Détection et reconnaissance de cibles en ligne pour des UAV autonomes avec un modèle de type POMDP

Caroline P Carvalho Chanel[1,2], Florent Teichteil-Königsbuch[2], Charles Lesire[2]

1. Université de Toulouse - ISAE - Institut Supérieur de l'Aéronautique et de l'Espace
2. Onera - The French Aerospace Lab, F-31055, Toulouse, France
name.surname@onera.fr

Résumé : Cet article présente une mission pour la détection et reconnaissance de cibles menée par un véhicule aérien inhabité (UAV) autonome. La mission est modélisée par un Processus de Markov Partiellement Observable (POMDP). Le modèle POMDP traite dans un cadre unique des actions de perception (comme l'angle de prise de vue de la caméra) et des actions qui mènent à l'accomplissement de la mission (changement de zone, altitude de vol, atterrissage). La mission consiste à atterrir dans la zone qui contient une voiture dont le modèle reconnu est celui recherché, avec un état de croyance suffisant. Nous expliquons comment nous avons appris le modèle d'observation probabiliste du POMDP à partir d'une étude statistique des sorties de l'algorithme de traitement d'image. Cet algorithme utilisé pour reconnaître des objets dans la scène est embarquée sur notre UAV. Nous présentons aussi notre cadre *optimize-while-executing*, qui administre un sous-planificateur POMDP pour optimiser et exécuter en parallèle la politique avec des contraintes de temps associées à la durée des actions, et qui raisonne sur les états futurs possibles du système robotique. Finalement, nos résultats expérimentaux sont présentés. Ils démontrent que des techniques d'intelligence artificielle comme les POMDP peuvent être appliquées avec succès pour contrôler automatiquement des actions de perception et d'accomplissement de mission pour des missions complexes en temps contraint pour un UAV autonome.

1 Introduction

La détection et la reconnaissance de cibles par des véhicules aériens inhabités (UAV) autonomes est un domaine de recherche actif (Wang *et al.*, 2012), dû à la croissante mise en place de systèmes de drones dans des missions civiles et militaires. Dans ce genre de mission, la stratégie de décision de haut niveau est généralement donnée par une règle écrite à la main (survoler une zone donnée, atterrir, prendre une image, etc), qui dépend des événements stochastiques (détection d'une cible dans une zone donnée, cible reconnue), qui peuvent survenir lors de l'exécution de la règle de décision. A cause de la complexité de la construction automatiquement des règles de décision sous incertitude (Littman *et al.*, 1995; Sabbadin *et al.*, 2007), appelées politique, peu de systèmes de drones construisent et optimisent des politiques automatiquement.

Quand les incertitudes dans l'environnement viennent de l'exécution imparfaite des actions, ou des observations de l'environnement, les politiques de haut niveau peuvent être automatiquement générées par l'optimisation de Processus de Markov Partiellement Observable (POMDP) (Smallwood & Sondik, 1973). Ce modèle a été appliqué avec succès dans la robotique mobile (Candido & Hutchinson, 2011; Spaan, 2008), et aussi dans la robotique aérienne (Miller *et al.*, 2009; Schesvold *et al.*, 2003; Bai *et al.*, 2011). Cependant, dans ces applications, du moins pour les drones, le problème POMDP est supposé connu avant que la mission ne commence, permettant aux concepteurs d'optimiser la politique du drone hors-ligne sans contrainte de temps.

Toutefois, dans une mission de détection et de reconnaissance (Wang *et al.*, 2012), si on la considère comme un problème de décision séquentielle autonome dans l'incertain, certains paramètres du problème sont inconnus avant le vol. En effet, le nombre de cibles, les zones qui composent l'environnement, et les positions des cibles dans ces zones, sont des exemples de paramètres généralement inconnus avant le vol et doivent être automatiquement extraits au début de la mission (par exemple, par des techniques de traitement d'image) afin de définir le problème de décision séquentielle à optimiser.

