, nous constatons que pour POMCP-GO  $V(b_0)$  tends vers 92 lors de l'optimisation et 75 lors des simulations, ce qui montre un biais relativement petit. Tandis que POMCP à une valeur de  $b_0$  moyenne qui converge vers 49 lors de l'optimisation, mais qui est de 213 en moyenne lors des simulations. Nous pouvons donc déduire que POMCP a un biais très élevé, car les valeurs lors de l'optimisation sont proches de la valeur donnée par la fonction heuristique et sont très éloignées des valeurs obtenues lors des simulations. Les politiques calculées n'ont jamais atteint le but durant l'optimisation et donc les valeurs sont celles du temps de vol parcouru plus celle donnée par la fonction heuristique qui est optimiste et ne prend pas en compte le risque de collision.

Le taux de succès pour POMCP-GO dans cette configuration est en moyenne de 99.7%, alors que le taux de succès est variable pour les politiques calculées par POMCP et sont compris entre 43% et 71% avec une moyenne de 63%. Les politiques de POMCP n'ont jamais atteint l'objectif et donc lors des simulations lorsqu'un nœud n'ont exploré lors de l'optimisation est rencontré le drone utilise la politique heuristique. Cette politique heuristique ne prenant pas en compte l'incertitude sur la position, elle est plus susceptible de mener le drone dans un état collision.

Les chemins simulés (Figure 5.8) montrent que dans le cas des politiques calculées par POMCP-GO les chemins survolent les obstacles afin de minimiser le risque de collision et d'obtenir une meilleure probabilité de disponibilité du GPS (Figure 4.9). Alors que pour la meilleure politique calculée par POMCP, les chemins simulés passent entre les obstacles sans essayer de les survoler, ce qui est dû au fait que le drone utilise la politique heuristique qui minimise le temps de vol sans minimiser le risque de collision.

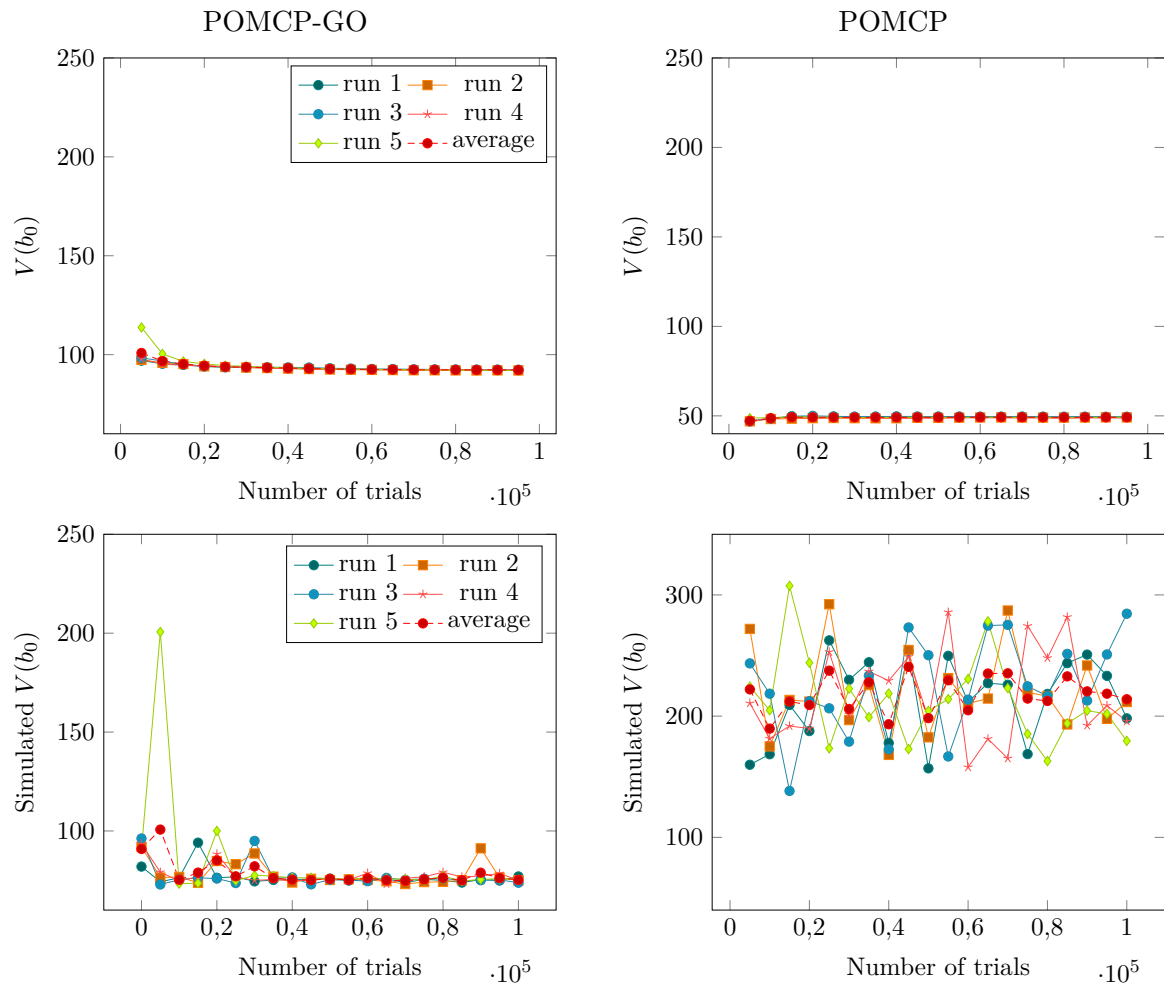


FIGURE 5.6 – Évolution de la fonction de valeur de la planification sur la carte *CubeBaffle* pour l'ensemble réduit d'actions et une pénalité  $K = 450$ .

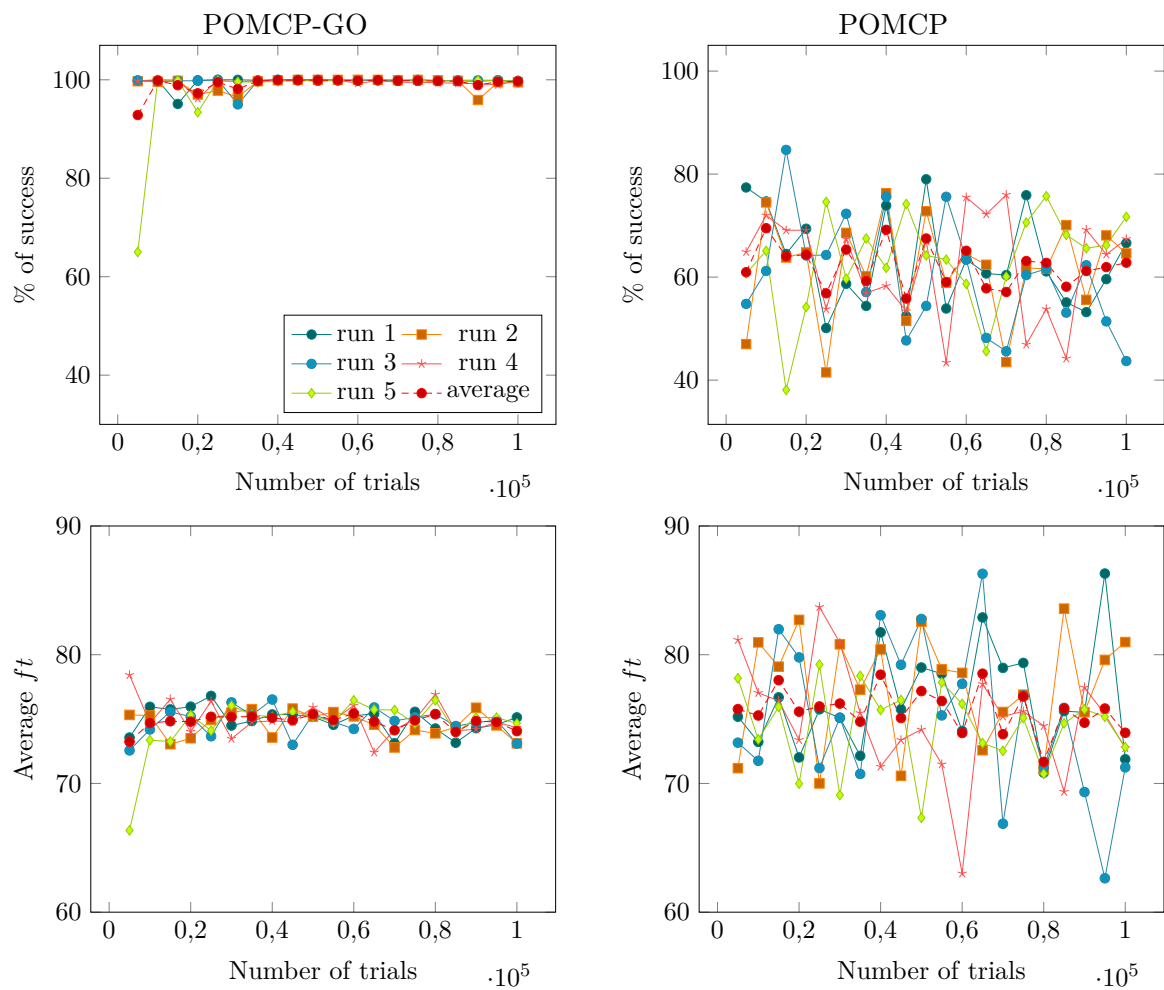
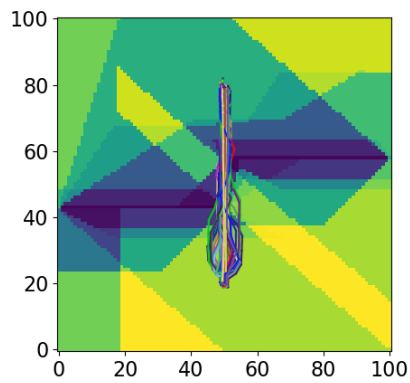
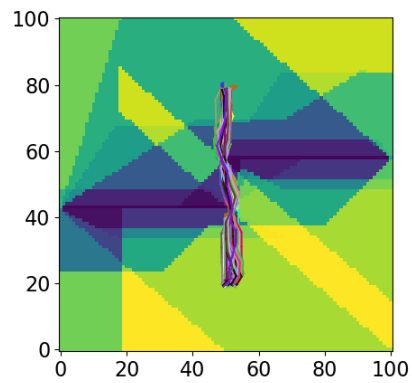


FIGURE 5.7 – Résultats de la planification sur la carte *CubeBaffle* pour l'ensemble réduit d'actions et une pénalité  $K = 450$ .

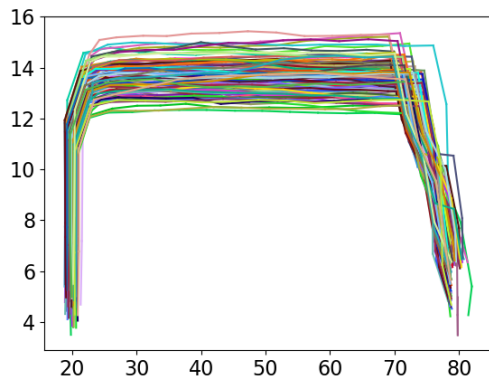




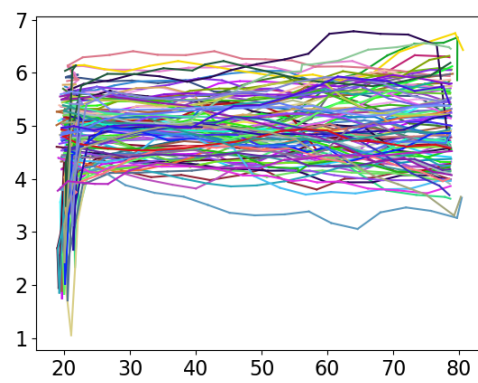
(a) Chemins simulé (POMCP-GO).



(b) Chemins simulé (POMCP).



(c) Chemins simulé (axe z, POMCP-GO).



(d) Chemins simulé (axe z, POMCP).

FIGURE 5.8 – Exemples des chemins simulé à partir des différentes politiques calculé pour l’algorithme POMCP-GO et POMCP.

## 5.4 Bilan

Dans ce chapitre, nous avons proposé un deuxième modèle MOMDP et un algorithme de résolution appelé POMCP-GO. Il a été comparé avec l'algorithme de la littérature POMCP sur deux problèmes de planification. Le premier problème étant un problème de taille réduit en deux dimensions et quatre directions de mouvements. Le deuxième problème étant un problème de plus grande taille.

Les résultats permettent de constater que POMCP-GO permet de calculer des politiques ayant un risque plus faible de collision que POMCP en augmentant le temps moyen de vol jusqu'au but avec le même nombre de trials. La différence de performance entre ces deux algorithmes vient de leur différence de fonctionnement. D'un côté POMCP, par son fonctionnement, va principalement faire de l'optimisation locale le long d'un même chemin. Chaque *trial* se terminant quand un nouveau nœud (état de croyance) est rencontré, il va retourner la valeur donnée par la fonction heuristique. Cette valeur estimant le temps minimal pour atteindre l'objectif, cela va toujours favoriser l'exploration de l'algorithme le long d'une même trajectoire. En effet, toute action s'éloignant de cette trajectoire va avoir une valeur supérieure à l'action qui va dans la direction du chemin le plus court. Le coefficient d'exploration fera explorer ces actions, mais elles resteront en valeur moins intéressantes, car elles ne prennent pas en compte les risques de collisions. En cas de collision, le *trial* suivant optimisera quelques états de croyance en amont du nœud collision. C'est pourquoi POMCP dans ces problèmes fait de l'optimisation locale, car il dépend beaucoup plus fortement de la valeur initialisée par la fonction heuristique.

POMCP-GO utilise la même fonction heuristique pour initialiser les nœuds. Cependant, comme un *trial* se termine quand un état terminal est trouvé, la valeur retournée représente mieux la valeur espérée en suivant cet historique d'action-observation. C'est pourquoi lors des premiers *trials* si les chemins explorés passent entre les deux obstacles et que des collisions sont détectées l'algorithme changera plus rapidement de chemin pour explorer des chemins plus sûrs.

Ces résultats montrent l'intérêt de l'algorithme POMCP-GO proposé par rapport à POMCP pour résoudre les problèmes de planification étudiés dans cette thèse. Dans le chapitre suivant, nous étudierons plus en détail la fonction de coût pour définir une pénalité de collision en fonction d'un taux de collision maximum acceptable. Nous appliquerons cette méthode avec l'algorithme POMCP-GO proposé dans ce chapitre.

# 6

## Étude du lien entre la fonction de coût et les politiques acceptables

### Sommaire

---

<b>6.1</b>	<b>Étude du lien entre la fonction de coût et les politiques acceptables .</b>	<b>97</b>
6.1.1	Réécriture de la valeur de l'état de croyance initial en matière de risque de collision et de temps de vol . . . . .	98
6.1.2	Détermination de la valeur de la pénalité de collision $K$ pour un risque de collision maximum autorisé . . . . .	99
<b>6.2</b>	<b>Expérimentations . . . . .</b>	<b>102</b>
6.2.1	Setup des expérimentations . . . . .	102
6.2.2	La politique le plus sûre pour obtenir le valeur « Tmax » . . . . .	103
6.2.3	Planification de chemins avec un taux de collision maximum acceptable donné . . . . .	115
<b>6.3</b>	<b>Bilan . . . . .</b>	<b>120</b>

---

Le modèle de planification basé sur un MOMDP présenté dans le chapitre précédent a pour objectif de calculer une politique sans risque de collision et minimisant le temps de vol. Mais pour certaines missions, il serait plus intéressant et important d'arriver au but rapidement même en autorisant un certain risque de collision. Nous étudierons dans ce chapitre une méthode permettant de définir la valeur de  $K$  à partir d'un risque de collision maximum acceptable qui est spécifié par la mission.

### 6.1 Étude du lien entre la fonction de coût et les politiques acceptables

Nous rappelons que le modèle de planification basé MOMDP présenté dans le chapitre précédent a pour objectif de calculer une politique qui va minimiser le temps de vol total espéré et le risque de collision, ce qui va donc maximiser l'efficacité et la sûreté de l'opération. Dorénavant, nous aimerions pouvoir déterminer la pénalité de collision  $K$  (dans la fonction du coût) qui autorise un taux maximum de collision qui est acceptable pour une mission donnée.

On rappelle que la fonction de coût est définie comme suit :

$$C(s_t) = \begin{cases} K - t \times f = K_t & \text{si } s_t \text{ en collision} \\ 0 & \text{si } s_t \text{ est un état but} \\ f & \text{sinon} \end{cases} \quad (6.1)$$

où  $f$  est la durée constante d'une époque,  $K$  la pénalité de collision et  $t$  l'horizon.

### 6.1.1 Réécriture de la valeur de l'état de croyance initial en matière de risque de collision et de temps de vol

Nous pouvons montrer qu'en utilisant la fonction de coût définie dans l'Équation 6.1 et avec  $\gamma = 1$ , la valeur initiale de l'état de croyance  $V(b_0)$  peut être réécrite comme une somme sur le temps de vol total espéré jusqu'à l'objectif et une pénalité de collision constante  $K$  pondérées par la probabilité de collision sur l'ensemble des parcours probables de la politique.

À partir de la définition de la fonction de valeur (Éq. 5.5),  $V(b_0)$  peut être étendu comme suit. Sachant la valeur de l'état de croyance initiale pour les états de croyance  $b_0$  dans lesquels on est sûr qu'il n'y a pas de collision,  $V(b_0)$  est redéfini comme suit :

$$\begin{aligned} V^{\pi^*}(b_0) &= \mathcal{C}(b_0) + \sum_{s_{v_1} \in \mathcal{S}_v} p(s_{v_1}|b_0, \pi(b_0))V(b_{\pi(b_0)}^{s_{v_1}}) \\ &= f + \sum_{s_{v_1} \in \mathcal{S}_v} p(s_{v_1}|b_0, \pi(b_0))V(b_{\pi(b_0)}^{s_{v_1}}) \\ &= \sum_{s_{v_1} \in \mathcal{S}_v} p(s_{v_1}|b_0, \pi(b_0)) \left( f + C(b_{\pi(b_0)}^{s_{v_1}}) + \sum_{s_{v_2} \in \mathcal{S}_v} p(s_{v_2}|b_{\pi(b_0)}^{s_{v_1}}, \pi(b_{\pi(b_0)}^{s_{v_1}}))V(b_{\pi(b_{\pi(b_0)}^{s_{v_1}})}^{s_{v_2}}) \right) \\ &= p(F_{Col_1} = 1|b_0, \pi(b_0))K + p(s_{h_1} \in G|b_0, \pi(b_0))f + \sum_{s_{v_1} \in \mathcal{S}_v, F_{Col_1}=0} p(F_{Col_1} = 0, s_{h_1} \notin G|b_0, \pi(b_0)) \\ &\quad \left( 2f + \sum_{s_{v_2} \in \mathcal{S}_v} p(s_{v_2}|b_{\pi(b_0)}^{s_{v_1}}, \pi(b_{\pi(b_0)}^{s_{v_1}}))V(b_{\pi(b_{\pi(b_0)}^{s_{v_1}})}^{s_{v_2}}) \right) \\ &= p_{c_1}K + p_{g_1}f + (p_{c_2}K + p_{g_2}2f + (p_{c_3}K + p_{g_3}3f + \dots)) \\ &= \dots \\ &= \sum_{t=1}^{\infty} p_{c_t}K + \sum_{t=1}^{\infty} p_{g_t}tf = p_cK + (1 - p_c)T \end{aligned} \quad (6.2)$$

où  $p_{c_t}$  est la probabilité qu'une collision soit observée à l'horizon  $t$  depuis l'état de croyance initiale  $b_0$  et en suivant la politique  $\pi$ . De façon similaire  $p_{g_t}$  est la probabilité d'atteindre le but à la profondeur  $t$ . Alors,  $p_c = \sum_{t=1}^{\infty} p_{c_t}$  représente la probabilité de collision totale depuis l'état de croyance initiale  $b_0$  quand la politique  $\pi$  est suivie. De plus  $T$  est le temps de vol total espéré pour atteindre le but depuis l'état de croyance initiale sachant que  $F_{Col_t} = 0$  pour  $\forall t \geq 0$ .

Grâce à la définition de la fonction de coût (Éq. 6.1), la valeur de l'état de croyance initiale

$V(b_0)$  devient une fonction linéaire de la pénalité de collision  $K$  et dont la pente représente la probabilité de collision  $p_c$ . Dans la suite de ces travaux, cette dépendance linéaire sera utilisée pour déterminer la valeur  $K$  pour un risque de collision maximum autorisé qui est spécifié par la mission.