Dans cet article nous avons étudié une mission de détection et reconnaissance de cible par un UAV

autonome, modélisée par un POMDP défini en ligne une fois que le nombre de zones à explorer et le nombre de cibles a été analysé en ligne. Nous pensons que ce travail est stimulant et original pour deux raisons au moins : (i) la mission de détection et reconnaissance de cibles est considérée comme un problème de planification séquentielle à long terme, avec des actions à la fois de perception (changement d'angle de vue, changement d'altitude) et d'accomplissement de mission (changement de zone, atterrissage) ; (ii) le POMDP est résolu en ligne pendant le vol, en tenant compte des contraintes de temps requises par la durée de la mission et des états futurs possibles du système robotique.

Mener entièrement de façon automatique une mission requiert plusieurs briques, techniques et théoriques, qui ne peuvent pas être décrites avec grande précision dans cet article, dû à la limite de pages. Ainsi, nous avons focalisé l'attention sur le modèle POMDP, avec une discussion détaillée sur la façon que nous avons appris le modèle d'observation à partir de données réelles. Et aussi, sur le cadre *optimize-while-executing*, que nous avons développé pour résoudre en ligne des problèmes POMDP relativement complexes sur des contraintes de durée de mission.

La section 2 introduit le modèle mathématique formel du POMDP. Dans la section 3, nous présentons le modèle POMDP utilisé pour notre mission de détection et reconnaissance de cibles par un UAV autonome. La section 4 explique comment nous optimisons et exécutons en parallèle la politique tenant compte des contraintes temporelles de durée d'action. Finalement, la section 5 expose et discute les résultats obtenus dans les expérimentations de notre approche. Ces résultats montrent que des techniques d'intelligence artificielle peuvent être appliquées à des missions de robotique aérienne relativement complexes, dont les règles de décision n'étaient pas auparavant automatisées, ni entièrement optimisées.

2 Cadre formel : POMDP

Formellement, un POMDP est défini comme un n-uplet $\langle S, A, \Omega, T, O, R, b_0 \rangle$ où : S est un ensemble d'états, A un ensemble d'actions, Ω un ensemble d'observations. Pour tout instant de décision $t \in \mathbb{N}$, $T : S \times A \times S \to [0; 1]$ est une fonction de transition entre les états où $\forall a \in A, \forall s_t \in S$, et $\forall s_{t+1} \in S$ on définit : $T(s_t, a, s_{t+1}) = p(s_{t+1} \mid a, s_t)$; $O : \Omega \times S \to [0; 1]$ une fonction d'observation où $\forall o_t \in \Omega, \forall a \in A$, et $\forall s_t \in S$, on définit : $O(o_t, s_t) = p(o_t|s_t, a)$; $R : S \times A \to \mathbb{R}$ une fonction de récompenses associées aux couples état-action $r(s, a)$; et b_0 une distribution de probabilité sur les états initiaux, appelée *état de croyance*. On note Δ l'ensemble des distributions de probabilités sur les états, appelé aussi espace (continu) d'états de croyance.

A chaque pas de temps t, l'agent choisit une action $a \in A$ étant donné l'état de croyance $b_t \in \Delta$, qui l'amène stochastiquement à un nouvel état $s_{t+1} \in S$; ensuite, l'agent perçoit une observation bruitée $o \in \Omega$. L'agent met à jour son *état de croyance*, grâce à la règle de Bayes, ce qui dépend de l'action réalisée, de l'observation reçue, et est fonction de l'état de croyance précédent ($s' = s_{t+1}$ suit l'état $s = s_t$).

$$b_a^o(s') = \frac{p(o|s') \sum_{s \in S} p(s'|s, a) b(s)}{\sum_{s \in S} \sum_{s'' \in S} p(o|s'') p(s''|s, a) b(s)} \tag{1}$$