### 6.1.2 Détermination de la valeur de la pénalité de collision $K$ pour un risque de collision maximum autorisé

Nous considérons un problème de planification pour lequel il existe trois politiques extrêmes :

— **La politique heuristique, notée  $\pi_h$ .**

Cette politique minimise le temps de vol espéré pour atteindre l'objectif, sans considérer l'incertitude de l'état initial ni le risque de collision induit par les erreurs de localisation (navigation) et d'exécution du chemin. C'est pourquoi le temps de vol espéré  $T_h$  (sachant  $F_{Col_t} = 0$ ) qui résulte de cette politique doit être le temps de vol le plus court possible pour une mission donnée. Cela garantit que  $T_h \leq T$  pour n'importe quelle politique  $\pi$ . Cependant, le risque de collisions  $p_{ch}$  de cette politique peut être grand et même atteindre 1, car ce risque n'est pas pris en compte lors de l'optimisation.

— **La politique la plus sûre sans risque de collision notée  $\pi_s$ .**

Cette politique minimise le temps de vol espéré tout en n'autorisant aucun risque de collision. Pour cela, pendant l'optimisation toutes les incertitudes du modèle sont prises en compte, tel que l'incertitude liée à la disponibilité des capteurs ainsi qu'à l'erreur d'exécution. Il n'y a aucune garantie que cette politique existe pour une mission donnée, mais dans cette section nous faisons l'hypothèse qu'elle existe. La probabilité de collision qui résulte de cette politique devrait être égale à  $p_c = 0$ . Le temps de vol espéré donné par cette politique donne une borne supérieure de  $V^{\pi^*}(b_0)$ , en effet si l'on utilise la fonction de valeur (Éq. 6.2) alors le coût optimal  $V^{\pi^*}(b_0) = p_c^*K + (1 - p_c^*)T^* \leq V^\pi(b_0)$  quelle que soit la politique  $\pi$  et nous notons alors  $V^{\pi_s}(b_0) = T_{max}$ .

— **La politique de collision notée  $\pi_c$ .**

Cette politique amène toujours un véhicule dans un état collision. Cette politique a donc  $p_c = 1$ .


Nous avons vu que la fonction de valeur pour l'état de croyance initial (Éq. 6.2) est linéaire en la pénalité de collision  $K$  de même que la pente de la fonction est donnée par la probabilité de collision  $p_c$ . La figure 6.1 affiche les fonctions de valeurs des trois politiques extrêmes définies précédemment par rapport à la pénalité de collision  $K$ . Dans cette figure, chaque politique est représentée par une ligne. Pour un  $K$  donné, la ligne de la politique la plus basse, minimise la valeur  $V(b_0)$  et elle est choisie comme étant la politique optimale.



On note que les politiques qui nous intéressent ont un temps de vol espéré  $T \leq T_{max}$  (plus efficace que  $\pi_s$ ) et  $p_c \leq p_{ch}$  (plus sûre que  $\pi_h$ ). Toutes politiques  $\pi$  avec le temps de vol espéré  $T$  intersectent la ligne  $\pi_c$  à  $K = T$ . C'est pourquoi la pénalité de collision doit être choisie

comme suit  $K > T_{max} \geq T$  afin que la politique de collision  $\pi_c$  ne devienne jamais la politique optimale.

Nous considérons maintenant une probabilité de collision maximale acceptable  $p_{thd}$  pour une mission. Nous imposons la condition que la politique acceptable pour la mission doit avoir une probabilité de collision inférieure à cette probabilité maximale de la façon suivante :  $p_c \leq p_{thd}$ . Pour toutes les politiques avec  $p_c \leq p_{thd}$ , l'équation suivante est satisfaite par le fait que  $K \geq T$  :

$$V^\pi(b_0) = p_c K + (1 - p_c)T \leq p_{thd}K + (1 - p_{thd})T \quad (6.3)$$

De plus, la politique heuristique  $\pi_h$  donne la borne inférieure en temps de vol total espéré. C'est-à-dire que le côté droit de l'inégalité (Éq. 6.3) a pour borne inférieure  $p_{thd}K + (1 - p_{thd})T_h$ . C'est pourquoi, il est possible de dire que si  $V^\pi(b_0) \leq p_{thd}K + (1 - p_{thd})T_h$ , (Éq. 6.3) est satisfaite pour tous les  $T \geq T_h$ . Par conséquent, n'importe quelle politique  $\pi$  qui satisfait cette condition garantit que sa probabilité de collision  $p_c$  ne dépasse pas le risque de collision maximum autorisé  $p_{thd}$ . Alors une politique « imaginaire » avec  $p_c = p_{thd}$  et  $T_h$  (Figure 6.1 ) donne la borne supérieure de la politique optimale acceptable.

On rappelle que la politique la plus sûre  $\pi_s$  donne la borne supérieure de la valeur de l'état de croyance initiale pour toutes les politiques optimales possibles, de la façon suivante :  $V^\pi(b_0) \leq V^{\pi_s}(b_0) = T_{max}$ . Par conséquent, si nous choisissons la valeur de la pénalité de collision  $K$  de telle façon que  $\pi_s$  (Figure 6.1 ) et la politique « imaginaire » (Figure 6.1 ) s'intersectent, il est garanti que toute politique qui satisfasse :

$$V^\pi(b_0) \leq V^{\pi_s}(b_0) = p_{thd}K + (1 - p_{thd})T_h = T_{max} \quad (6.4)$$

a donc  $p_c \leq p_{thd}$ . Parce que la politique  $\pi_s$  est supposée exister, la politique optimale a la valeur  $V^{\pi^*}(b_0) \leq T_{max}$  et donc elle est garantie d'être acceptable (i.e.,  $p_c^* \leq p_{thd}$ ).

La valeur de  $K$  est dérivée (Éq. 6.4) de la façon suivante :

$$K(T_{max}, T_h, p_{thd}) = T_h + \frac{\overbrace{(T_{max} - T_h)}^{\text{perte maximale de temps de vol}}}{\underbrace{p_{thd}}_{\text{probabilité de collision maximale autorisable}}} \quad (6.5)$$

Nous allons dans la suite de ce chapitre appliquer cette méthode sur différents problèmes, mais pour cela il est nécessaire de connaître  $T_{max}$ .

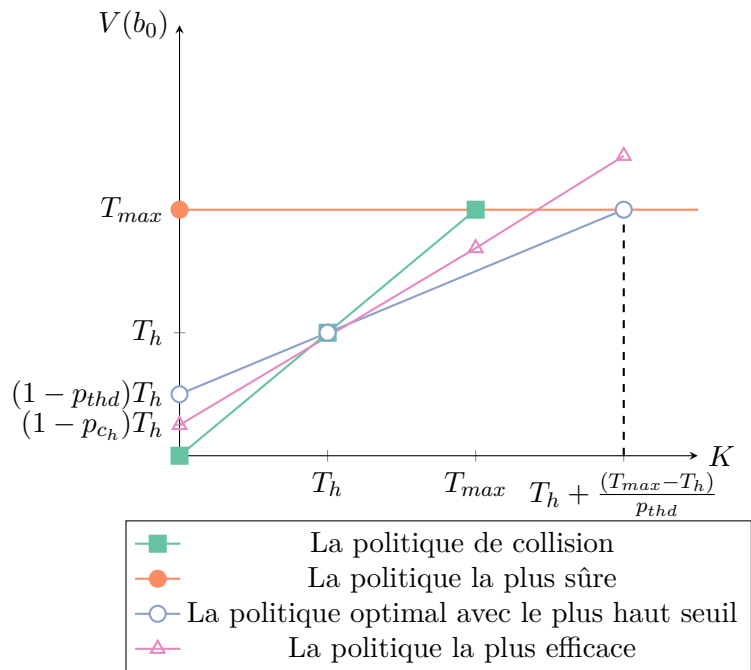


FIGURE 6.1 – Représentation de  $V(b_0)$  en fonction linéaire de la pénalité de collision  $K$ .

## 6.2 Expérimentations

Nous avons effectué plusieurs test afin d'étudier le comportement en pratique de la méthode proposée et ainsi montrer, que pour une pénalité de la collision calculée (Éq. 6.5), nous obtenons la politique minimisant le temps de vol tout en respectant le taux de collision souhaité ( $\leq p_{thd}$ ).

### 6.2.1 Setup des expérimentations

**Les cartes** Nous avons utilisé trois cartes de la plate-forme de test :

- *WallBaffle* : La carte contenant deux murs.
- *CubeBaffle* : La carte contenant deux cubes.
- San Diego : La carte la plus complexe et réaliste.

**Conditions initiales** Certaines conditions initiales sont communes à toutes les cartes :

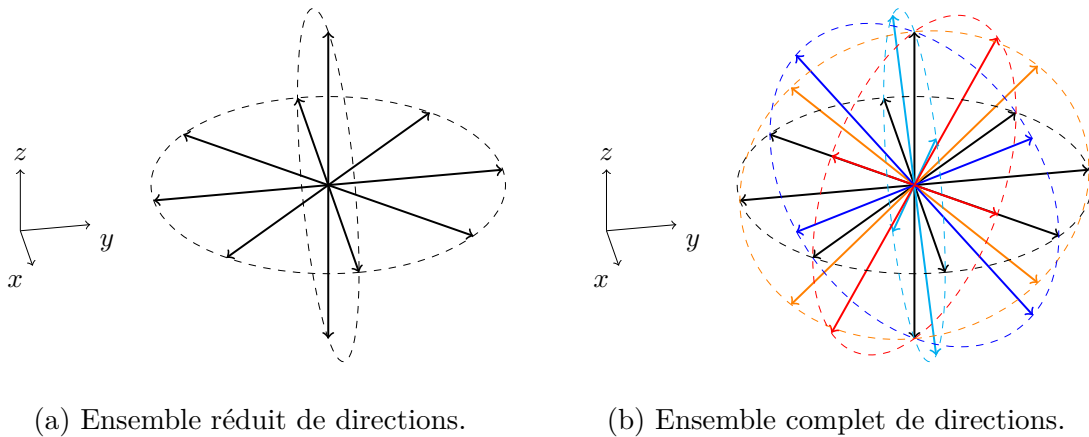
**Les capteurs :** Les capteurs embarqués sont l'INS disponible tout le temps et le GPS dont la disponibilité probable varie selon la position dans l'environnement.

**Les actions :** Pour rappel, une action est composé d'une direction ( $\mathcal{V}_{ref}$ ) et d'un mode de navigation (capteur à utiliser : sans ou avec GPS dans notre cas). Nous avons défini deux ensembles d'actions, le premier ensemble d'action appelé « réduit » (noté  $A^-$ ) est l'ensemble d'action n'utilisant que 10 directions en 3D (Figure 6.2a) soit 20 actions, et l'ensemble des actions appelé « complet » (noté  $A$ ) contenant 26 directions en ajoutant des diagonales monter/descendre (Figure 6.2b) soit 52 actions. Les autres conditions sont différentes en fonction de la carte :

- *WallBaffle* : Pour tous les tests réalisés sur cette carte l'objectif se trouve en  $(50, 80, 5)$  et l'état de croyance initial est une loi normale centrée en  $(50, 20, 5)$  (pour la position initiale du drone) sachant que les dimensions de la carte sont  $100 \times 100 \times 20$ , une unité étant équivalent à 2 mètres.
- *CubeBaffle* : Pour tous les tests réalisés sur cette carte l'objectif se trouve en  $(50, 80, 5)$  et l'état de croyance initial est une loi normale centrée en  $(50, 20, 5)$  (pour la position initiale du drone) sachant que les dimensions de la carte sont  $100 \times 100 \times 20$ , une unité étant équivalent à 2 mètres.
- San Diego : Pour tous les tests réalisés sur cette carte l'objectif se trouve en  $(100, 70, 5)$  et l'état de croyance initial est une loi normale centrée en  $(50, 20, 5)$  (pour la position initiale du drone) sachant que les dimensions de la carte sont  $217 \times 167 \times 21$ , une unité étant équivalent à 4 mètres.

Pour chaque test, nous effectuons cinq optimisations pour mieux analyser les performances de l'algorithme.





(a) Ensemble réduit de directions.

(b) Ensemble complet de directions.

FIGURE 6.2 – Représentation des ensembles de directions  $\mathcal{V}_{\text{ref}}$  utilisés.

### Paramètres

- Le coefficient d'exploration  $c$  pour la stratégie de sélection d'action UCB1 a été fixé à  $c = 0.222 \times K$ .
- La durée d'une époque de planification a été fixée à  $f = 4$  secondes.
- Le nombre de simulations effectué afin d'évaluer chaque politique a été fixé à 1000.
- Le nombre de *trials* est fixé à  $10^5$ .

## 6.2.2 La politique la plus sûre pour obtenir la valeur « Tmax »

### 6.2.2.1 Comparaison de deux pénalités de collisions

Nous avons vu précédemment (Section 6.1) qu'il était possible de définir la pénalité de collision  $K$  à partir d'un taux de collision maximal acceptable. Pour cela, il est nécessaire de trouver la politique dite « la plus sûre »  $\pi_s$  (en supposant qu'elle existe). La première étape est de choisir une pénalité de collision  $K$  suffisamment grande pour retrouver cette politique. Nous avons testé deux pénalités de collisions différentes, une pénalité très élevée  $K = 10^5$  et une pénalité beaucoup plus faible  $K = 450$ .

Les figures 6.3 et 6.4 présentent les résultats pour l'ensemble réduit des actions ( $A^-$ ). La colonne de gauche de chaque figure montre les résultats des politiques pour la plus grande pénalité de collision ( $K = 10^5$ ) et la deuxième colonne montre les résultats pour les politiques calculées avec la plus faible pénalité de collision ( $K = 450$ ).

À la fin des *trials* quel que soit  $K$ , les politiques semblent avoir atteint la même valeur. Les valeurs lors des optimisations sont supérieures à celles simulées, ce qui est dû au biais dans le calcul de la valeur de  $b_0$  lors de l'optimisation comme expliqué dans le chapitre 5.

L'évolution du taux de succès de chaque politique calculée est affichée dans la figure 6.4. Quelle que soit la pénalité de collision, l'algorithme calcule des politiques dont le taux de

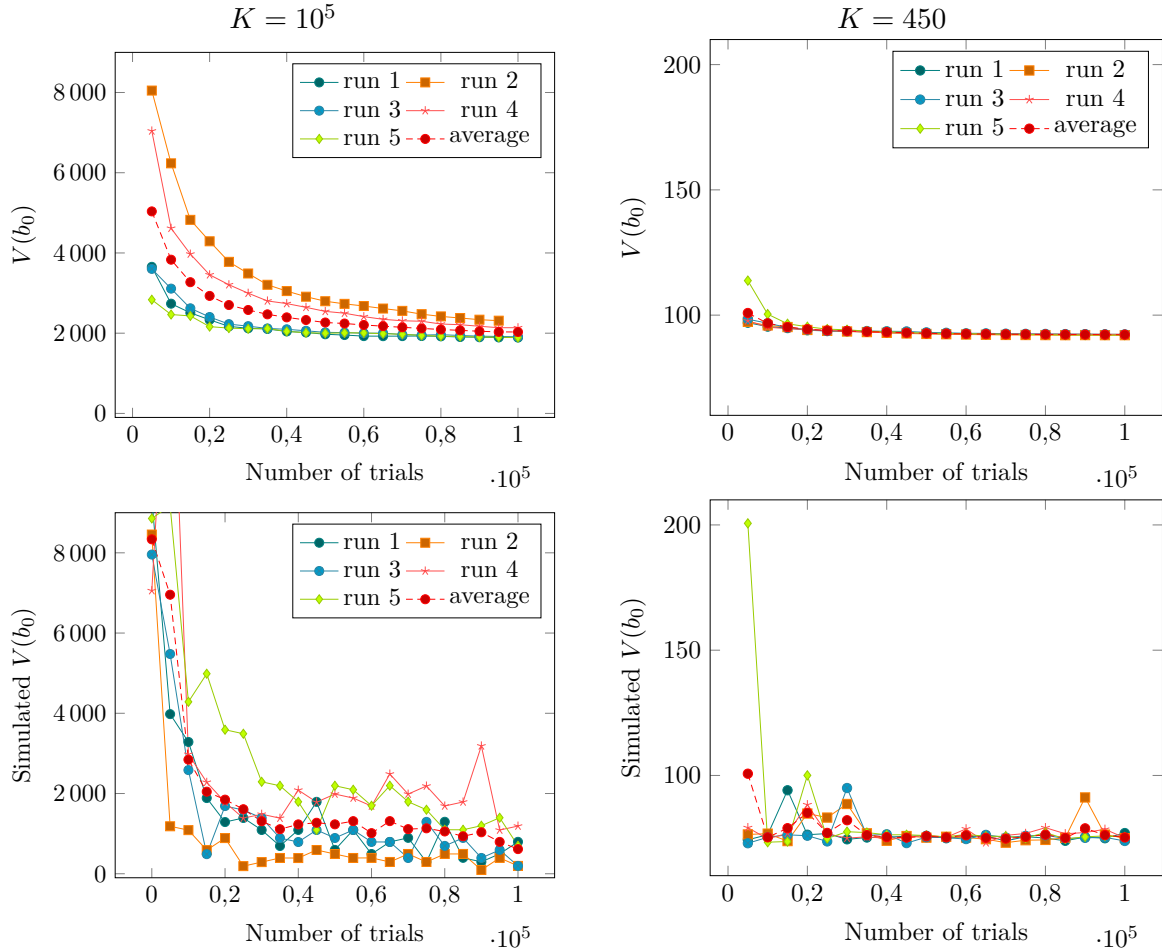


FIGURE 6.3 – Évolution de la fonction de valeur lors de la planification sur la carte *WallBaffle* pour l'ensemble réduit des actions  $A^-$  et pour les pénalités de collision  $K = 10^5$  et 450.

succès atteint très rapidement un taux de succès proche de 100%. Pour  $K = 10^5$  le temps de vol est plus élevé que pour  $K = 450$ . La différence du temps de vol peut s'expliquer par l'impact de la pénalité de collision sur l'optimisation. En effet, pour  $V(b_0) = p_c K + (1 - p_c)T$  quand  $K$  est grand la probabilité de collision joue beaucoup sur la valeur de  $b_0$  et donc l'optimisation sera tout d'abord faite sur la probabilité de collision avant de minimiser le temps de vol. Alors, l'optimisation tend vers des politiques qui évitent plus les collisions et donc en les contournant plus ce qui augmentera forcément le temps de vol moyen. Le coût associé à cette augmentation de temps est négligeable par rapport au coût associé au risque de collision.

Cela est visible sur les chemins simulés pour la politique qui donne les meilleurs résultats pour chaque  $K$  (Figure 6.5). Nous constatons que les chemins simulés pour  $K = 450$  prennent moins de hauteur pour éviter les murs que ceux pour  $K = 10^5$ . Pour la politique calculée avec  $K = 10^5$ , plusieurs chemins à deux altitudes différentes ont été empruntés. Cela est dû aux différentes observations qu'a reçu le drone lors de l'exécution de la politique pendant les 1000

simulations. Les chemins à plus basse altitude sont ceux qui ont eu le plus le GPS disponible que ceux à plus haute altitude.