L'objectif de la résolution d'un POMDP est de construire une politique, c'est-à-dire une fonction $\pi : \Delta \to A$, qui maximise un critère de performance. En robotique, où des buts symboliques de mission sont représentés par des récompenses numériques, il est généralement convenable d'optimiser l'espérance de la somme pondérée pour tout état de croyance initial (Cassandra *et al.*, 1996; Spaan & Vlassis, 2004) :

$$V^\pi(b) = E_\pi \left[\sum_{t=0}^{\infty} \gamma^t \sum_{s \in S} r(s_t, \pi(b_t)) b_t(s)) \middle| b_0 = b \right] \tag{2}$$

où γ est le facteur d'actualisation de la valeur. La valeur de la politique optimale π^* est définie par la fonction de valeur optimale qui satisfait l'équation d'optimalité de Bellman :

$$V^*(b) = \max_{a \in A} \left[\sum_{s \in S} r(s, a) b(s) + \gamma \sum_{o \in O} p(o|a, b) V^*(b_a^o) \right] \tag{3}$$

Cette fonction de valeur est linéaire par morceaux et convexe (Smallwood & Sondik, 1973), i.e, à l'instant $n < \infty$, la fonction de valeur V_n peut être représentée par des hyperplans sur Δ, nommés α-vecteurs. Un α-vecteur et l'action associée $a(\alpha_n^i)$ définissent une région dans l'espace de croyance pour lequel ce vecteur

maximise V_n. Donc, la valeur d'un état de croyance peut être définie comme $V_n(b) = \max_{\alpha_n^i \in V_n} b \cdot \alpha_n^i$. Et la politique optimale à cette étape est $\pi_n(b) = a(\alpha_n^b)$.

Des algorithmes récents de résolution hors ligne, comme PBVI (Pineau *et al.*, 2003), HSVI2 (Smith & Simmons, 2005), SARSOP (Kurniawati *et al.*, 2008) et symbolic PERSEUS (Poupart, 2005), et des algorithmes de résolution en ligne comme RTDP-bel (Bonet & Geffner, 2009) et AEMS (Ross & Chaib-Draa, 2007) approchent la fonction de valeur pour un ensemble borné d'états de croyance B, où $B \subset \Delta$. Ces algorithmes implémentent différentes heuristiques pour explorer l'espace d'états de croyance, et différentes techniques pour mettre à jour la fonction de valeur V, par un opérateur de $backup$. La fonction de valeur est représentée par une ensemble de α-vecteurs (à l'exception de RTDP-bel), et contient un nombre limité à $|B|$ de α-vecteurs.

3 Mission de détection et de reconnaissance de cibles

3.1 Description de la mission

Nous considérons un véhicule aérien inhabité (UAV) autonome qui doit détecter et reconnaître des cibles sous des contraintes réelles de l'environnement. La mission consiste à détecter et identifier un modèle particulier de voiture dans la scène, et atterrir à côté de cette cible recherchée parmi d'autres. A cause de la nature du problème, soit l'observabilité partielle due à la croyance probabiliste sur le modèles de voitures, la mission est modélisé par un POMDP. L'UAV peut effectuer des tâches de haut niveau de déplacement (changer de zone, atterrir) et de perception (changer l'angle de vue, changer d'altitude). Les voitures peuvent être dans n'importe quelle zone de l'environnement. Ces zones sont extraites en vol au cours de la mission, par un algorithme de traitement d'image (on suppose pas plus qu'une voiture par zone).

Le nombre total d'état dépend de plusieurs variables qui sont discretisées : le nombre de zones (N_z), d'altitudes de vol (H), d'angles de vue (N_Φ), de cibles ($N_{targets}$), de modèles de voiture (N_{models}), et d'un état terminal que caractérise la fin de la mission. Comme les voitures, i.e. les cibles, peuvent être dans n'importe quelle zone de l'environnement, et peuvent aussi a priori être de tous les modèles possibles, le nombre total d'état est calculé par :

$$|S| = N_z \cdot H \cdot N_\Phi \cdot (N_z \cdot N_{models})^{N_{targets}} + T_s$$

où T_s représente l'état terminal.

Pour ce cas d'application, nous considérons 4 observations possibles, soit $|\Omega| = 4$ en chaque état : $\{$*voiture non détectée, voiture détectée mais non identifiée, voiture identifiée comme cible, voiture identifiée comme non cible*$\}$. Ces observations reposent sur le résultat du traitement d'image (décrit dans la section 3.3).

Comme mentionné auparavant, les actions de haut niveau réalisées par l'UAV autonome sont : changer de zone, changer d'altitude de vol, changer d'angle de vue, atterrir. Le nombre d'actions de changement de zone dépend du nombre de zones considérées. Ces actions sont appelées $go\_to(\hat{z})$, où $\hat{z}$ représente la zone de destination. Changer l'altitude de vol dépend aussi du nombre d'altitudes considérées pour le vol de l'UAV. Ces actions sont appelées $go\_to(\hat{h})$, où $\hat{h}$ représente l'altitude désirée. L'action $land$, soit atterrir, peut être réalisé par l'UAV à tout moment, et finalise la mission. Nous considérons aussi l'action de changement d'angle de vue, appelée *change_view*, qui mène à changer d'angle de vue pour observer une zone. Deux angles sont pris en compte dans le modèle $\Phi = \{front, side\}$. Donc le nombre total d'actions est donné par : $|A| = N_z + H + (N_\Phi - 1) + 2$.

3.2 Dynamique du modèle

Nous allons maintenant décrire les modèles de transition et de récompense. Les effets de chaque action sont formalisés par des équations mathématiques, qui reposent sur des variables et des fonctions décrites par la suite, que nous aideront à comprendre l'évolution des états du POMDP.

3.2.1 Variables d'état

L'environnement est décrit par 7 variables discrètes d'état. Nous supposons une connaissance a priori sur l'environnement : il y a 2 cibles possibles et 2 modèles possibles, soit $N_{models} = \{cible, non\ cible\}$. Les variables d'état sont :

1. z avec N_z valeurs possibles, qui indique la position de l'UAV ;
2. h avec H valeurs possibles, qui indiquent les altitudes de vol ;
3. $\Phi = \{front, side\}$, qui indiquent l'angle de vue entre l'UAV et la voiture observée.
4. Id_{target_1} (resp. Id_{target_2}) avec N_{models} valeurs possibles, qui indiquent l'identité (modèle de la voiture) de la cible 1 (resp. cible 2) ;
5. z_{target_1} (resp. z_{target_2}) avec N_z valeurs possibles, qui indiquent la position de la cible 1 (resp. cible 2).

3.2.2 Fonctions de transition et de récompense

Pour définir la dynamique du modèle, on caractérise chaque action par :
— *effets* : description textuelle expliquant comment les variables d'état évoluent une fois l'action appliquée.
— fonction de transition T ;
— fonction de récompense R.

En ce qui concerne la notation utilisée, les variables primées représentent la variable d'état successeur, et les variables non primées l'état courant. De plus, on définit une fonction indicative : $\mathbb{I}_{\{cond\}}$ égale à 1 si la condition $cond$ est vérifiée, ou égale à 0 autrement. Une autre notation utile est la fonction $\delta_x(x')$ égale à 1 si $x = x'$, et égale à 0 autrement. Cette notation nous permet d'exprimer les différentes valeurs possibles prises par la variable d'état successeur x'.

A partir de mission préalables réalisées avec notre UAV, nous supposons que le changement de zone, d'altitude et l'atterrissage peuvent être considérées comme des actions déterministes : le fait d'aller d'une zone à l'autre, ou de changer l'altitude de vol par exemple sont toujours déterministes. Cependant, le problème est toujours un POMDP, parce que les observations des modèles des voitures sont probabilistes, de plus, il est prouvé que la complexité pour résoudre un POMDP est essentiellement liée aux effets probabilistes des observations plutôt qu'aux effets probabilistes des actions (Sabbadin *et al.*, 2007).