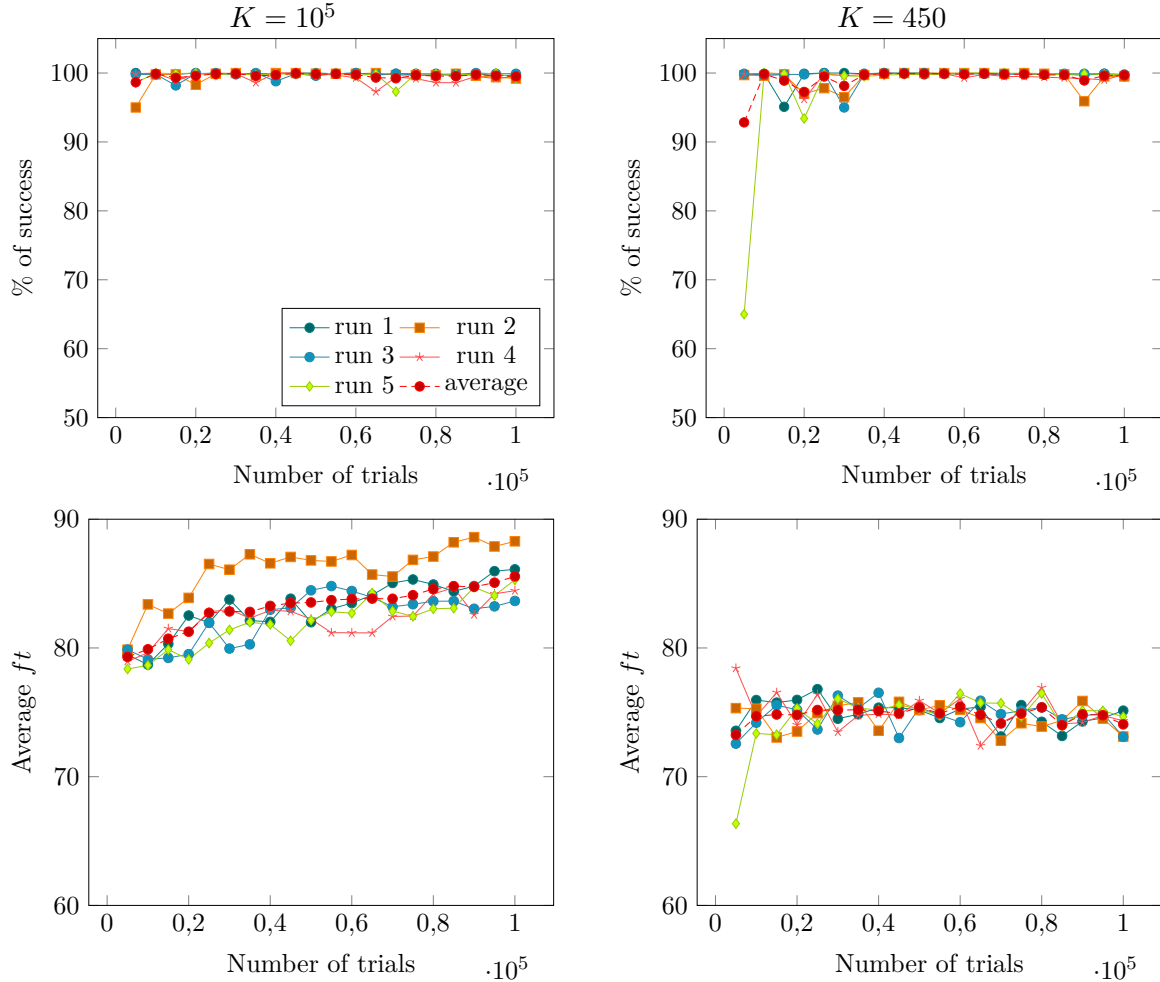


FIGURE 6.4 – Résultats de la planification sur la carte *WallBaffle* pour l'ensemble réduit des actions et pour les pénalités de collision  $K = 10^5$  et 450.

Nous avons ensuite testé pour l'ensemble complet des actions ( $A$ ) pour la même configuration (carte). Nous avons constaté qu'aucune politique ne tend vers une même valeur que soit pour  $K = 450$  ou  $K = 10^5$  dans le nombre de *trials* imparti pendant l'optimisation. Ces résultats viennent de l'augmentation de la taille du problème qui augmente la dimension de l'espace de recherche ( $(|A| \times |\Omega|)^t$ ,  $|A^-| = 20$ ,  $|A| = 52$ ).

En nous basant sur les résultats présentés et la figure 6.6 qui présente le résultat moyen pour chaque scénario, nous pouvons dire que le coût de collision a un impact faible sur la convergence et la qualité des résultats, car les politiques calculées ont un taux proche de 100%. Néanmoins, nous constatons qu'augmenter le nombre d'actions augmente la complexité de résolution. Les politiques ont alors besoin de plus de *trials* pour converger quelle que soit la pénalité de collision. En nous basant sur ces résultats, dans cette thèse, nous avons choisi d'utiliser une pénalité de  $K = 450$  pour obtenir la politique la plus sûre  $\pi_s$  sur toutes les cartes

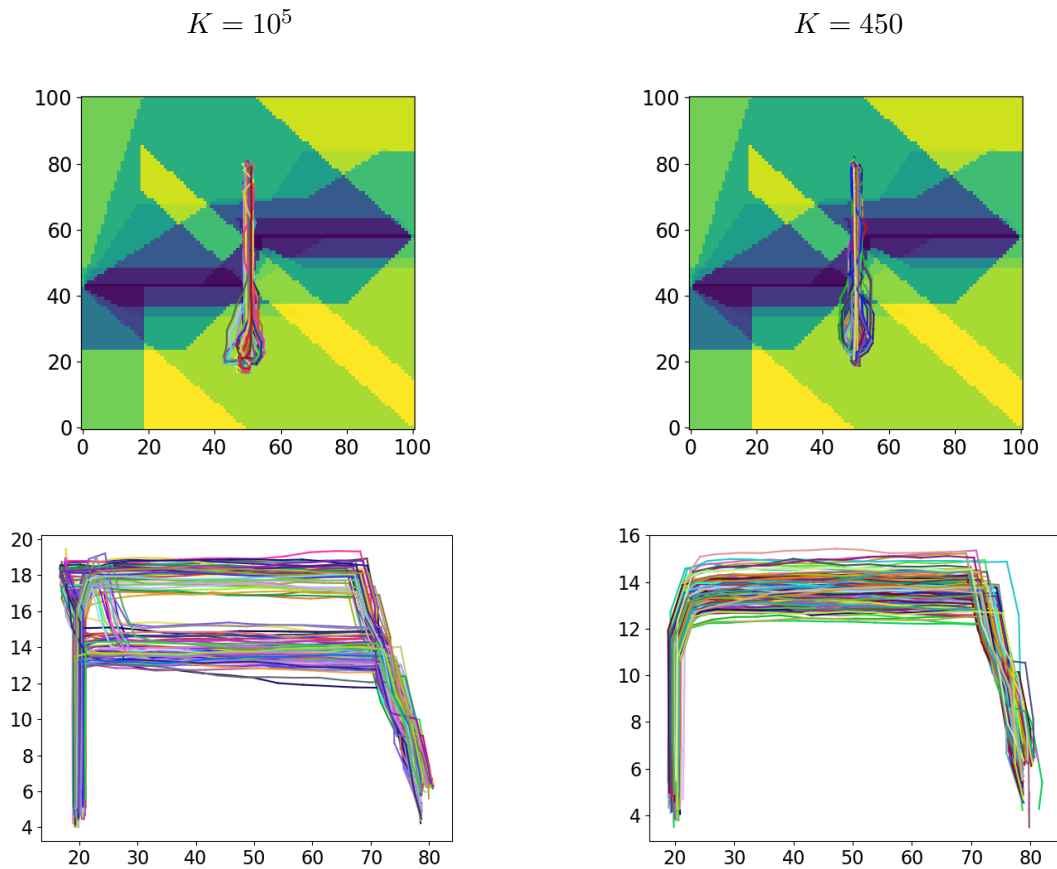
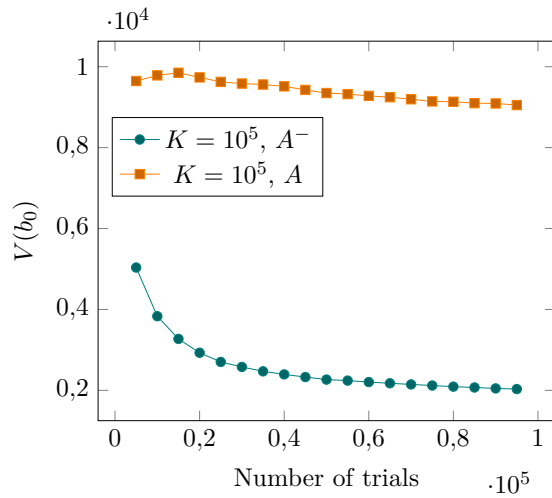


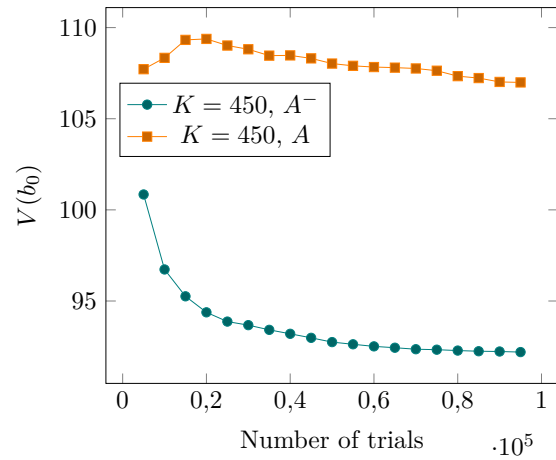
FIGURE 6.5 – Exemples des chemins simulés à partir des différentes politiques.

du benchmark, qui sont assez similaires en termes de distance de la mission et de complexité de l'environnement (à l'exception de San Diego). Il faut noter que l'ensemble réduit d'actions  $A^-$  est un sous-ensemble de l'ensemble complet d'actions  $A$ . De ce fait, des politiques calculées avec  $A^-$  sont aussi des politiques applicables avec  $A$ . La politique la plus sûre  $\pi_s$  résultant de l'optimisation avec  $A^-$  peut être aussi utilisée pour le problème de planification avec  $A$ .

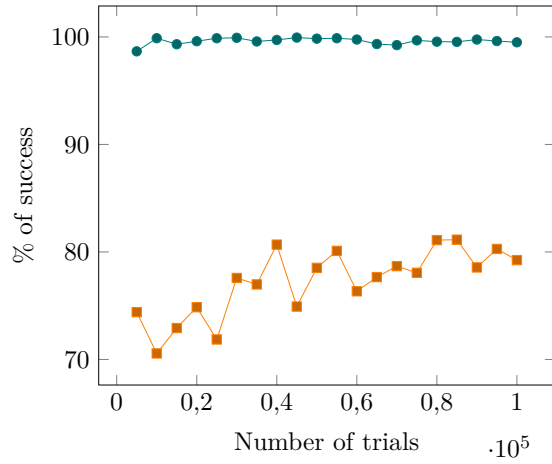
Dans la suite, on vérifie pour les deux autres cartes (*CubeBaffle* et San Diego) que l'on peut obtenir la politique la plus sûre  $\pi_s$  avec  $K = 450$ .



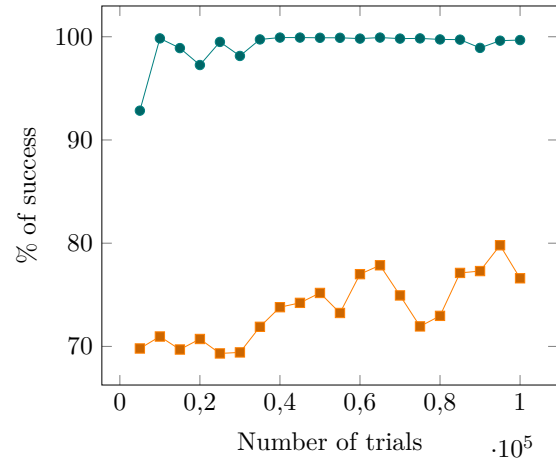
(a) Comparaison de l'évolution de la fonction de valeur pour  $K = 10^5$ .



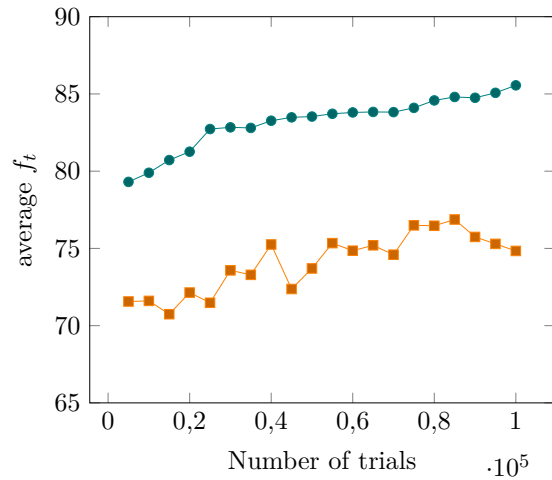
(b) Comparaison de l'évolution de la fonction de valeur pour  $K = 450$ .



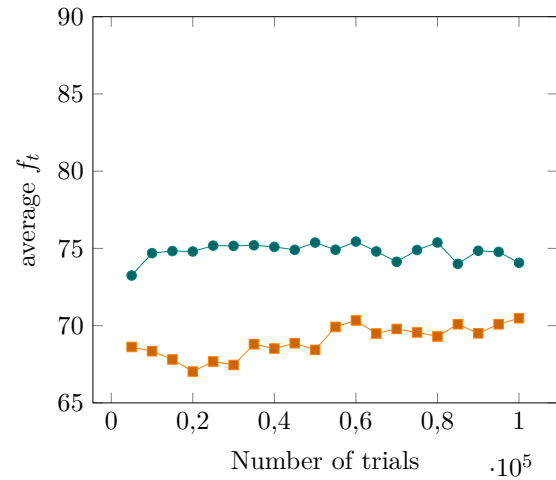
(c) Comparaison de l'évolution du taux de succès pour  $K = 10^5$ .



(d) Comparaison de l'évolution du taux de succès pour  $K = 450$ .



(e) Comparaison de l'évolution du temps de vol pour  $K = 10^5$ .



(f) Comparaison de l'évolution du temps de vol pour  $K = 450$ .

FIGURE 6.6 – Comparaison des différents résultats en fonction de l'ensemble d'actions utilisé et de la pénalité de collision.

### 6.2.2.2 *CubeBaffle*, $K = 450$ et les ensembles d'actions $A^-$ et $A$

Quand nous regardons l'évolution de la valeur de l'état de croyance initiale (Figure 6.7), nous constatons que pour l'ensemble réduit d'action la valeur de  $b_0$  pour chaque politique tends vers la même valeur (environ 93). Les valeurs de  $b_0$  lors des simulations ont convergé dès les premiers *trials* à environ 75. La différence entre les valeurs pendant les optimisations et les simulations s'expliquent toujours par le biais de l'algorithme. En revanche et à l'instar du problème sur la carte *WallBaffle*, les politiques n'ont pas convergé avec l'ensemble d'actions  $A$  lors des optimisations ce qui est confirmé par les valeurs lors des simulations qui sont plus élevées que celles lors de l'optimisation. Les politiques à la fin des optimisations sont partielles, c'est-à-dire qu'elles contiennent des états de croyances inexplorés. Or lors des simulations quand un état de croyance, qui n'est pas dans la politique, est rencontré nous appliquons la politique heuristique pour finir les simulations. Cette politique étant optimiste (ne considérant pas le risque de collision), elle peut avoir une valeur réelle supérieure à celle approximée par la valeur heuristique admissible. .

Avec l'ensemble réduit d'actions  $A^-$  le taux de succès est en moyenne de 99,9% avec un temps de vol moyen de 75 en moyenne (Figure 6.8). On peut considérer cette politique comme la politique la plus sûre  $\pi_s$  dans cette configuration. Pour l'ensemble complet d'actions  $A$ , nous constatons que le taux de succès est de 80% en moyenne avec un temps de vol moyen de 73 à cause de l'augmentation de la dimension de l'espace de recherche, les politiques obtenues après  $10^5$  *trials* restent encore assez proche et guidées par la politique heuristique.

En analysant les chemins simulés pour l'ensemble d'actions réduit, le drone évite les obstacles en passant au-dessus pour redescendre ensuite. Prendre de la hauteur lui permet d'avoir une plus grande probabilité de disponibilité du GPS et ainsi réduire son incertitude. Contourner les obstacles ne permet pas d'éviter les zones où le GPS a une probabilité faible d'être disponible (Figure 6.9) et le temps de vol pourrait être plus long. En revanche, pour l'ensemble complet  $A$  nous voyons des chemins commencer à monter puis suivant les observations reçues ils vont soit continuer jusqu'au but, soit descendre pour voler entre les deux obstacles. Cette politique semble être une politique intermédiaire en cours d'optimisation qui part de la politique heuristique vers la politique la plus sûre qui survole les obstacles. La première action est bien optimisée et l'action « monter » est choisie. Par contre, plus loin en profondeur, l'optimisation n'est pas encore finie et est encore guidée par l'heuristique (qui amène le chemin vers le bas pour passer entre les deux obstacles).

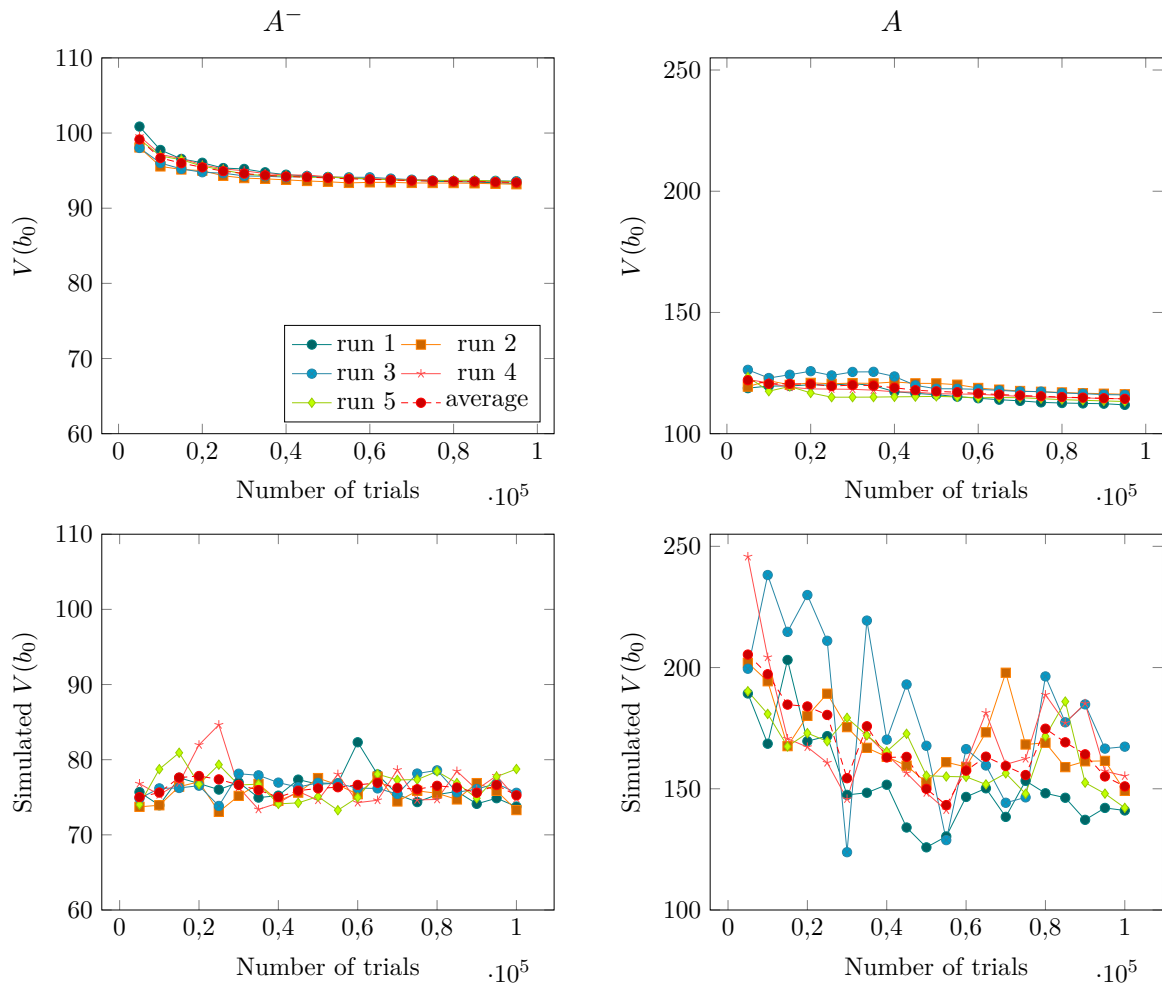


FIGURE 6.7 – Évolution de la fonction de valeur de la planification sur la carte *CubeBaffle* pour les deux ensembles d’actions, une pénalité de collision  $K = 450$  et le coefficient d’exploration  $c = 0.222 \times K$ .