En outre, afin d'être compatible avec le modèle POMDP, qui suppose que les observations sont disponibles après que chaque action est réalisée, toutes les actions de notre modèle fournissent une observation des modèles de voiture. La seule observation possible, après atterrissage est *non détecté*, car cette action ne permet pas à l'UAV de prendre des images de l'environnement. Toutes les autres actions décrites ci-après permettent de prendre automatiquement des images de la scène en face de l'UAV. Ce qui donne lieu au traitement d'image et à la classification des symboles d'observation (voir plus loin). Comme la caméra est fixe, il est important de contrôler l'orientation de l'UAV afin d'observer les différentes parties de l'environnement.

action go_to($\hat{z}$)

Cette action amène l'UAV à la zone désirée. La dynamique est décrite ci-d'après, mais notez que si l'UAV est dans l'état terminal (T_s), cette action n'a pas d'effet ni coût associé (ce qui n'est pas formalisé ci-dessous).
— Effets : l'UAV change de zone.
— Fonction de transition :

$$\begin{aligned} T(s', \texttt{go\_to}(\hat{\texttt{z}}), s) \quad = \quad & \delta_{\hat{z}}(z')\delta_h(h')\delta_\Phi(\Phi') \\ & \delta_{Id_{target_1}}(Id'_{target_1})\delta_{z_{target_1}}(z'_{target_1}) \\ & \delta_{Id_{target_2}}(Id'_{target_2})\delta_{z_{target_2}}(z'_{target_2}) \end{aligned}$$

Ceci est en accord avec la définition de la fonction δ mentionnée précédemment. La fonction est différente de zéro seulement pour la transition vers l'état s' dans lequel les variables d'état de post-action sont toutes égales à variables d'état de pré-action, sauf la variable zone z' qui est égale à $\hat{z}$, la zone désirée.
— Fonction de récompense : $R(s, \texttt{go\_to}(\hat{\texttt{z}})) = C_{z,\hat{z}}$, où $C_{z,\hat{z}} < 0$ représente le coût de changer de z à $\hat{z}$. Pour l'instant, nous avons choisi d'utiliser un coût constant, parce que la consommation est difficile à mesurer avec une précision suffisante pour notre UAV. Et, aussi parce que la génération automatique du modèle POMDP ne prend pas encore en compte les coordonnées des zones. Ces coordonnées sont indispensables pour un calcul approprié de distances entre zones dans la définition des coûts de déplacement.

action go_to($\hat{h}$)

Cette action amène l'UAV à l'altitude de vol désirée. Comme pour l'action go_to($\hat{z}$), si l'UAV est dans l'état terminal (T_s), cette action n'a aucun effet ou coût.

— Effets : l'UAV change son altitude de vol à $\hat{h}$.

— fonction de transition :

$$\begin{aligned} T(s', \texttt{go\_to}(\hat{h}), s) &= \delta_z(z')\delta_{\hat{h}}(h')\delta_\Phi(\Phi') \\ &\quad \delta_{Id_{target_1}}(Id'_{target_1})\delta_{z_{target_1}}(z'_{target_1}) \\ &\quad \delta_{Id_{target_2}}(Id'_{target_2})\delta_{z_{target_2}}(z'_{target_2}) \end{aligned}$$

— Fonction de récompense : $R(s, \texttt{go\_to}(\hat{h})) = C_{h,\hat{h}}$, où $C_{h,\hat{h}} < 0$ représente le coût associé au changement d'altitude de h à $\hat{h}$. Ce coût peut aussi modéliser la consommation de carburant qui dépend de la distance entre altitudes. Ces coûts sont typiquement moins élevés que le changement de zone. Pour la même raison que pour l'action précédente, nous avons choisi d'utiliser une constante pour modéliser ce coût, telle quel $C_z < C_h$.

action change_view

Cette action change l'angle de vue de l'UAV par rapport à la voiture observée dans une zone. Nous supposons que toutes les voitures on la même orientation dans les zones (comme par exemple dans les parkings), donc chaque angle de vue a la même orientation pour toutes les zones. Comme pour les actions précédentes, si l'UAV est dans l'état terminal (T_s), cette action n'a pas d'effets ou coût.

— Effets : l'UAV change d'angle de vue (*front* à *side* et vice versa).