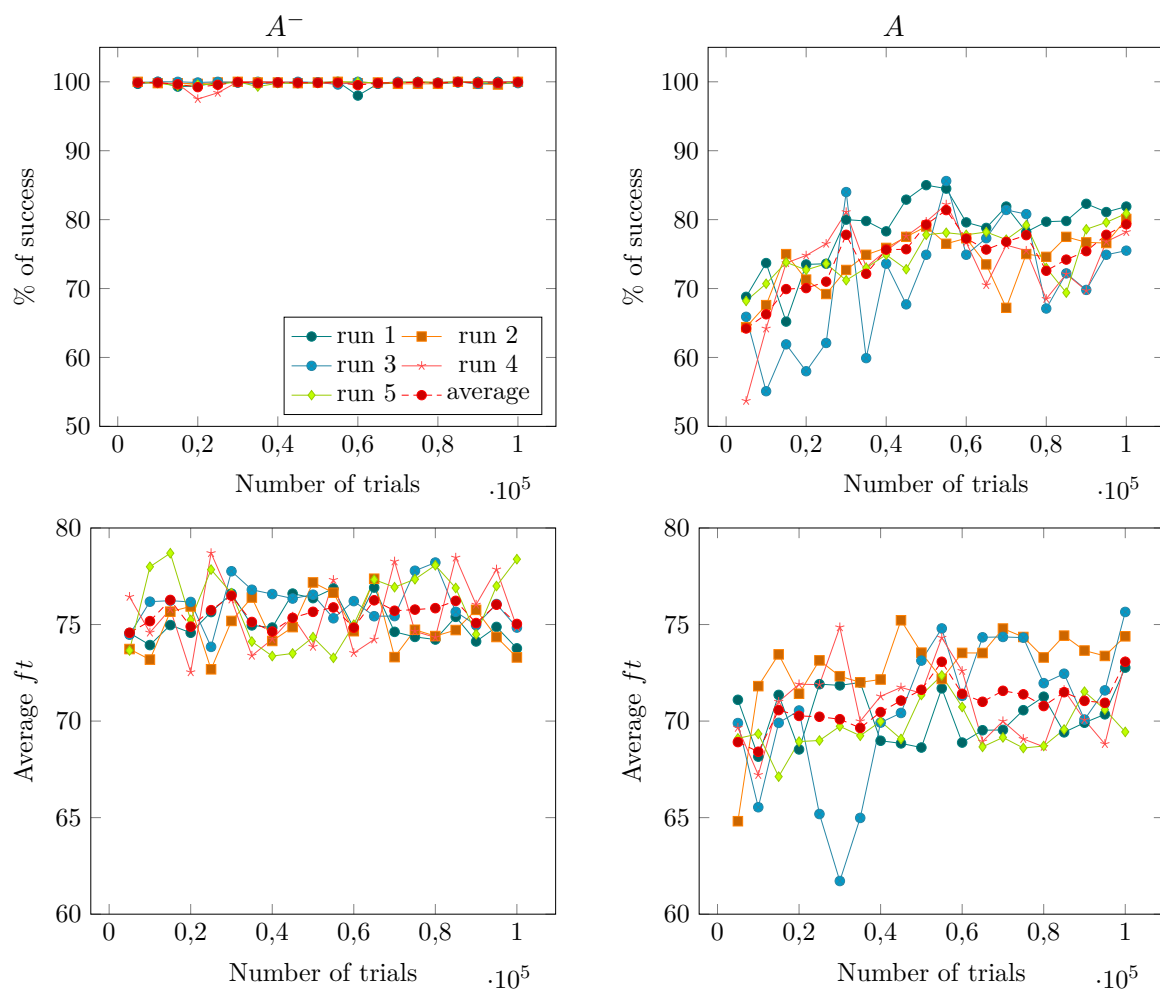


FIGURE 6.8 – Résultats de la planification sur la carte *CubeBaffle* pour les deux ensembles d'actions, une pénalité de collision  $K = 450$  et le coefficient d'exploration  $c = 0.222 \times K$ .



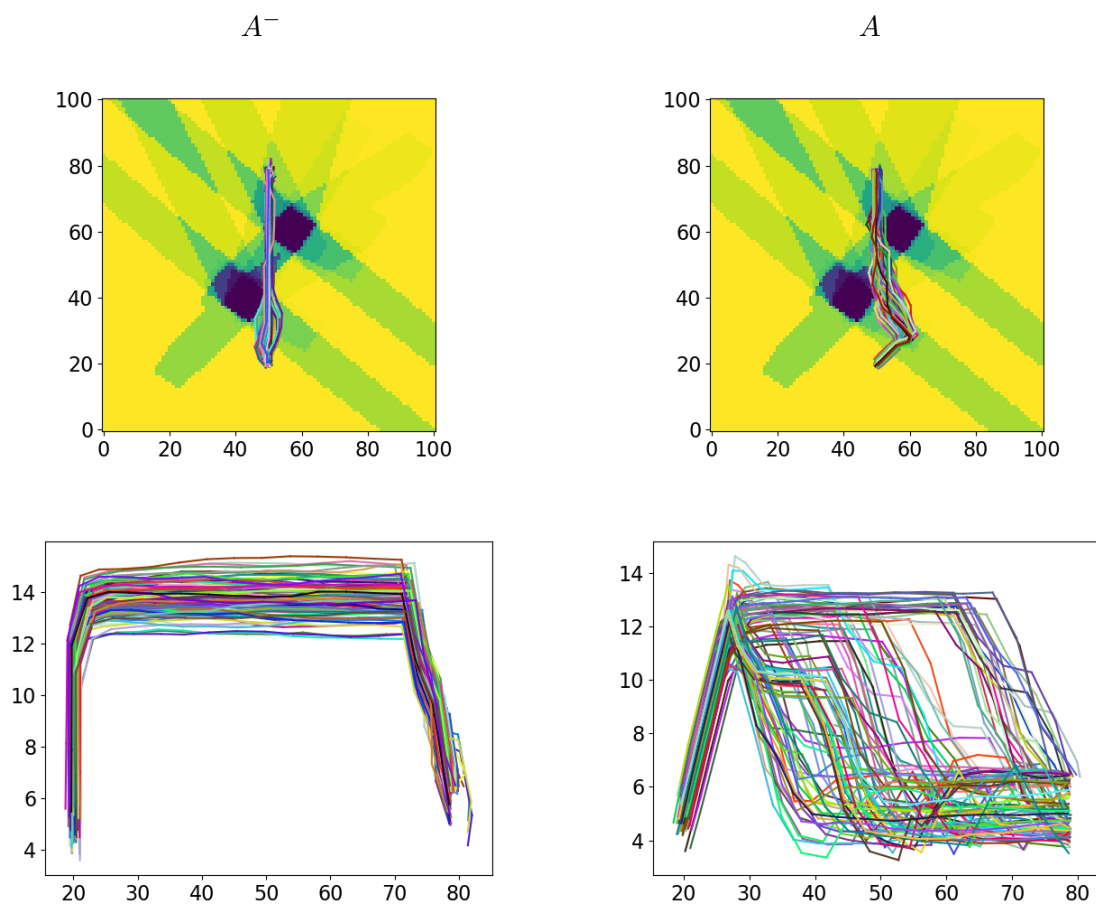


FIGURE 6.9 – Exemples des chemins simulés à partir des différentes politiques.

### 6.2.2.3 San Diego, $K = 450$ et les ensembles d'actions $A^-$ et $A$

Nous avons vu que le modèle permettait de calculer des politiques proches de la politique « la plus sûre » pour les cartes *WallBaffle* et *CubeBaffle* pour l'ensemble réduit d'action et avec une pénalité de collision  $K = 450$ .

Néanmoins, il est important de vérifier que le modèle puisse être utilisé sur des cartes plus complexes et notamment pour des environnements plus réalistes. Nous avons testé notre modèle sur la carte de San Diego mise à disposition par la plate-forme de test (Figure 4.9). La carte de probabilité sur la disponibilité du GPS montre que la disponibilité varie beaucoup près des bâtiments et même dans certaines zones pourtant assez éloignées à cause de la hauteur des bâtiments.

Nous avons testé notre planificateur sur la carte de San Diego avec une pénalité de collision  $K = 450$  pour l'ensemble réduit des actions et l'ensemble complet des actions (Figure 6.10).

L'évolution de la valeur de l'état initial pour l'ensemble réduit d'actions montre que deux politiques (run 3 et 4) semblent avoir convergé parmi les cinq. Quand nous regardons les valeurs de  $b_0$  lors des simulations, seules ces deux politiques ont une valeur inférieure, ce qui confirme leur convergence. Tandis que pour l'ensemble complet des actions  $A$ , quelle que soit la politique calculée aucune d'entre elles n'a convergé, ce qui est encore dû à l'augmentation de la taille du problème. En effet, pour l'ensemble réduit d'actions  $A^-$ , la meilleure politique (run 4) qui a convergé en valeur a un taux de succès de 100% et un temps de vol moyen de 105 secondes. Elle peut donc être considérée comme la politique la plus sûre  $\pi_s$  de ce problème.

Dans un problème avec un environnement plus réaliste, nous constatons la difficulté de l'algorithme à calculer des politiques sans risque de collision en  $10^5$  *trials* même pour l'ensemble réduit d'actions. Dans ce problème, la disponibilité du GPS est plus incertaine, ce qui augmente le nombre d'observations possibles lors de l'optimisation de la politique et donc la complexité de résolution. De plus la taille des cartes est plus grande. Il est demandé à l'algorithme d'explorer plus pour trouver un chemin qui contourne des obstacles.

Les chemins simulés des politiques minimisant la valeur pour chaque ensemble d'actions (run 4 pour  $A^-$  et run 1 pour  $A$ , Figure 6.11) passent au-dessus des bâtiments. Cela permet d'augmenter la disponibilité probable d'avoir le GPS (Figure 4.9), comme dans les exemples avec *WallBaffle* et *CubeBaffle*.

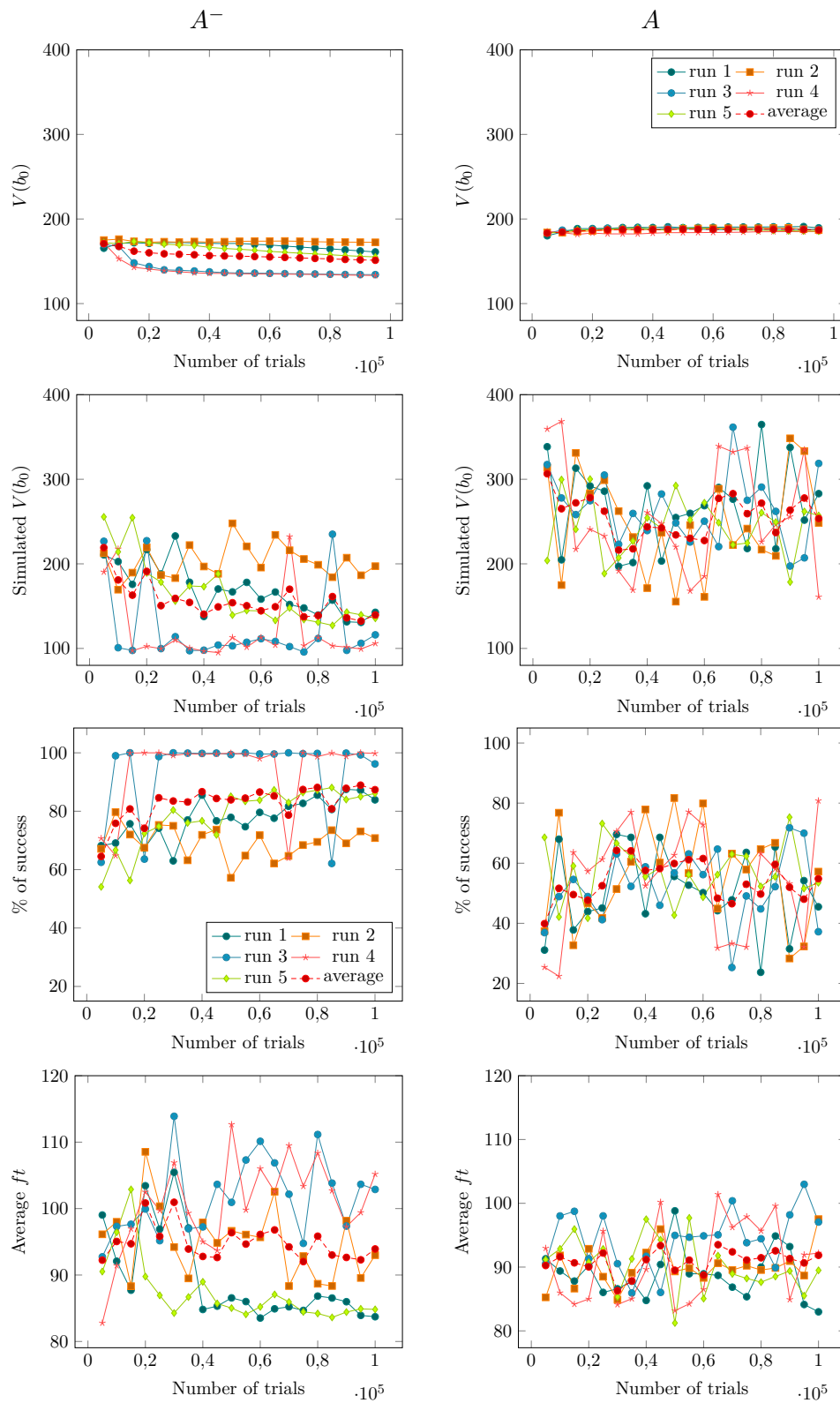


FIGURE 6.10 – Résultats de la planification sur la carte de San Diego pour  $A^-$  et  $A$ .

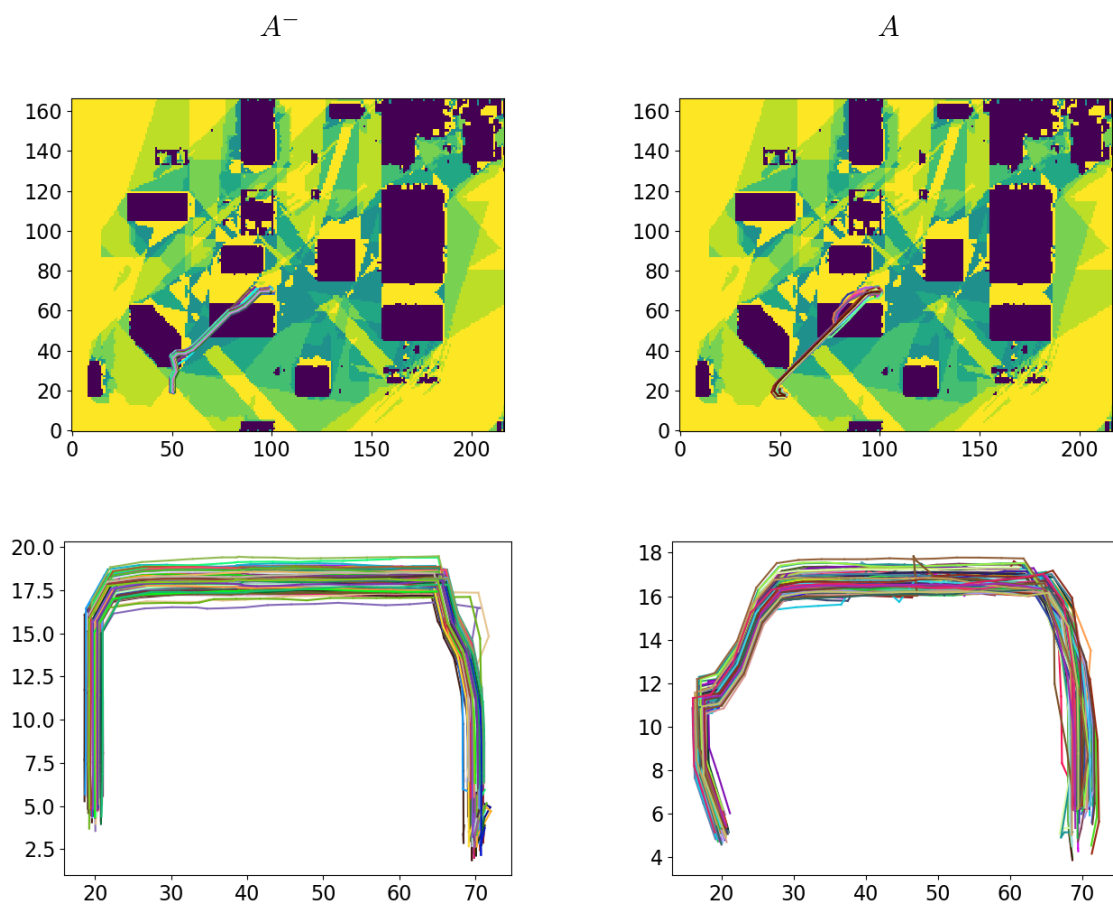


FIGURE 6.11 – Exemples des chemins simulés à partir des différentes politiques.

### 6.2.3 Planification de chemins avec un taux de collision maximum acceptable donné

Nous avons vu précédemment dans la section 6.1 qu'il était possible de déterminer quelle valeur doit prendre la pénalité de collision  $K$  afin que la politique optimale respecte un taux de collision maximum autorisé  $p_{thd}$  donné par la mission.

Afin de pouvoir calculer la valeur de  $K$  à partir de  $p_{thd}$  par (Éq. 6.5), il est nécessaire de connaître la politique la plus sûre (supposée existante) afin d'obtenir le temps de vol du chemin minimum sans risque de collision ( $T_{max}$ ). C'est pourquoi dans la section 6.2.2.1, nous avons effectué les planifications avec une valeur de  $K$  suffisamment grande pour obtenir ce  $T_{max}$  pour chaque carte. Nous considérons  $T_h$  le temps de vol espéré issue de la politique heuristique.

$$K(T_{max}, T_h, p_{thd}) = T_h + \frac{(T_{max} - T_h)}{p_{thd}} \quad (6.6)$$

#### 6.2.3.1 *WallBaffle* , l'ensemble réduit d'actions $A^-$ , $p_{thd} = 0.10$ et $p_{thd} = 0.40$

Nous avons repris le problème d'optimisation sur la carte *WallBaffle* et une pénalité de collision  $K = 450$  (Section 6.2.2.1). Nous avons alors pris le temps de vol moyen  $T_{max} = 75s$  d'une politique calculée qui avait un taux de succès de 100% (Section 6.2.2.1, Figure 6.4) et avec  $T_h = 61$  (connu a priori par l'heuristique utilisée). Nous utilisons les deux taux de collision  $p_{thd} = 0.10$  et  $p_{thd} = 0.40$ , pour lesquels la pénalité de collision  $K$  est calculée de la façon suivante :

$$K(T_{max} = 75, T_h = 61, p_{thd} = 0.10) = T_h + \frac{(T_{max} - T_h)}{p_{thd}} = 61 + \frac{(75 - 61)}{0.10} = 201 \quad (6.7)$$

$$K(T_{max} = 75, T_h = 61, p_{thd} = 0.40) = T_h + \frac{(T_{max} - T_h)}{p_{thd}} = 61 + \frac{(75 - 61)}{0.40} = 96 \quad (6.8)$$

La figure 6.12 compare les résultats des politiques pour les deux taux de collisions et les résultats de la politique la plus sûre déjà présentés (Section 6.2.2.1).

Nous pouvons constater pour  $p_{thd} = 0.10$  que la valeur de l'état de croyance initiale  $b_0$  lors de l'optimisation tend en moyenne vers 90 à la fin des  $10^5$  trials. Lorsque nous regardons l'évolution des valeurs de  $b_0$  lors des simulations (qui ne sont pas biaisées), nous constatons qu'à l'exception d'une politique (run 1) elles ont toutes convergé vers 74 ce qui est inférieur à  $T_{max}$ . De même pour  $p_{thd} = 0.40$ , nous constatons aussi que les valeurs de  $b_0$  lors des simulations sont inférieur à  $T_{max}$ . Nous pouvons donc garantir que les politiques respectent le taux de collision maximal autorisé. Il serait nécessaire de pouvoir le garantir avec la valeur

calculée lors de l'optimisation, mais celle-ci étant biaisé nous pouvons uniquement vérifier cela avec la valeur non biaisé issue des simulations.

Effectivement, quand nous regardons l'évolution du taux de succès (Figure 6.12, 3e ligne) pour les politiques finales avec  $p_{thd} = 0.10$ , les politiques (à l'exception du run 1) ont un taux de succès d'environ 99.9% qui respecte le taux minimum acceptable 90%. Ces politiques diminuent un peu le temps de vol par rapport à  $T_{max}$  (73s contre 75s). C'est un effet attendu de la diminution de la pénalité de collision  $K$ , qui augmente la priorité de l'efficacité du chemin par rapport à la sécurité.

Les politiques calculées au bout de  $10^5$  *trials* avec un taux de collision maximum autorisé de 40% ont un taux de succès moyen de 64% et un temps de vol moyen de 62s. Nous voyons que les politiques respectent bien le taux de collision autorisé, et comme prévu elles ont un temps de vol moyen beaucoup plus faible, environs 10s de moins que la politique la plus sûre. De plus quand nous regardons les chemins simulés d'une des politiques finales, nous voyons qu'ils passent entre les deux murs sans les survoler en prenant le risque d'entrer en collision avec les obstacles.

Les résultats des politiques calculées avec les deux taux de collisions maximum autorisés montrent bien l'intérêt de la méthode proposée dans ce chapitre. En partant de la politique dite « la plus sûre », nous avons pu définir des taux de collision maximum acceptable afin d'autoriser l'algorithme à calculer une politique minimisant plus le temps de vol en prenant un certain risque de collision. Les résultats des tests montrent de quelle façon la politique optimale change en fonction de la pondération entre la sécurité et l'efficacité (défini par la mission) quand le planificateur tient compte de la disponibilité probable de capteurs. C'est l'objectif et l'intérêt principal des travaux de cette thèse.

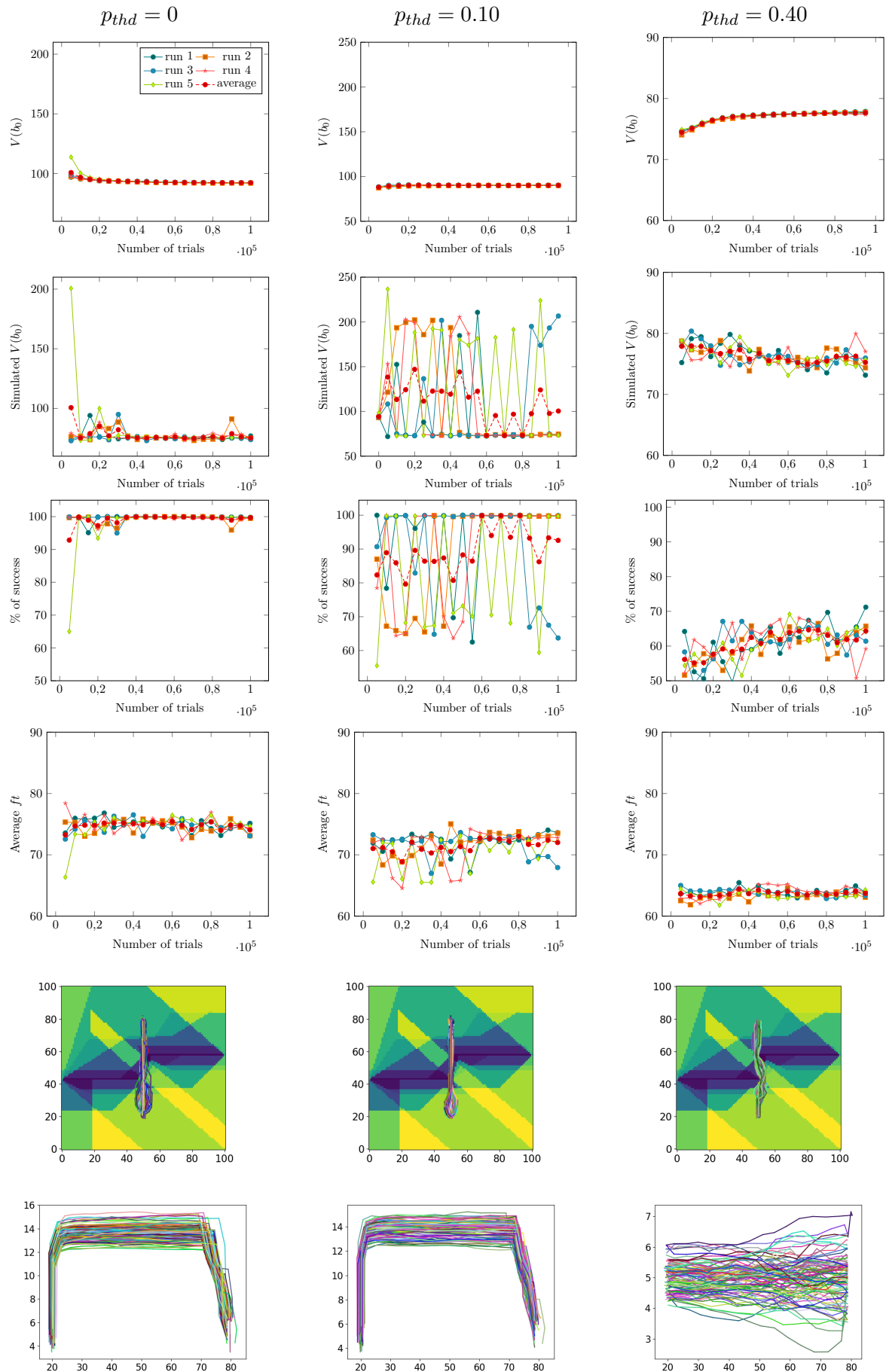


FIGURE 6.12 – Résultats de la planification sur la carte *WallBaffle* pour l'ensemble réduit des actions et différents taux de collisions.

### 6.2.3.2 San Diego, l'ensemble d'action $A^-$ et $p_{thd} = 0.40$

Nous avons ensuite testé la méthode sur la carte de San Diego pour l'ensemble réduit d'action et en autorisant un taux de collision  $p_{thd} = 0.40$ . Nous avons utilisé la politique la plus sûre calculée pour ce problème (Section 6.2.2.3) qui donne  $T_{max} = 105$  pour trouver la pénalité de collision  $K$  selon la méthode proposée dans ce chapitre :

$$\begin{aligned} K(T_{max} = 105, T_h = 82, p_{thd} = 0.40) &= T_h + 1 \frac{(T_{max} - T_h)}{p_{thd}} \\ &= 82 + \frac{(105 - 82)}{0.4} = 140 \end{aligned} \quad (6.9)$$

La figure 6.13 présente les résultats obtenus en comparant avec les résultats de la politique la plus sûre. L'évolution de la valeur de  $b_0$  pour chaque politique lors de l'optimisation semble tendre vers 106, ce qui est très proche de  $T_{max}$ . Les valeurs non biaisées sont en revanche en dessous de  $T_{max}$ , nous pouvons donc garantir que les politiques respectent le taux maximum de collision donné. Le taux de succès des politiques à la fin des  $10^5$  trials est d'environ 75% avec un temps de vol moyen d'environ 83s. Le temps de vol est bien inférieur au temps de vol de la politique la plus sûre en respectant le taux de collision maximum autorisé de 40%.

Les chemins simulés passent entre les bâtiments au lieu de les survoler afin de diminuer le temps de vol. Nous voyons que la partie entre les bâtiments, où la disponibilité du GPS est moins probable, que l'incertitude sur la localisation a un impacte sur la largeur de tous les chemins simulés qui représente l'incertitude sur la position (erreur d'exécution).

Le planificateur proposé est capable d'anticiper le risque de collision dû à cette incertitude sur la position le long du chemin, et de calculer la politique qui minimise le temps de vol moyen en respectant le taux de collision maximum autorisé.



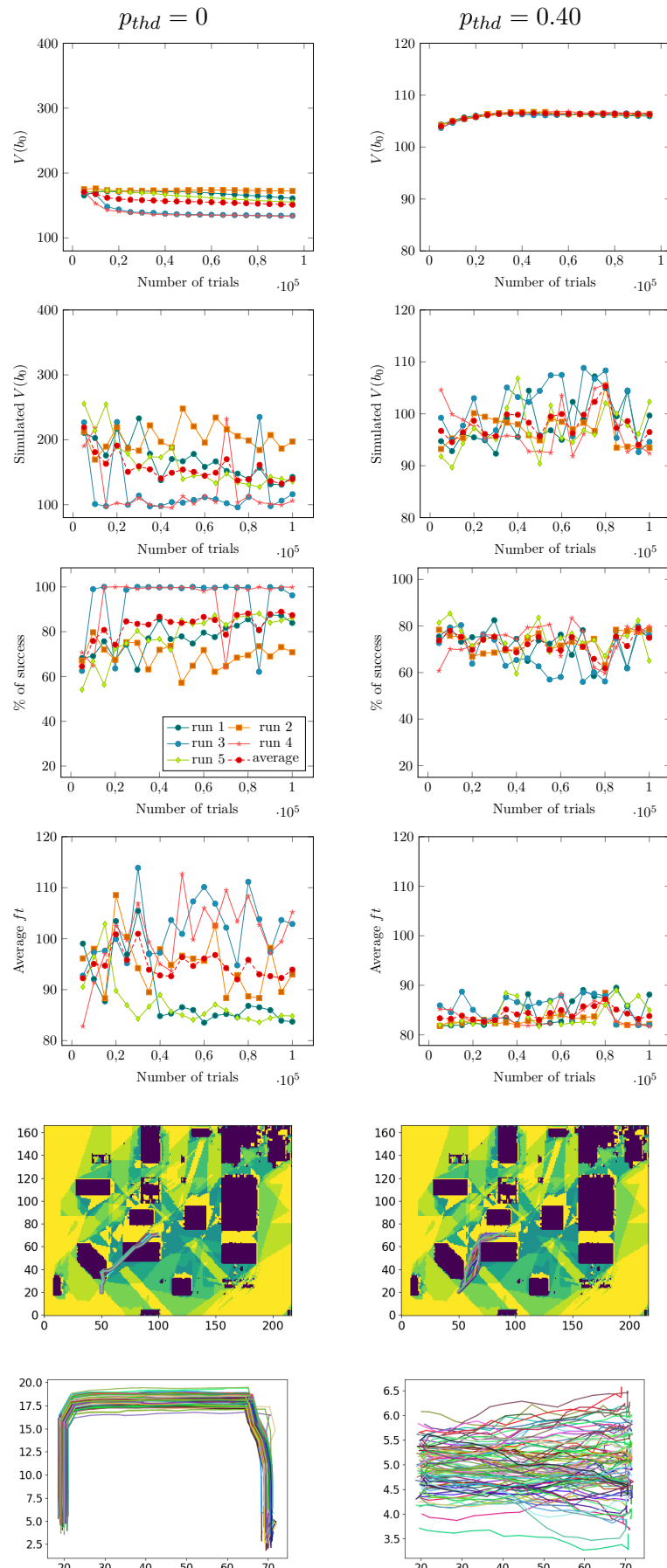


FIGURE 6.13 – Résultats de la planification sur la carte de San Diego pour les ensembles réduit des actions.

### 6.3 Bilan

Dans ce chapitre, nous avons proposé un planificateur qui minimise le temps de vol en respectant un taux de collision maximum autorisé, en tenant compte de la disponibilité probable des capteurs embarqués. Pour cela, nous avons réécrit la fonction de valeur (Éq. 6.2) comme une fonction linéaire de la pénalité de collision  $K$ , afin de déterminer sa valeur à partir du taux maximum de collision autorisé pour la mission. Pour cela, il était nécessaire de calculer la politique la plus sûre, celle qui minimise le temps de vol en n'autorisant aucun risque de collision. C'est pourquoi nous avons testé deux coûts de collision différents suffisamment grand pour déterminer lequel permettait d'obtenir cette politique. Puis nous avons appliqué ce coût de collision sur deux cartes du benchmark pour obtenir ces politiques les plus sûres.

Nous avons ensuite appliqué la méthode proposée sur la carte simple *WallBaffle* et la carte plus réaliste de San Diego. Nous avons constaté que sur la carte *WallBaffle*, si le taux de collision autorisé n'est pas assez élevé les politiques calculées restent proche de la politique la plus sûre. En revanche, en autorisant un taux de collision plus élevé il a été possible à l'algorithme de calculer un chemin plus efficace et plus risqué, mais tout en respectant le taux de collision autorisé par la mission. Pour la carte de San Diego nous avons vu que l'algorithme arrive à calculer dans le nombre de *trial* impartis des politiques respectant ce taux de collision maximum. Cela montre que la méthode fonctionne même sur des problèmes difficiles ou il y a de grandes zones d'incertitude sur la disponibilité des capteurs embarqués.

Les résultats démontrent bien l'intérêt et la capacité de l'algorithme proposé à planifier un chemin pour un véhicule autonome en prenant en compte sa capacité de navigation (capteurs embarqués) dans un environnement donné, ainsi que les critères de la mission (le niveau de sûreté et l'efficacité demandée). La méthode proposée est une méthode viable et prometteuse pour cette classe de problèmes de planification de chemins.

## Conclusion et perspectives

### Sommaire

---

<b>7.1 Conclusion</b>	<b>121</b>
7.1.1 Proposition d'un modèle MOMDP et des algorithmes de résolution avec représentation d'état de croyance par des modèles de mélange gaussiens	122
7.1.2 Évolution du modèle MOMDP et algorithme de résolution POMCP-GO	123
7.1.3 Étude du lien entre la fonction de coût et les politiques acceptables	123
<b>7.2 Perspectives</b>	<b>124</b>
7.2.1 Proposer une nouvelle heuristique pour initialiser les $Q$ -valeurs	124
7.2.2 Un coefficient d'exploration adaptatif en fonction de la complexité de l'environnement	125
7.2.3 Approfondir la méthode d'élagage de l'arbre de recherche	125
7.2.4 Comparer et regrouper des états de croyances proches	125
7.2.5 Planification en ligne	126

---

### 7.1 Conclusion

La navigation de véhicule autonome dans des milieux encombrés reste un domaine de recherche très actif. Pour naviguer dans ces environnements complexes, les véhicules dépendent des capteurs embarqués ainsi que de leurs précisions. La nature même de ces capteurs fait que leur précision et leur disponibilité d'usage dépendent de l'environnement. Par ailleurs, ce sont les environnements les plus encombrés, donc contenant le plus d'obstacles qui impactent le plus la disponibilité des capteurs et notamment le GPS.

C'est pourquoi l'objectif de cette thèse est de proposer un modèle pour la planification d'une stratégie de navigation pour des drones autonomes dans des milieux encombrés en tenant compte de la capacité de navigation d'un véhicule dans un environnement donné, ainsi que les critères de la mission (sécurité/efficacité). L'enjeu principal étant de pouvoir planifier des chemins sûrs, c'est-à-dire sans risque de collision, tout en minimisant le temps de vol. Pour cela, il est nécessaire de prendre en compte le modèle de mouvement du véhicule concerné en boucle fermé avec le module GNC, l'ensemble des capteurs embarqués ainsi

que leurs disponibilités en fonction de la position dans l'environnement. Dans la suite, nous détaillerons les contributions par chapitres.

### 7.1.1 Proposition d'un modèle MOMDP et des algorithmes de résolution avec représentation d'état de croyance par des modèles de mélange gaussiens

Dans le chapitre intitulé "Proposition d'un modèle MOMDP et des algorithmes de résolution avec représentation d'état de croyance par des modèles de mélange gaussiens", nous avons proposé un premier modèle basé sur les MOMDP. Ce modèle intègre un modèle de la dynamique du GNC (en boucle fermée) capable de gérer plusieurs modes de navigation (capteurs). De plus, ce modèle de GNC permet de modéliser la propagation de l'erreur d'exécution en tant que loi normale.

En exploitant cette particularité, il est possible de définir les états de croyances comme des lois normales, ainsi la fonction de transition du modèle de planification retournera après chaque action une loi normale représentant un nouvel état de croyance avant la prise en compte de la nouvelle observation qui renseigne sur l'ensemble des capteurs disponibles. Cependant, la prise en compte de l'observation déforme la distribution de probabilité qui ne reste plus une loi normale. Pour pallier à ce problème, nous utilisons un algorithme d'apprentissage *Expectation-Maximization* afin d'approximer la distribution de probabilité en modèle de mélange gaussien.

Cette représentation des états de croyances permet d'utiliser la notion de couloir d'incertitude afin de calculer le coût de transition entre deux états de croyances en tenant compte de l'incertitude. Lors de l'optimisation, la minimisation de ce coût permet de réduire la distance parcourue, mais aussi l'incertitude sans pénaliser l'un de ces critères par rapport à l'autre. C'est donc une fonction de coût qui ne nécessite pas de réglage particulier. De plus en minimisant le volume, nous minimisons aussi l'incertitude et donc indirectement le risque de collision.

Cependant, l'utilisation de cette fonction dans le modèle ne permet plus d'obtenir une fonction de valeur linéaire par morceaux et convexe comme dans la plupart des modèles POMDP et MOMDP. Cela restreint l'utilisation des algorithmes de résolution (exacte ou approchée). Nous nous sommes alors appuyés sur des algorithmes de résolutions, qui ne représentent pas la fonction de valeur par des  $\alpha$ -vecteurs, par exemple RTDP-bel et POMCP. Nous avons proposé deux variantes de ces algorithmes LRTDP-bel et MCTBS. Ces contributions ont donné lieu à des publications [DELAMER et al. 2017a ; DELAMER et al. 2017b].

Les résultats obtenus avec ce modèle sur des cartes simples de la plate-forme de test étaient prometteurs. En revanche, quel que soit l'algorithme utilisé, il était impossible de calculer une politique en temps raisonnable sur une carte plus complexe. Une des raisons possibles est que dans des environnements complexes, le temps de calcul des modèles de mélange gaussien devient plus important. En effet sur les cartes simples le GPS est disponible sur une très grande partie de la carte avec une probabilité élevée. Or sur une carte complexe, la disponibilité du GPS est plus incertaine ce qui augmente le nombre d'observations différentes pouvant

être constatées après une transition et donc le nombre d'états de croyances atteignables. Or la probabilité d'une observation est compliquée à calculer, car il est nécessaire de calculer la densité de probabilité de l'observation pour chaque GMM. En effet, quand les capteurs embarqués sont disponibles avec une très forte probabilité le nombre d'observations différentes est plus faible que pour des environnements complexes ou la disponibilité va beaucoup varier.

Pour toutes ces raisons, nous avons fait évoluer le modèle pour pallier le problème rencontré avec celui-ci avec des environnements plus complexes.

### 7.1.2 Évolution du modèle MOMDP et algorithme de résolution POMCP-GO

L'objectif de ce deuxième modèle était de prendre en compte les limites du modèle précédent afin de pouvoir calculer des politiques sur des cartes complexes. Nous avons estimé que le problème venait des probabilités des différentes observations qui était trop longues à calculer.

L'idée dans ce modèle est de garder la base du précédent modèle, mais de ne plus utiliser des modèles de mélange gaussien pour représenter les états de croyances. Cependant, nous avons vu dans le précédent modèle que la mise à jour classique des états de croyances pouvait être compliquée dans notre problème. Pour éviter de retomber sur ce problème, nous avons fait le choix d'utiliser des algorithmes de résolution qui n'utilisent pas de représentation directe de l'état de croyance. Nous utilisons comme fonction de transition la dynamique du modèle GNC, représentant la transition d'un état à un autre. Cependant, la fonction de coût change, nous la définissons par le temps de vol de l'action qui est constant et une pénalité en cas de collision.

Pour résoudre ce modèle, nous avons proposé l'algorithme POMCP-GO. L'utilisation de cet algorithme permet de raisonner sur des nœuds d'historiques. En effet, POMCP-GO simule des couples action-observations à partir d'un état initial choisi aléatoirement parmi l'état de croyance initiale. POMCP-GO ne garde que la valeur du nœud représentant l'état de croyance ainsi que le nombre de fois que chaque action a été exécutée dans ce nœud. Ainsi il approxime la valeur de chaque état de croyance. Nous avons comparé POMCP-GO et POMCP sur différents problèmes afin de montrer l'intérêt de ce nouvel algorithme. Les résultats ont montré que sur un problème en 2D POMCP-GO et POMCP trouvent des politiques sans risque de collision, mais quand le problème est difficile seul POMCP-GO arrive à calculer des politiques sans risque de collision en temps raisonnable. Ces résultats justifient la proposition de ce nouvel algorithme.

### 7.1.3 Étude du lien entre la fonction de coût et les politiques acceptables

Le deuxième modèle proposé permet de calculer des politiques qui planifient des chemins sans risque de collision. Nous avons montré que l'acceptabilité d'un risque de collision dans

notre modèle est liée à la pénalité de collision étant donnée la fonction de coût proposé. En ce sens, nous avons proposé une méthode permettant à partir d'une politique sans risque de collision de calculer la valeur de la pénalité de collision afin d'autoriser un taux de collision maximum acceptable.

Nous avons ensuite appliqué la méthode proposée sur le benchmark de test. Nous pouvons dire que l'algorithme POMCP-GO permet de calculer des politiques sans risque de collision. De plus, la méthode proposée pour minimiser le temps de vol en autorisant un certain niveau de risque de collision a été démontrée sur des problèmes simples, mais aussi sur des problèmes difficiles. Les résultats démontrent que la méthode proposée est une méthode viable et prometteuse pour cette classe de problèmes de planification de chemins. Ces contributions ont fait l'objet de publications [DELAMER et al. 2018; DELAMER et al. 2019].

## 7.2 Perspectives

Dans la suite, nous présenterons quelques pistes pour prolonger ces travaux. Tout d'abord, nous proposerons des pistes d'améliorations algorithmiques puis nous proposerons des pistes de travaux à plus long terme.

### 7.2.1 Proposer une nouvelle heuristique pour initialiser les $Q$ -valeurs

Nous avons vu dans le chapitre 5 que la valeur d'un nouvel état de croyances devait être initialisée dans les algorithmes proposés. Jusqu'à présent, nous avons utilisé une valeur estimée du temps de vol calculé à partir de l'algorithme de Dijkstra. Cette valeur donne une bonne estimation du temps de vol minimum, mais ne permet pas de prendre en compte le risque de collision du chemin. Or, POMCP-GO utilise beaucoup cette valeur pour guider sa recherche dans les premières itérations alors que ce que nous souhaitons c'est avant tout planifier une stratégie de navigation sûre et donc sans collision. Les résultats présentés sur la carte *WallBaffle* montrent que l'algorithme explore beaucoup le passage entre les deux murs même quand nous n'autorisons pas de collision. C'est en partie dû au fait que l'heuristique ne permet pas d'estimer ce risque. C'est pourquoi il paraît important de rechercher une nouvelle fonction heuristique qui donnerait pour un état donné une valeur basée sur le temps de vol estimé pour arriver jusqu'au but tout en incluant un risque de collision estimé. Pour cela, il serait possible d'utiliser la fonction heuristique du chapitre 4 qui utilisait l'algorithme  $A^*$  en propageant l'incertitude sur l'état estimé. Cependant, cette fonction n'est calculable que pendant l'optimisation ce qui augmenterait considérablement le temps de calcul.