— Fonction de transition :

$$\begin{aligned} T(s', \texttt{change\_view}, s) &= \delta_z(z')\delta_{\hat{h}}(h') \\ &\quad (\mathbb{I}_{\{\Phi=front\}}\delta_{side}(\Phi') + \mathbb{I}_{\{\Phi=side\}}\delta_{front}(\Phi')) \\ &\quad \delta_{Id_{target_1}}(Id'_{target_1})\delta_{z_{target_1}}(z'_{target_1}) \\ &\quad \delta_{Id_{target_2}}(Id'_{target_2})\delta_{z_{target_2}}(z'_{target_2}) \end{aligned}$$

— Fonction de récompense : $R(s, \texttt{change\_view}) = C_v$, où $C_v < 0$ représente le coût de changer d'angle de vue. Ceci est représenté par une constante moins importante que pour les autres actions. Suivant nos précédentes hypothèses de coûts constants : $C_v \geq C_h > C_z$.

action land

Cette action finalise la mission de l'UAV autonome et l'amène à l'état terminal. Si l'UAV est dans l'état terminal (T_s) cette action n'a pas d'effets ni de coût.

— Effets : l'UAV finalise la mission, et va à l'état terminal.

— Fonction de transition : $T(s', land, s) = \delta_{T_s}(s')$

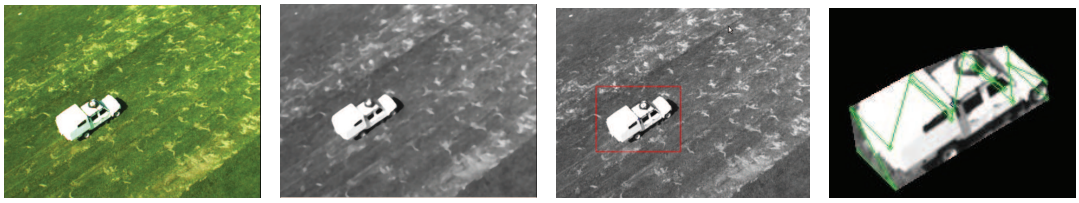
— Fonction de récompense :

$$\begin{aligned} R(s, \texttt{land}) = \quad & \mathbb{I}_{\{(z=z_{target_1})\&(Id_{target_1}=cible)\}}R_l + \\ & \mathbb{I}_{\{(z=z_{target_2})\&(Id_{target_2}=cible)\}}R_l + \\ & \mathbb{I}_{\{(z=z_{target_1})\&(Id_{target_1}=non\ cible)\}}C_l + \\ & \mathbb{I}_{\{(z=z_{target_2})\&(Id_{target_2}=non\ cible)\}}C_l + \\ & \mathbb{I}_{\{(z!=z_{target_1})\&(z!=z_{target_2})\}}C_l \end{aligned}$$

où $R_l > 0$ représente la récompense associée avec une mission réussie (l'UAV atterrir dans la zone où la cible recherchée se localise) et $C_l < 0$ représente le coût d'une mission raté. Notez que : $R_l \gg C_v \geq C_h > C_z \ggg C_l$.

3.3 Modèle d'observation

Les modèle de POMDP requièrent une description fidèle, ou appropriée des effets probabilistes des actions et des observations, ce qui est difficile à obtenir dans des applications réelles relativement complexes. Pour notre mission de détection et reconnaissance de cibles, nous avons automatiquement appris à partir



(a) Image d'entrée (b) Filtrage (c) Détection de voiture (d) Matching

FIGURE 1 – Détection et reconnaissance de cibles par un algorithme de traitement d'image (Saux & Sanfourche, 2011).

o_i	$p(o_i\|s)$
voiture non détectée	0.045351
voiture détectée mais non identifiée	0.090703
voiture identifiée comme cible	0.723356
voiture identifiée comme non cible	0.140590

TABLE 1 – Tableau de probabilités d'observation apprises à partir de l'analyse statistique des réponses du traitement d'image sur des données image réelles, avec $s = \{z = z_{target_1}, Id_{target_1} = cible, h = 30, z_{target_2} \neq z, Id_{target_2} = non\ cible\}$.

de données réelles, le modèle d'observation. Ce modèle repose sur le traitement d'image. Nous rappelons que nous considérons 4 observations possibles en chaque état : {*voiture non détectée, voiture détectée mais non identifiée, voiture identifiée comme cible, voiture identifiée comme non cible*}. La question clé consiste à attribuer une probabilité a priori sur les possibles sorties sémantiques du traitement de l'image pour une scène en particulier.