### 7.2.2 Un coefficient d'exploration adaptatif en fonction de la complexité de l'environnement

Bien que nous ayons étudié divers coefficients d'exploration pour la stratégie de sélection d'action UCB1 de l'algorithme POMCP-GO (Annexe A), il nous paraît prometteur de travailler sur la création d'un coefficient basé sur la complexité de l'environnement. Actuellement, le coefficient est le même quel que soit le nœud exploré par POMCP-GO, or dans un problème de planification de chemin pour des véhicules autonomes cela ne semble pas être idéal. En effet, en fonction de la profondeur il peut être moins intéressant d'explorer beaucoup. De même quand le véhicule se trouve dans une zone sans obstacle, il y a peu de chance qu'explorer beaucoup permette de converger plus vite ou de trouver une meilleure politique. Cependant, explorer plus lorsque le véhicule est dans une zone dense en obstacles où dans des zones où la disponibilité des capteurs est plus incertaine permettrait d'améliorer les politiques calculées en anticipant mieux les obstacles ou en prenant mieux en compte la disponibilité probable des capteurs. Il serait possible en utilisant les connaissances a priori de créer un score en fonction de la complexité de l'environnement. Plus le score serait élevé plus le coefficient d'exploration serait important afin de mieux explorer dans ces zones. Parallèlement, dans les zones où le score est faible le coefficient serait faible, ce qui pourrait permettre de gagner aussi en temps de calcul en évitant de parcourir de nouveaux nœuds lorsque cela n'est pas nécessaire. Les premiers travaux de Raquel Carmo pendant son *master thesis* sur la création d'un score en fonction de la disponibilité probable des capteurs et de la position des obstacles donnent des résultats encourageants.

### 7.2.3 Approfondir la méthode d'élagage de l'arbre de recherche

Dans le chapitre 5, étant donnée la complexité du problème, nous avons vu que la taille de l'arbre de recherche associée est exponentielle en nombre d'époques. C'est pourquoi pour éviter d'être confrontés à des problèmes de mémoire nous avons décidé d'élaguer certaines branches de l'arbre de recherche pendant l'optimisation. La problématique principale était supprimer les nœuds les moins intéressants sans supprimer les nœuds qui pourraient mener à une solution optimale. Même si la méthode proposée nous a permis d'élaguer l'arbre de recherche sans supprimer de nœuds importants, il nous semble intéressant d'apporter des preuves théoriques à cette méthode. De futurs travaux pourraient être de prouver, que pour des critères donnés, certains nœuds pourraient être élagués sans affecter la recherche de la politique optimale, c'est-à-dire des nœuds qui ne seront jamais explorés de nouveau après leur élagage. Et si cela n'est pas possible, l'idée serait de borner les probabilités qu'un nœud soit de nouveau exploré par la suite en fonction des critères d'élagage donnés.

### 7.2.4 Comparer et regrouper des états de croyances proches

Dans les chapitres 4 et 5, nous avons vu que l'ensemble des états (de croyance) dit atteignable est très grand. De plus, des algorithmes comme POMCP-GO ont besoin d'explorer les

nœuds plusieurs fois pour obtenir une valeur suffisamment représentative de la valeur réelle de ce nœud. Cependant, dans un problème de planification de chemin, il est possible que pour plusieurs nœuds différents, la distribution sur la position du véhicule soit proche. En effet, si nous considérons le fonctionnement d'un algorithme comme POMCP-GO, même si les historiques d'action-observation ne sont pas identiques la distribution sur la position du véhicule peut être proche. Or, la valeur d'un nœud est une information utile pour d'autres nœuds dont la distribution est proche, car elle peut permettre d'obtenir une meilleure estimation surtout pour les nœuds les moins explorés. Être capable d'identifier les nœuds les plus proches permettraient non seulement d'obtenir une estimation plus précise de la valeur de ce nœud, mais aussi d'identifier quel historique est le plus avantageux pour arriver jusqu'à cet état de croyance. Dans un premier temps, il serait nécessaire d'utiliser ou créer une métrique pour comparer deux états de croyances. Puis dans un second temps, adapter l'algorithme POMCP-GO afin de les stocker sous forme de table de hachage. Chaque entrée de cette table représenterait des états de croyance proches (comme dans le cas de l'algorithme RTDP-bel) ainsi que les informations nécessaires comme la distribution sur les états de croyances.

### 7.2.5 Planification en ligne

Finalement, l'objectif à long terme serait d'utiliser ce modèle pour la planification en ligne en boucle fermée avec le système GNC et des mesures de capteurs. L'algorithme POMCP-GO présenté dans le chapitre 5 est basé sur POMCP qui est conçu pour la planification en ligne. Il y aurait peu d'adaptations algorithmiques à effectuer pour utiliser POMCP-GO en ligne cependant la puissance de calcul des systèmes embarqués sur les drones étant plus faible la question d'un calcul partiel hors ligne se pose afin d'obtenir une première politique qui serait améliorée en ligne, le but étant de pouvoir garantir un certain taux de succès avant le lancement de la mission. Pour appliquer cette planification en ligne sur un véhicule autonome, il est nécessaire que l'algorithme retourne en un temps suffisamment court (4 secondes ou moins) une action proche de l'action optimale. De plus, avec ou sans politique partielle, il nous paraît difficile de pouvoir utiliser les travaux présentés dans le chapitre 6 sur l'utilisation d'un taux de collision acceptable qui nécessite de connaître la politique la plus sûre pour ensuite optimiser une politique respectant ce critère. Néanmoins, les travaux préliminaires sur la planification en ligne effectuée par Raquel Carmo pendant son *master thesis* sont encourageants pour de futures recherches.





# Étude de l'impact du coefficient d'exploration sur la convergence de la valeur

Dans les algorithmes comme POMCP dont le choix de l'action à explorer pendant l'optimisation est basé sur UCB1 [AUER et al. 2002], le coefficient d'exploration  $c$  est utilisé pour forcer l'algorithme à essayer des actions qui semblent a priori moins intéressantes. Cependant, le choix du bon coefficient peut accélérer cette convergence. C'est pourquoi étant donné la complexité du problème à résoudre, il est important de trouver un coefficient permettant d'accélérer au mieux la convergence.

Nous avons testé trois coefficients d'exploration sur la carte *WallBaffle* pour l'ensemble d'actions réduit  $A^-$  et une pénalité de collision  $K = 450$ . Les deux premiers coefficients ont été  $c = 5$  et  $c = 100$  qui représente respectivement 1,1% et 22,2% de la pénalité de collision. Nous avons aussi testé un coefficient variable en fonction de la profondeur dans l'arbre de recherche (c'est-à-dire l'époque du planificateur). Le coefficient variable est alors calculé de la façon suivante :

$$c_{\text{var}} = (K - t \times f_t) \frac{c}{t} \quad (\text{A.1})$$

Le coefficient variable est paramétré de telle sorte qu'à l'époque  $t = 1$  il soit égal à 100 afin de pouvoir comparer avec les deux autres coefficients. La figure A.1 montre son évolution en fonction de l'époque. Cependant, dans des problèmes de planifications de chemins il est souvent plus intéressant d'explorer au début ou près des obstacles qu'à la fin du chemin quand le drone est proche du but. En effet, c'est durant les premières époques qu'il est intéressant d'explorer des chemins alternatifs, qui anticipent les collisions probables. De même en explorant beaucoup près des obstacles, il est plus probable de pouvoir éviter les collisions. C'est pourquoi en utilisant un coefficient variable qui décroît au fur et à mesure des époques nous espérons favoriser la convergence en explorant beaucoup les actions pour les premiers états de croyances afin d'anticiper les obstacles et trouver des chemins potentiellement plus intéressants.

Les figures A.2 et A.3 montrent les résultats des politiques calculées pour les coefficients

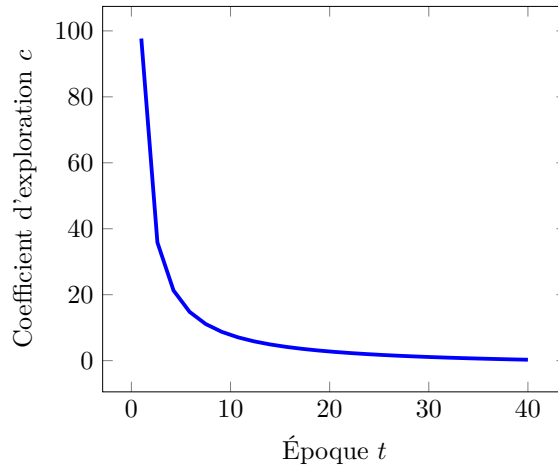
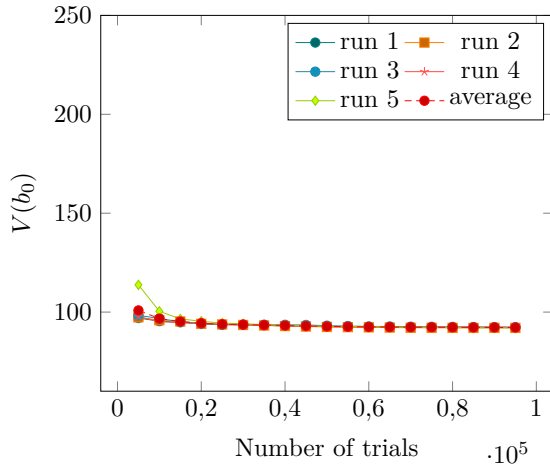
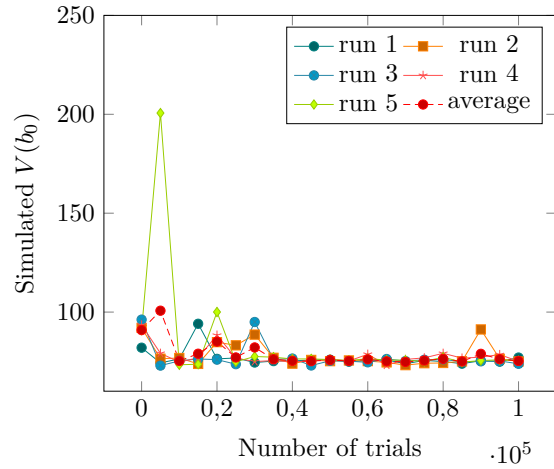


FIGURE A.1 – Évolution du coefficient variable en fonction de l'époque

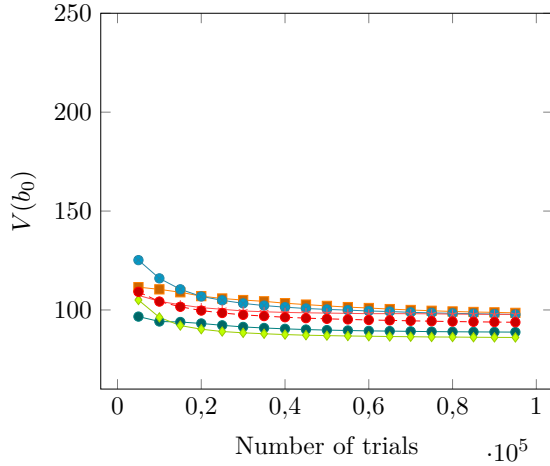
$c = 100$ ,  $c = c_{\text{var}}$  et  $c = 5$  pour l'ensemble réduit des actions et une pénalité de collision  $K = 450$ . Quand, on compare l'évolution de la valeur de l'état de croyance initiale  $b_0$  pour ces trois coefficients (Figures A.2a, A.2c et A.2e), nous constatons que les coefficients  $c = 5$  et  $c = c_{\text{var}}$  ne convergent pas tous vers des politiques donnant une même valeur. Alors que dans le cas où  $c = 100$  la valeur de  $b_0$  converge très rapidement pour toutes les politiques calculées. Cependant, la valeur moyenne pour tous les coefficients reste proche. Ces observations semblent cohérentes avec l'évolution de la valeur de  $b_0$  lors des simulations. L'évolution du taux de succès représentés par les figures A.3a, A.3c et A.3e est cohérente avec l'évolution de la valeur de l'état de croyance initiale. Pour le coefficient  $c = 100$ , nous voyons que le taux de succès atteint dès les premiers *trials* les 100% pour toutes les politiques. Tandis que pour le coefficient  $c = c_{\text{var}}$  les politiques approchent les 100% seulement vers la fin de l'optimisation. Les politiques qui avaient donné une valeur plus faible à  $b_0$  ont logiquement le meilleur taux de succès et ceci le plus rapidement. Pour le coefficient  $c = 5$  les politiques avec une valeur de l'état de croyance initial plus faible ont un taux de succès de 100%. Tandis que les autres ont un taux de succès plus faible aux alentours de 95%. L'évolution du temps de vol pour  $c = 100$  (Figure A.3b) est en moyenne de 75s. C'est cohérent avec le fait que les politiques ont toute convergé vers la même valeur pour l'état de croyance initiale. Pour les deux autres coefficients (Figures A.3d et A.3d), les temps de vols sont comme pour les valeurs de  $b_0$  assez disparates, aux alentours de 88s pour  $c = 5$  et 80s pour  $c = c_{\text{var}}$ .



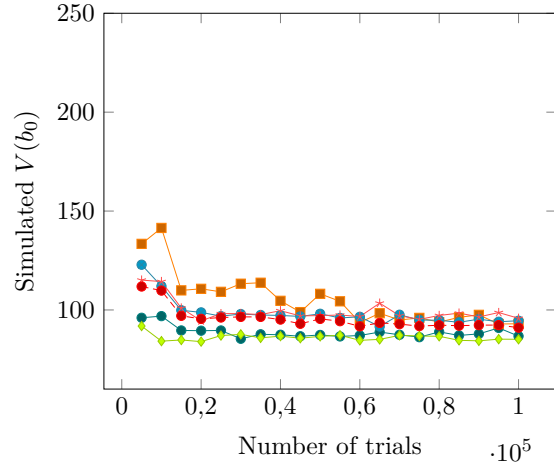
(a) Évolution de la fonction de valeur de  $b_0$  pour  $c = 100$ .



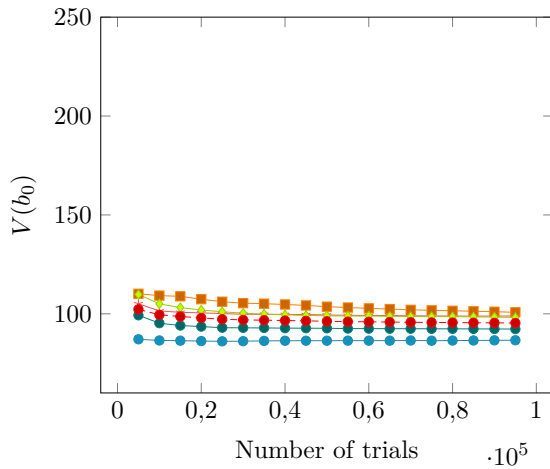
(b) Évolution de la fonction de valeur simulée de  $b_0$  pour  $c = 100$ .



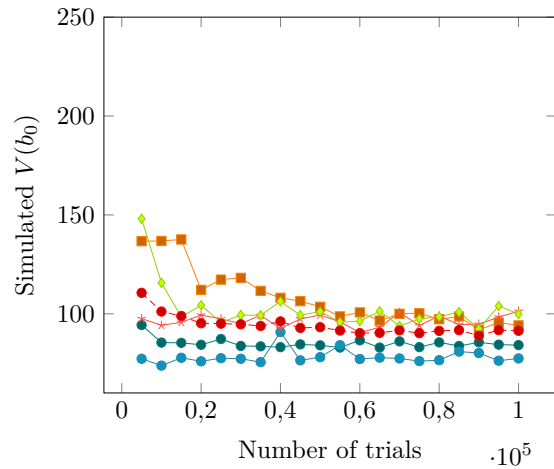
(c) Évolution de la fonction de valeur de  $b_0$  pour  $c = c_{\text{var}}$ .



(d) Évolution de la fonction de valeur simulée de  $b_0$  pour  $c = c_{\text{var}}$ .

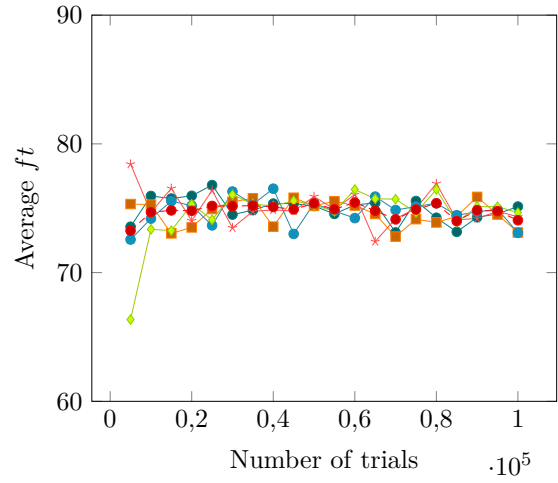
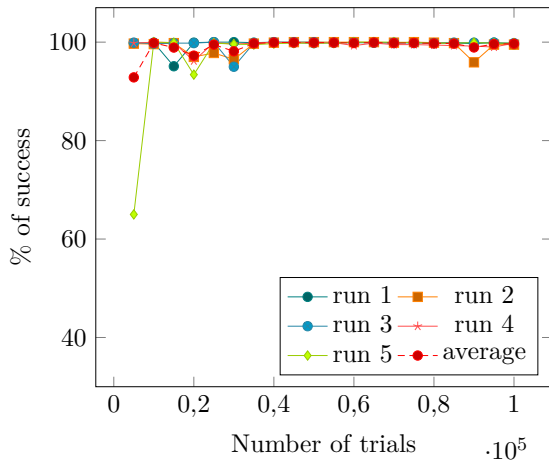


(e) Évolution de la fonction de valeur de  $b_0$  pour  $c = 5$ .



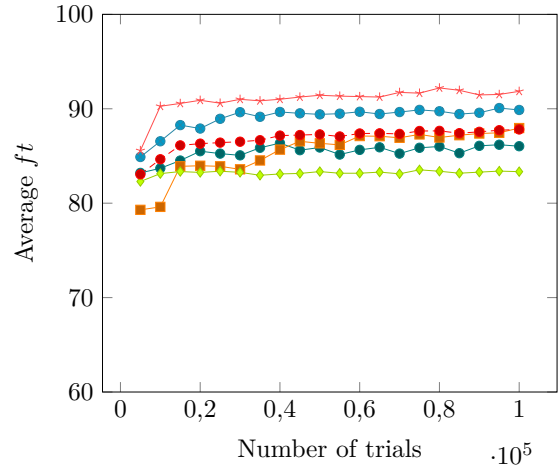
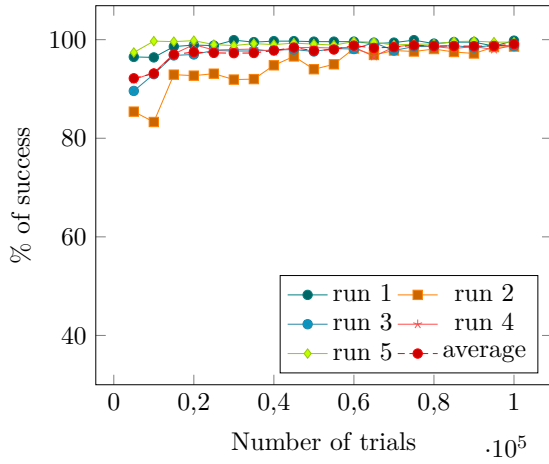
(f) Évolution de la fonction de valeur simulée de  $b_0$  pour  $c = 5$ .

FIGURE A.2 – Évolution de la fonction de valeur lors de la planification sur la carte *WallBaffle* pour l'ensemble réduit des actions et pour les coefficients  $c = 100$ ,  $c = 5$  et  $c = c_{\text{var}}$  avec  $K = 450$ .