L'observation de voiture est basée sur un algorithme de reconnaissance d'objets par traitement d'image (Saux & Sanfourche, 2011), déjà embarqué sur notre UAV autonome. Ceci prend en entrée une image (voir Fig. 1(a)) qui provient de la caméra embarquée. D'abord, l'image est filtrée (Fig. 1(b)) pour détecter automatiquement la zone d'intérêt, soit la cible dans l'image (Fig. 1(c)). Si aucune cible n'est détectée l'algorithme renvoie directement le symbole *non détectée*. Si une cible est détectée, l'algorithme prend la zone d'intérêt de l'image (rectangle dans la Fig.1(c)), et génère une projection locale qui est comparée avec les gabarits 3D de la base de données de modèles de voiture (Fig. 1(d)). La projection locale dépend uniquement de l'altitude, de la focale et de l'azimut de la caméra comme paramètres de prise de vue. Finalement, le traitement d'image choisit le gabarit 3D qui maximise la similarité (voir Saux & Sanfourche, 2011), et renvoie le symbole qui correspond ou non avec la cible recherchée : *voiture identifiée comme cible* ou *voiture identifiée comme non cible*. Si le niveau de similarité est plus bas qu'un seuil défini a priori l'algorithme de traitement d'image renvoie le symbole *voiture détectée mais non identifiée*.

Pour apprendre le modèle d'observation POMDP à partir de données réelles, nous avons effectué des campagnes de prise de vue en plein air avec notre drone et quelques voitures connues. L'apprentissage conduit à un modèle d'observation appris par l'intermédiaire d'une analyse statistique des réponses de l'algorithme de traitement d'image en se basant sur les images prises lors de ces tests. Plus précisément, pour approcher la fonction d'observation $O(o_t, s_t)$, nous avons compté le nombre de fois qu'un des quatre symboles a été renvoyé par le traitement d'image comme réponse pour un état donné s_t. Donc nous avons construit $O(o_i, s) = p(o_i|s)$, où o_i est une des 4 observations possibles selon :

$$p(o_i|s) \simeq \frac{1}{N_{exp}} \sum_{n=1}^{N_{exp}} \mathbb{I}_{\{o_n = o_i|s\}}, \tag{4}$$

où N_{exp} représente le nombre d'expériences, soit le nombre de fois où l'algorithme de traitement d'image a été lancé pour les différentes images, et o_n le symbole obtenu pour l'expérience n. Notez que nous disposons de plus de 500 images par état, donc $N_{exp} \gg 1$ et ainsi nous admettons que les approximations statistiques sont suffisamment bonnes. Le tableau 1 montre un exemple de tableau de probabilité obtenu après apprentissage pour un état donné.

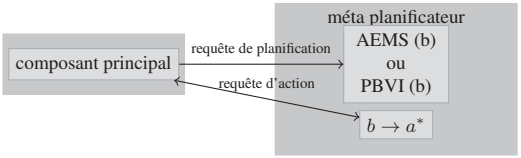


FIGURE 2 – Schéma du méta planificateur.

4 Cadre *optimize-while-execute*

Les problèmes POMDP grands et complexes peuvent rarement être optimisés en ligne à cause de l'absence de moyens de calcul suffisants. De plus, le problème à résoudre n'est pas toujours connu à l'avance. Par exemple, notre mission de détection et reconnaissance de cibles qui dépend du nombre de zones considérées qui sont automatiquement extraites de l'environnement par un traitement d'image pendant un vol préalable à la mission. De telles applications nécessitent un système efficace pour la résolution de POMDPs en ligne et pour l'exécution de politiques avant la date limite de la mission.