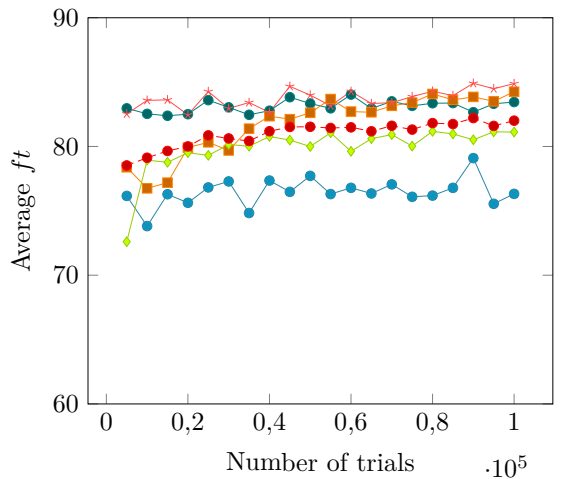
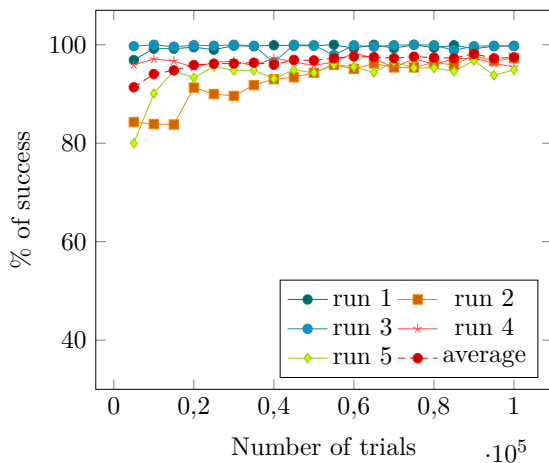
(a) Évolution du taux de succès pour  $c = 100$ .

(b) Évolution du temps de vol pour  $c = 100$ .



(c) Évolution du taux de succès pour  $c = c_{var}$ .

(d) Évolution du temps de vol pour  $c = c_{var}$ .



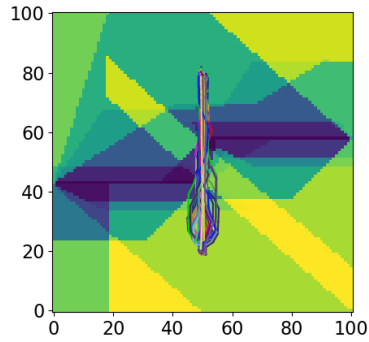
(e) Évolution du taux de succès pour  $c = 5$ .

(f) Évolution du temps de vol pour  $c = 5$

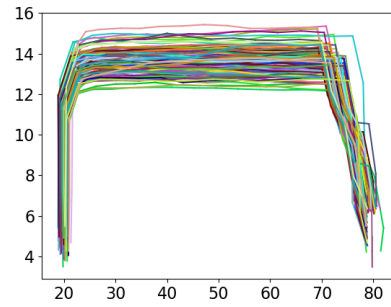
FIGURE A.3 – Résultats de la planification sur la carte *WallBaffle* pour l'ensemble réduit des actions et pour les coefficients  $c = 5$  et  $c = c_{var}$  avec  $K = 450$ .

Les chemins simulés pour chaque coefficient (Figure A.4), nous pouvons constater que les chemins passent par-dessus les deux murs et en ligne droite. Seul le coefficient  $c = 5$  montre une direction vers l'Est au début ainsi que quelques chemins qui descendent plus tôt. Ceci peut être dû au fait que comme le coefficient d'exploration est assez faible l'algorithme aura moins tendance à explorer quand il a trouvé un chemin dans l'arbre qui donne un taux de collision très faible et un temps de vol court.

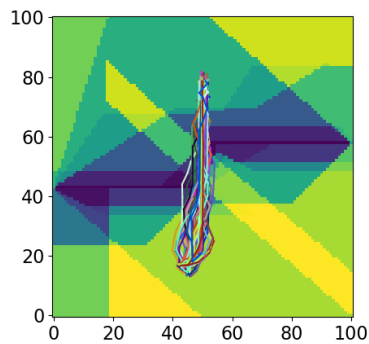
En nous basant sur les résultats présentés ainsi que la figure A.5 qui présente les résultats moyens pour les différents paramètres testés, nous pouvons dire que le coefficient d'exploration a un impacte. En effet, le coefficient d'exploration  $c = 100$  converge en valeur plus rapidement et vers une valeur plus faible que les autres coefficients. De même pour le taux de succès et le temps de vol moyen, nous constatons que le coefficient  $c = 100$  donne en moyenne un taux de succès plus élevé ainsi qu'un temps de vol moyen plus faible. Nous pouvons donc dire que le coefficient qui semble le plus adapté parmi ceux testés est le coefficient  $c = 100$ .



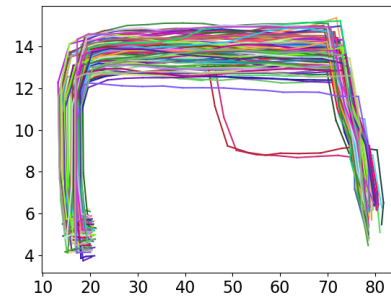
(a) Chemins simulé avec l'ensemble réduit des actions et  $c = 100$ .



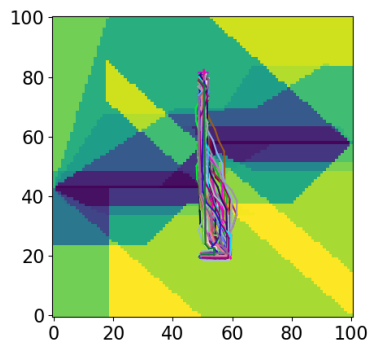
(b) Chemins simulé avec l'ensemble réduit des actions et  $c = 100$  (axe z).



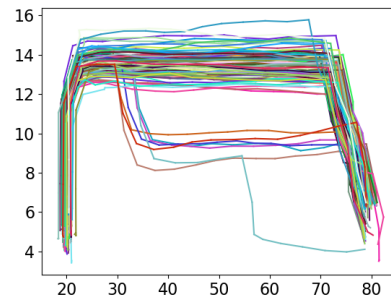
(c) Chemins simulé avec l'ensemble réduit des actions et  $c = c_{\text{var}}$ .



(d) Chemins simulé avec l'ensemble réduit des actions et  $c = c_{\text{var}}$  (axe z).

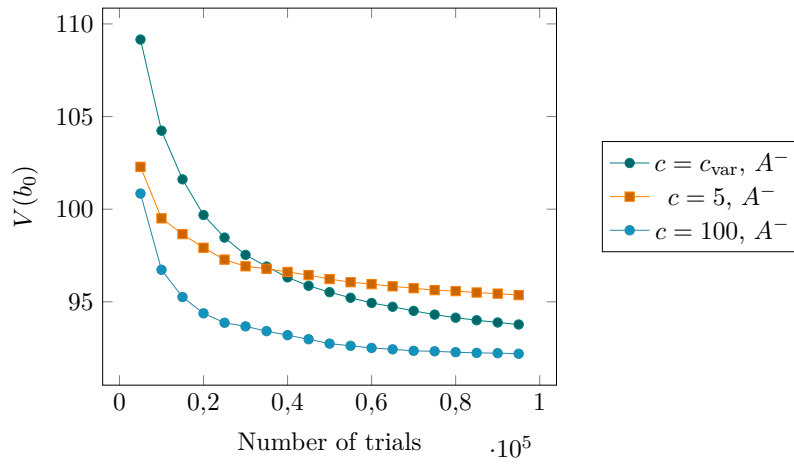


(e) Chemins simulé avec l'ensemble réduit des actions et  $c = 5$ .

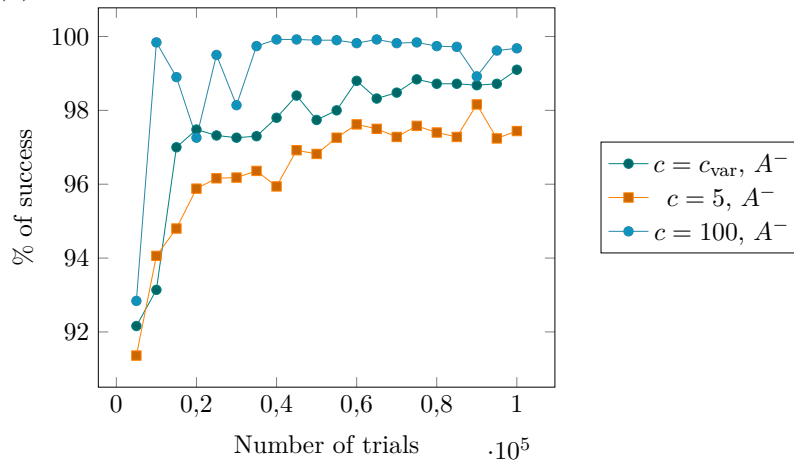


(f) Chemins simulé avec l'ensemble réduit des actions et  $c = 5$  (axe z).

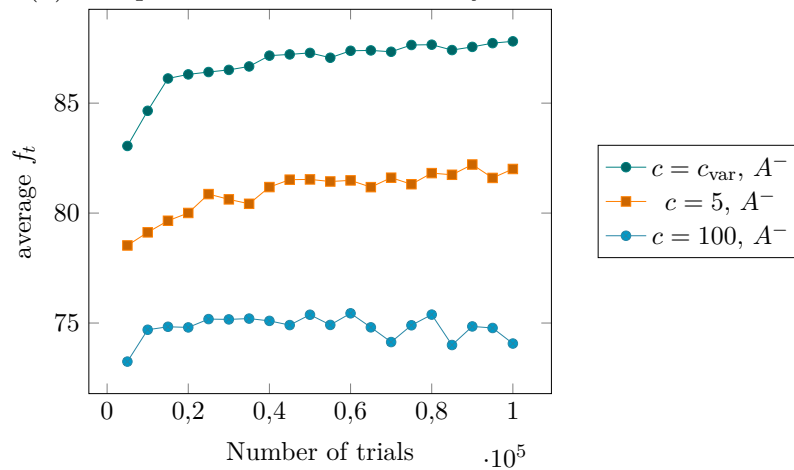
FIGURE A.4 – Exemples des chemins simulé à partir des différentes politiques.



(a) Comparaison de l'évolution moyenne de la fonction de valeur.



(b) Comparaison de l'évolution moyenne du taux de succès.



(c) Comparaison de l'évolution moyenne du temps de vol.

FIGURE A.5 – Comparaison des différents résultats en fonction du coefficient d'exploration utilisé.





# B

## Étude de l'impact de la durée de l'époque de planification sur la convergence et les chemins planifiés

Nous avons vu dans le Chapitre 4, qu'une époque de planification correspondait à plusieurs étapes GNC. Or la durée d'une époque va tout d'abord influencer le nombre minimum d'époques pour atteindre l'objectif et donc la taille de l'arbre de recherche. Jusqu'à présent, nous avons considéré qu'une époque correspondait à 10 étapes GNC soit  $k = 10$ , ce qui correspond à 4 secondes dans notre configuration. Augmenter la durée de cette époque aurait pour conséquence d'obtenir des politiques générant des chemins moins fluides puisqu'ils seraient composés de moins d'actions qui seraient exécutées plus longtemps. Diminuer la durée des époques pourrait permettre d'obtenir des chemins plus lisses et qui tiendraient compte plus précisément de l'environnement.

C'est pourquoi, nous avons décidé de comparer deux durées d'époques, la première durée  $k = 10$  qui est celle utilisée jusqu'à présent et  $k = 5$  (2 secondes). Cette étude a été faite sur la carte *WallBaffle* (Figure B.1) avec un coefficient d'exploration  $c = 100$  (voir Annexe A) et une pénalité de collision  $K = 450$  pour l'ensemble réduit d'actions  $A^-$ .

En nous basant sur les résultats de la figure B.2, qui représente les résultats moyens pour

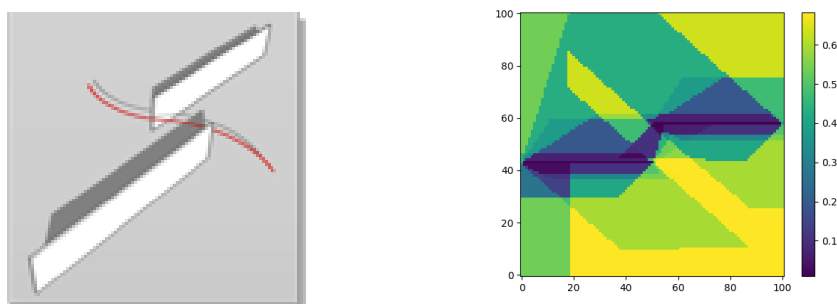
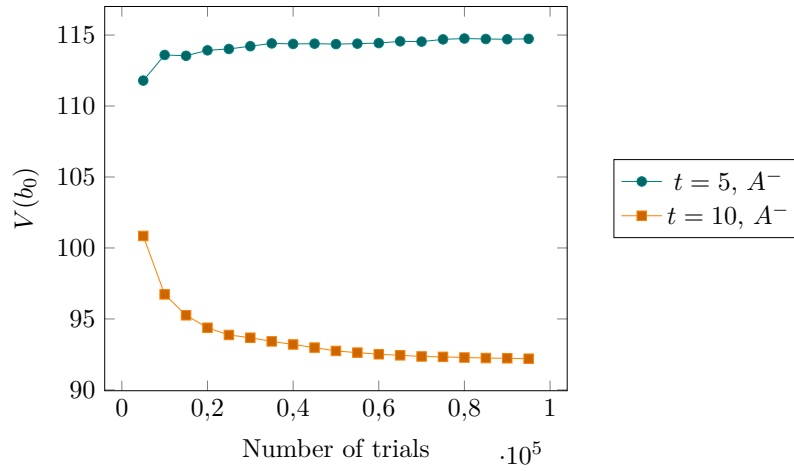
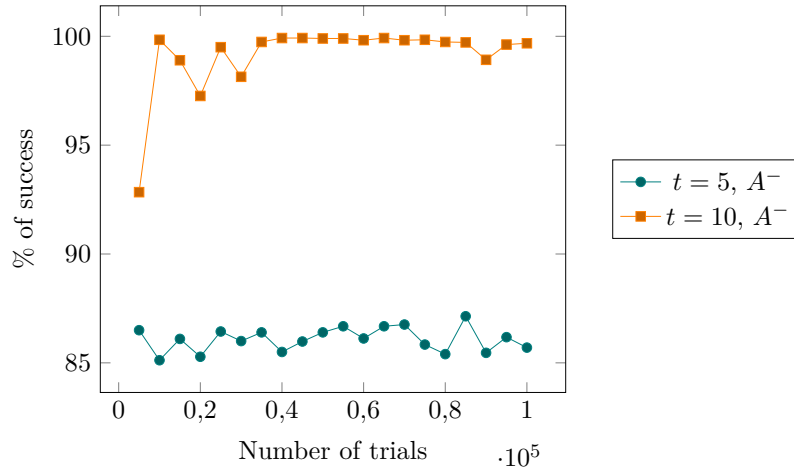


FIGURE B.1 – Représentation de la carte utilisé pour les test, ainsi que la carte de probabilité de disponibilité du GPS.

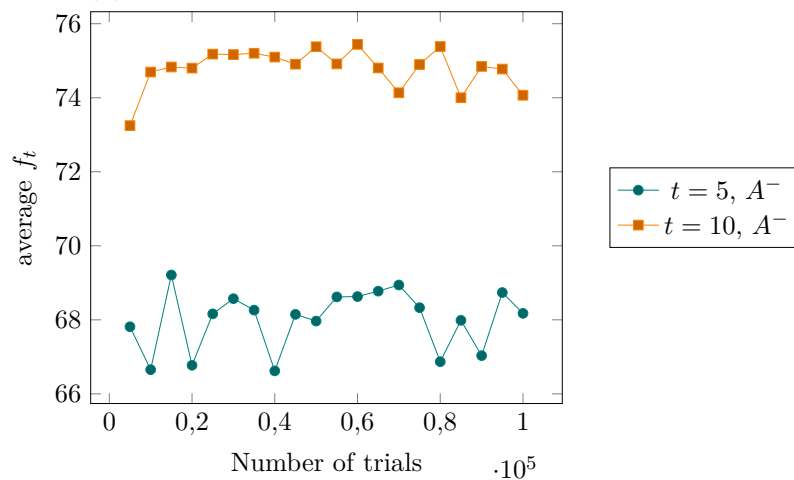
$k = 10$  et  $k = 5$ , nous pouvons dire qu'il y a un impact important de la durée de l'époque sur la convergence de la valeur de l'état de croyance initiale  $V(b_0)$  et sur les performances des politiques calculées. Tout d'abord pour l'ensemble réduit d'action, nous pouvons constater que choisir pour la durée d'une époque  $k = 10$  permet d'obtenir les meilleurs résultats avec un taux de succès de 100%. Cependant, pour une époque de  $k = 5$  nous obtenons un taux de succès 85%, mais un temps de vol plus faible. Le fait de raccourcir la durée de l'époque augmente la complexité de résolution du problème, car l'arbre de recherche grandit de façon exponentielle plus on diminue la durée d'une époque. Considérons qu'il faut au minimum  $t$  époques pour atteindre le but, alors la taille de l'arbre de recherche est au minimum de  $(|A| \times |O|)^t$ . En réduisant de moitié la durée d'une époque, alors la taille de l'arbre de recherche est au minimum de  $(|A| \times |O|)^{2t}$ , ce qui représente une augmentation exponentielle. Cela explique la difficulté pour l'algorithme de converger pour  $k = 5$ .



(a) Comparaison de l'évolution de la fonction de valeur.



(b) Comparaison de l'évolution du taux de succès.



(c) Comparaison de l'évolution du temps de vol.

FIGURE B.2 – Comparaison des différents résultats en fonction de la durée de l'époque pour l'ensemble d'action réduit  $A^-$ .





## Représentativité des simulations

Les taux de succès et les temps de vols présentés jusqu'à présent ont été obtenus en effectuant 1000 simulations par politique calculée. Cependant, il est important de choisir un nombre de simulations suffisamment grand pour être représentatif afin de pouvoir évaluer la qualité des politiques tout en restant raisonnable pour limiter le temps de calcul. Nous avons donc vérifié que 1000 simulations sont suffisantes pour évaluer correctement une politique. Pour cela nous avons repris les politiques calculées pour la carte *WallBaffle* avec une pénalité de collision  $K = 450$  et l'ensemble réduit d'actions ainsi qu'avec l'ensemble complet des actions. Puis nous avons regardé l'évolution du taux de succès et du temps de vol en fonction du nombre de simulations.

La figure C.1 présente les résultats obtenus, la colonne de gauche étant les résultats des politiques calculées avec l'ensemble réduit d'actions et la colonne de droite ceux des politiques calculées avec l'ensemble complet d'actions. Pour l'ensemble  $A^-$ , nous constatons que lorsque le nombre de simulations est fixé à 100, le taux de succès est de 100%. Or lorsque nous augmentons le nombre de simulations, nous voyons qu'en moyenne le taux de succès est de 99,6%. Quand nous comparons les résultats avec 1000 simulations et 10000 simulations, nous constatons que la différence est en moyenne de 0.1% (99.72% - 99.64%) à l'exception d'une politique où la différence est de 0.2%. De plus, pour 100000 simulations la différence par rapport à 10000 simulations est en moyenne de 0.06% (99.64% - 99.58%). L'évolution du temps vol moyen est en moyenne stable jusqu'à 10000 simulations (74s - 74.2s), puis augmente légèrement pour 100000 simulations (environ 74.7s). Cela représente donc dans cet exemple une évolution de moins d'un quart d'une époque (4s).

Pour l'ensemble  $A$ , nous constatons que pour 100 simulations le taux de succès est beaucoup plus élevé que pour un nombre plus grand de simulations. Néanmoins, nous constatons peu de différence entre 1000 et 10000 simulations, le taux de succès en moyenne est passé de 77.46% à 76.47. Pour 100000 simulations nous constatons une baisse pour la moitié des politiques et en moyenne la différence est de -2% de succès par rapport à 10000 simulations. L'évolution du temps de vol est en légère hausse et en moyenne passe de 69.9s à 71.5s ce qui représente une demi-époque du planificateur.

À partir de ces résultats, nous pouvons dire qu'augmenter le nombre de simulations permet bien d'affiner les résultats. Cependant, le gain obtenu en passant de 1000 simulations à 100000

simulations est faible comparé aux nombres de simulations et au temps de calcul nécessaires pour l'obtenir. C'est pourquoi nous avons décidé d'utiliser 1000 simulations pour évaluer les politiques.

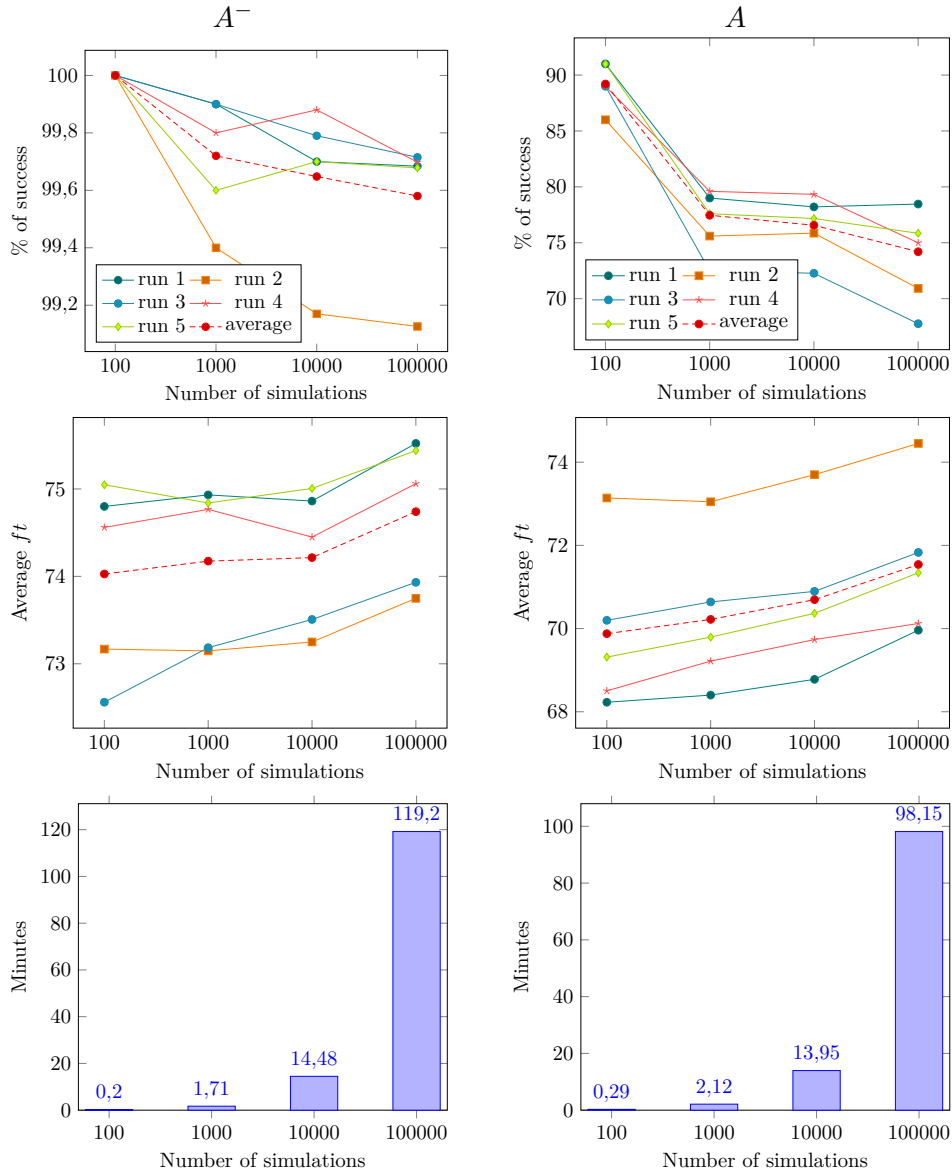


FIGURE C.1 – Évolution du taux de succès et du temps de vol en fonction du nombre de simulations pour  $K = 450$  avec l'ensemble réduit  $A^-$  et l'ensemble complet d'actions  $A$ .

## Bibliographie

- ACHTELIK, Markus W. et al. (2014). « Motion- and Uncertainty-aware Path Planning for Micro Aerial Vehicles ». In : *Journal of Field Robotics* (cf. p. 3, 39, 43).
- ARAYA-LÓPEZ, Mauricio et al. (2010). « A closer look at MOMDPs ». In : *22nd IEEE International Conference on Tools with Artificial Intelligence (ICTAI)* (cf. p. 4, 26, 27, 29, 55).
- ARTIEDA, Jorge et al. (2009). « Visual 3-d slam from uavs ». In : *Journal of Intelligent and Robotic Systems* 55.4-5, p. 299 (cf. p. 2).
- ATAEI, Mansour et Aghil YOUSEFI-KOMA (2015). « Three-dimensional optimal path planning for waypoint guidance of an autonomous underwater vehicle ». In : *Robotics and Autonomous Systems* 67, p. 23-32 (cf. p. 1).
- AUER, Peter, Nicolo CESA-BIANCHI et Paul FISCHER (2002). « Finite-time analysis of the multiarmed bandit problem ». In : *Machine learning* 47.2-3, p. 235-256 (cf. p. 25, 127).
- BAI, Haoyu, Shaojun CAI et al. (2015). « Intention-aware online POMDP planning for autonomous driving in a crowd ». In : *Robotics and Automation (ICRA), 2015 IEEE International Conference on*. IEEE, p. 454-460 (cf. p. 37, 39).
- BAI, Haoyu, David HSU et al. (2010). « Monte Carlo Value Iteration for Continuous-State POMDPs ». In : *Workshop on the algorithmic foundations of robotics* (cf. p. 5).
- BARTO, Andrew G, Steven J BRADTKE et Satinder P SINGH (1995). « Learning to act using real-time dynamic programming ». In : *Artificial intelligence* (cf. p. 24, 66).
- BELLMAN, Richard (1957). « A Markovian Decision Process ». In : *Indiana Univ. Math. J.* (4) (cf. p. 10, 50).
- BERG, Jur van den, Sachin PATIL et Ron ALTEROVITZ (2012). « Motion planning under uncertainty using iterative local optimization in belief space ». In : *The International Journal of Robotics Research* 31.11, p. 1263-1278 (cf. p. 3, 38).
- BERTSEKAS, Dimitri P (1995). *Dynamic programming and optimal control*. Athena scientific Belmont, MA (cf. p. 12).
- BONET, Blai et Hector GEFFNER (2003). « Labeled RTDP : Improving the Convergence of Real-Time Dynamic Programming. » In : *ICAPS* (cf. p. 5, 66).
- BONET, Blai et Héctor GEFFNER (2009). « Solving POMDPs : RTDP-Bel vs. Point-based Algorithms. » In : *IJCAI* (cf. p. 5, 24, 29, 66, 81).

- BRECHTEL, Sebastian, Tobias GINDELE et Rüdiger DILLMANN (2013). « Solving continuous POMDPs : Value iteration with incremental learning of an efficient space representation ». In : *In Proc. Int. Conf. on Machine Learning* (cf. p. 3, 43).
- BRY, Adam et Nicholas ROY (2011). « Rapidly-exploring random belief trees for motion planning under uncertainty ». In : *2011 IEEE International Conference on Robotics and Automation (ICRA)* (cf. p. 3, 32, 33, 35, 39, 43).
- BUFFET, Olivier et Olivier SIGAUD (2008). *Processus décisionnels de Markov en intelligence artificielle* (cf. p. 15, 16, 19).
- CABALLERO, Fernando et al. (2009). « Vision-based odometry and SLAM for medium and high altitude flying UAVs ». In : *Journal of Intelligent and Robotic Systems* 54.1-3, p. 137-161 (cf. p. 2).
- CAPITAN, Jesús, Luis MERINO et Ambal OLLERO (2016). « Cooperative decision-making under uncertainties for multi-target surveillance with multiples UAVs ». In : *Journal of Intelligent & Robotic Systems* 84.1-4, p. 371-386 (cf. p. 1).
- CASSANDRA, Anthony, Michael L LITTMAN et Nevin L ZHANG (1997). « Incremental pruning : A simple, fast, exact method for partially observable Markov decision processes ». In : *Proceedings of the Thirteenth conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc., p. 54-61 (cf. p. 19, 64).
- CENSI, Andrea et al. (2008). « A Bayesian framework for optimal motion planning with uncertainty ». In : *2008 IEEE International Conference on Robotics and Automation*. IEEE, p. 1798-1805 (cf. p. 35).
- CHANEL, Caroline P Carvalho, Florent TEICHTEIL-KÖNIGSBUCH et Charles LESIRE (2013). « Multi-target detection and recognition by uavs using online pomdps ». In : *Twenty-Seventh AAAI Conference on Artificial Intelligence* (cf. p. 1).
- DELAMER, Jean-Alexis, Yoko WATANABE et Caroline PONZONI CARVALHO CHANEL (2017a). « MOMDP modeling for UAV safe path planning in an urban environment ». In : *JFPDA 2017* (cf. p. 5, 77, 122).
- (2017b). « Towards a MOMDP model for UAV safe path planning in urban environment ». In : *IMAV 2017* (cf. p. 5, 77, 122).
- (2018). « MOMDP solving algorithms comparison for safe path planning problems in urban environments ». In : *IROS 2018* (cf. p. 6, 124).
- (2019). « Solving path planning problems in urban environments based on a priori sensors availabilities and execution error propagation ». In : *AIAA Scitech 2019 Forum*, p. 2202 (cf. p. 6, 124).
- DEMPSTER, Arthur P, Nan M LAIRD et Donald B RUBIN (1977). « Maximum likelihood from incomplete data via the EM algorithm ». In : *Journal of the Royal Statistical Society : Series B (Methodological)* 39.1, p. 1-22 (cf. p. 60).
- DEVAURS, Didier, Thierry SIMÉON et Juan CORTÉS (2016). « Optimal path planning in complex cost spaces with sampling-based algorithms ». In : *IEEE Transactions on Automation Science and Engineering* 13.2, p. 415-424 (cf. p. 1).
- DIJKSTRA, Edsger W (1959). « A note on two problems in connexion with graphs ». In : *Numerische mathematik* 1.1, p. 269-271 (cf. p. 84).
- DOLGOV, Dmitri et al. (2010). « Path planning for autonomous vehicles in unknown semi-structured environments ». In : *The International Journal of Robotics Research* (cf. p. 1).



- EATON, Christopher M, Edwin KP CHONG et Anthony A MACIEJEWSKI (2017). « Robust UAV path planning using POMDP with limited FOV sensor ». In : *Control Technology and Applications (CCTA), 2017 IEEE Conference on*. IEEE, p. 1530-1535 (cf. p. 1, 3).
- GEFFNER, Hector et Blai BONET (1998). « High-level planning and control with incomplete information using POMDPs ». In : *Proc. Fall AAAI Symposium on Cognitive Robotics* (cf. p. 24, 29).
- KAELBLING, Leslie Pack, Michael L LITTMAN et Anthony R CASSANDRA (1998). « Planning and acting in partially observable stochastic domains ». In : *Artificial intelligence* (cf. p. 17, 19, 64).
- KELLER, Thomas et Malte HELMERT (2013). « Trial-based heuristic tree search for finite horizon MDPs ». In : *Twenty-Third International Conference on Automated Planning and Scheduling* (cf. p. 82).
- KLEIJER, Frank, Dennis ODIJK et Edward VERBREE (2009). « Prediction of GNSS availability and accuracy in urban environments case study schiphol airport ». In : *Location Based Services and TeleCartography II*. Springer (cf. p. 2).
- KOCSIS, Levente et Csaba SZEPESVÁRI (2006). « Bandit based monte-carlo planning ». In : *ECML* (cf. p. 25, 26, 69).
- KOLOBOV, Andrey (2012). *Planning with Markov decision processes : An AI perspective*. T. 6. 1. Morgan & Claypool Publishers, p. 1-210 (cf. p. 12, 25, 66, 67).
- KURNIAWATI, Hanna, David HSU et Wee Sun LEE (2008). « SARSOP : Efficient Point-Based POMDP Planning by Approximating Optimally Reachable Belief Spaces. » In : *Robotics : Science and Systems* (cf. p. 23, 29, 64, 65, 81).
- KURNIAWATI, Hanna et Vinay YADAV (2016). « An online POMDP solver for uncertainty planning in dynamic environment ». In : *Robotics Research*. Springer, p. 611-629 (cf. p. 37).
- LAVALLE, Steven M (1998). « Rapidly-exploring random trees : A new tool for path planning ». In : (cf. p. 32).
- LITTMAN, Michael L, Thomas L DEAN et Leslie Pack KAELBLING (1995). « On the complexity of solving Markov decision problems ». In : *Proceedings of the Eleventh conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc., p. 394-402 (cf. p. 17).
- LITTMAN, Michael Lederman (1996). « Algorithms for sequential decision making ». In : (cf. p. 19).
- MADANI, Omid, Steve HANKS et Anne CONDON (1999). « On the undecidability of probabilistic planning and infinite-horizon partially observable Markov decision problems ». In : *AAAI/IAAI*, p. 541-548 (cf. p. 17, 81).
- METTLER, Bernie et al. (2010). « Benchmarking of obstacle field navigation algorithms for autonomous helicopters ». In : *66th Forum of the American Helicopter Society : " Rising to New Heights in Vertical Lift Technology", AHS Forum 66* (cf. p. 5, 71).
- MONAHAN, George E (1982). « State of the art—a survey of partially observable Markov decision processes : theory, models, and algorithms ». In : *Management Science* 28.1, p. 1-16 (cf. p. 19).
- ONG, Sylvie C. W. et al. (2009). « POMDPs for Robotic Tasks with Mixed Observability ». In : *Proceedings of Robotics : Science and Systems (RSS)* (cf. p. 3).

- (2010). « Planning under uncertainty for robotic tasks with mixed observability ». In : *The International Journal of Robotics Research* (cf. p. 4, 26-29, 52, 55).
- PADEN, Brian et al. (2016). « A survey of motion planning and control techniques for self-driving urban vehicles ». In : *IEEE Transactions on intelligent vehicles* 1.1, p. 33-55 (cf. p. 1).
- PAPADIMITRIOU, Christos H et John N TSITSIKLIS (1987). « The complexity of Markov decision processes ». In : *Mathematics of operations research* 12.3, p. 441-450 (cf. p. 17).
- PINEAU, Joelle, Geoff GORDON, Sebastian THRUN et al. (2003). « Point-based value iteration : An anytime algorithm for POMDPs ». In : *IJCAI*. T. 3, p. 1025-1032 (cf. p. 17, 21, 22, 29).
- PORTA, Josep M et al. (2006). « Point-based value iteration for continuous POMDPs ». In : *Journal of Machine Learning Research* (cf. p. 20, 23).
- PRENTICE, Sam et Nicholas ROY (2010). « The belief roadmap : Efficient planning in linear POMDPs by factoring the covariance ». In : *Robotics Research*. Springer (cf. p. 2, 3).
- RICHTER, Charles, William VEGA-BROWN et Nicholas ROY (2018). « Bayesian learning for safe high-speed navigation in unknown environments ». In : *Robotics Research*. Springer, p. 325-341 (cf. p. 3).
- SCHIRMER, Robert, Peter BIBER et Cyrill STACHNISS (2017). « Efficient path planning in belief space for safe navigation ». In : *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, p. 2857-2863 (cf. p. 35, 36).
- SCHWARZ, Gideon (mar. 1978). « Estimating the Dimension of a Model ». In : *Ann. Statist.* 6.2, p. 461-464 (cf. p. 61).
- SEILER, Konstantin M, Hanna KURNIAWATI et Surya PN SINGH (2015). « An online and approximate solver for pomdps with continuous action space ». In : *2015 IEEE international conference on robotics and automation (ICRA)*. IEEE, p. 2290-2297 (cf. p. 37).
- SHANI, Guy, Ronen I BRAFMAN et Solomon Eyal SHIMONY (2007). « Forward Search Value Iteration for POMDPs ». In : *IJCAI*, p. 2619-2624 (cf. p. 29).
- SHANI, Guy, Joelle PINEAU et Robert KAPLOW (2013). « A survey of point-based POMDP solvers ». In : *Autonomous Agents and Multi-Agent Systems* 27.1, p. 1-51 (cf. p. 22).
- SIGAUD, Olivier et Olivier BUFFET (2008). *Processus décisionnels de Markov en intelligence artificielle*. Lavoisier - Hermes Science Publications, p. 258 (cf. p. 16, 17).
- SILVER, David et Joel VENESS (2010). « Monte-Carlo planning in large POMDPs ». In : *Advances in neural information processing systems* (cf. p. 24, 29, 69, 81, 82, 84, 87).
- SMALLWOOD, Richard D et Edward J SONDIK (1973). « The optimal control of partially observable Markov processes over a finite horizon ». In : *Operations research* (cf. p. 3, 15, 17-19).
- SMITH, Trey et Reid SIMMONS (2004). « Heuristic search value iteration for POMDPs ». In : *Proceedings of the 20th conference on Uncertainty in artificial intelligence* (cf. p. 23, 29).
- SOMANI, Adhiraj et al. (2013). « DESPOT : Online POMDP planning with regularization ». In : *Advances in neural information processing systems*, p. 1772-1780 (cf. p. 37).
- SUNBERG, Zachary N et Mykel J KOCHENDERFER (2018). « Online algorithms for POMDPs with continuous state, action, and observation spaces ». In : *Twenty-Eighth International Conference on Automated Planning and Scheduling* (cf. p. 37).

- VANEGAS, Fernando et al. (2016). « Uav based target finding and tracking in gps-denied and cluttered environments ». In : *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, p. 2307-2313 (cf. p. 1).
- WAHARTE, Sonia et Niki TRIGONI (2010). « Supporting search and rescue operations with UAVs ». In : *2010 International Conference on Emerging Security Technologies*. IEEE, p. 142-147 (cf. p. 1).
- WATANABE, Yoko, Sylvain DESSUS et Sylvain FABIANI (2014). « Safe Path Planning with Localization Uncertainty for Urban Operation of VTOL UAV ». In : *AHS Annual Forum* (cf. p. 2, 3, 5, 34, 39, 43, 52).
- WATANABE, Yoko, Aurélien VEILLARD et Caroline CHANEL (2016). « Navigation and Guidance Strategy Planning for UAV Urban Operation ». In : *AIAA Infotech@ Aerospace* (cf. p. 34, 35, 39, 63).
- WHITE, Chelsea C (1991). « A survey of solution techniques for the partially observed Markov decision process ». In : *Annals of Operations Research* 32.1, p. 215-230 (cf. p. 19).
- YEE, Timothy et al. (2016). « Monte Carlo Tree Search in Continuous Action Spaces with Execution Uncertainty. » In : *IJCAI*, p. 690-697 (cf. p. 37).
- ZHANG, Nevin L et Wenju LIU (1996). *Planning in stochastic domains : Problem characteristics and approximation*. Rapp. tech. Technical Report HKUST-CS96-31, Hong Kong University of Science et Technology (cf. p. 19).



## Planification d'une stratégie de navigation et de guidage pour des drones autonomes dans des milieux encombrés

**Résumé** — Avec la demande croissante pour l'utilisation de drones autonomes dans des milieux urbains, la sûreté et l'efficacité de ces missions doivent être garanties. D'une part, il est connu que dans ces milieux la haute densité d'obstacles peut mettre en péril le vol en cas de collision. D'autre part, la capacité de navigation - estimation de la position et par conséquent l'erreur d'exécution - de ces drones dépend de la disponibilité et de la performance de leurs capteurs embarqués qui varient selon l'environnement. Dans ce contexte, cette thèse propose un cadre de planification de chemin sûr et efficace pour des drones en milieux encombrés. Nous avons fondé notre travail sur les Processus Décisionnel de Markov à Observabilité Mixte (MOMDP), car ils permettent de modéliser des processus décisionnels à long terme tout en prenant en compte l'incertitude sur l'environnement et son observabilité mixte. Nous proposons une modélisation du problème de planification qui intègre le système de guidage, navigation et contrôle (GNC), afin de mieux représenter la dynamique de la transition d'état pour un drone, ainsi que l'erreur d'exécution associée. La disponibilité probable, connue a priori, des capteurs embarqués, est aussi considérée dans le modèle. Afin d'assurer les contraintes de sûreté, nous proposons une fonction de coût qui nous permet de raisonner en termes de taux de collision maximal à respecter. Ceci permet au planificateur de définir des politiques qui sont à la fois efficaces - minimisation du temps de vol - et sûres, par le choix des chemins qui favorisent le respect du taux de collision maximal défini. De plus, dus à la complexité du problème de planification, nous proposons un nouvel algorithme de résolution "POMCP-GO", qui est basé sur deux algorithmes de l'état de l'art. Nous avons exhaustivement évalué cet algorithme pour notre cadre d'application.

**Mots clés** : Planification, MOMDP, UAV, GNC

## Navigation and guidance strategy planning for autonomous drones in clustered environments

**Abstract** — With the growing demand for the use of UAVs in urban environments, the safety and effectiveness of these missions must be guaranteed. It is known that in these environments the high density of obstacles can put flight at risk in the case collision. The navigation capacity - estimation of the position and therefore the execution error - of these UAVs depends on the availability and performance of their on-board sensors, which vary in the environment. In this context, this thesis proposes a safe and effective path planning framework for UAVs in clustered environments. We based our work on the Mixed Observability Markov Decision Processes (MOMDP) because it allows us to model long-term decision processes while taking into account environmental uncertainty and mixed observability. We propose a planning model that integrates the guidance, navigation and control system (GNC), in order to better represent the dynamics of the state transition for a drone, as well as the associated execution error. The probable availability, known a priori, of on-board sensors is also considered in the model. In order to ensure safety constraints, we propose a cost function that allows us to reason in terms of maximum allowable collision rate. This allows the planner to compute policies that are both effective - minimizing flight time - and safe, by choosing paths that comply with the defined maximum collision rate. In addition, due to the complexity of the planning problem, we propose a new "POMCP-GO" resolution algorithm, which is based on two state-of-the-art algorithms. We have thoroughly evaluated this algorithm for our application framework.

**Keywords** : Planing, MOMDP, mobile robotique, GNC