Nous avons travaillé dans l'extension du cadre *optimize-while-execute* proposé par Teichteil-Konigsbuch *et al.* (2011), auparavant limité à la planification déterministe ou de MDP (Processus de Décision de Markov), à résoudre en ligne des POMDPs soumis à des contraintes de temps. Notre extension est un méta planificateur qui repose sur les algorithmes standards de planification de POMDP tels quels PBVI, HSVI, PERSEUS, AEMS, etc. Ces algorithmes sont appelés pour d'éventuels états d'exécution futurs, lors de l'exécution de l'action optimisée courante à l'état courant d'exécution en anticipation de l'évolution du système probabiliste et de son environnement.

L'un des travaux de notre extension consiste à adapter les mécanismes de Teichteil-Konigsbuch *et al.* (2011) basés sur des états complètement observables, à des états de croyance et à des algorithmes $point-based$ utilisés par de nombreux planificateurs que l'on peut trouver dans l'état de l'art (Pineau *et al.*, 2003; Ross & Chaib-Draa, 2007). Ce cadre est différent des algorithmes tels quels RTDP-bel (Bonet & Geffner, 2009) et AEMS (Ross & Chaib-Draa, 2007) qui résolvent le POMDP *seulement* pour l'état courant d'exécution, mais pas depuis les états d'exécution possiblement futurs comme nous le proposons.

Nous avons implémenté le méta planificateur avec les algorithmes $anytime$ PBVI (Pineau *et al.*, 2003) et AEMS (Ross & Chaib-Draa, 2007). Ces algorithmes ont l'avantage d'apprendre des choses réutilisables aux prochains pas de temps, comme par exemple l'ensemble d'états de croyance de PBVI qui grandit au fur et à mesure que l'optimisation avance, et pour AEMS l'arbre des états de croyance atteignables qui est réutilisable pour le calcul des futurs états de croyance (car il suffit d'avancer dans l'arbre). De plus, AEMS est particulièrement intéressant et utile pour notre cadre *optimize-while-executing* avec des contraintes de temps, car nous pouvons explicitement contrôler le temps mis par AEMS pour optimiser une action pour un état de croyance donné.

Le méta planificateur gère les requêtes de planification et d'exécution en parallèle, comme montré dans la Fig. 2. En quelques mots, il fonctionne comme décrit dans ci-dessous :

1. Initialement, le méta planificateur planifie pour un état de croyance initial b à l'aide de PBVI ou AEMS pendant un certain temps ($bootstrap$).
2. Après, le méta planificateur reçoit une requête d'action pour laquelle il retourne l'action optimisée par PBVI ou AEMS pour b au composant principal.
3. Le temps approché d'exécution de l'action retournée est estimé, par exemple 8 secondes, donc le méta planificateur recevra des requêtes de planification pour chaque état de croyance futur et planifiera à l'aide de PBVI ou AEMS pendant une partie de ce temps disponible (i.e. 2 seconds pour chacun des 4 états de croyance futurs), pendant l'exécution de l'action courante. [1]
4. Ensuite, une fois que l'action courante a été exécutée, une observation est reçue et l'état de croyance est mis à jour, ce qui amène à un nouveau b' pour lequel l'action optimale courante est envoyée par le méta planificateur au composant principal par une requête d'action.

1. Notez qu'on devrait pourvoir faire mieux pour répartir le temps de planification entre les futurs états de croyance. Nous pourrions donner la priorité pour le calcul de l'action pour l'état de croyance le plus probable, ou pour celui où l'estimation de la valeur est la plus incertaine, ou encore diviser le temps de calcul d'un état de croyance de manière proportionnelle à la probabilité qu'il soit le prochain état de croyance.

Ce cadre propose un algorithme de planification continu qui prend pleinement en charge les incertitudes probabilistes : il construit plusieurs morceaux de politique pour le différents états probables d'exécution futurs.

Encore, comme montré dans la Fig. 2, les requêtes de planification et d'action sont les informations de base échangées entre le composant principal et le méta planificateur. Fait intéressant, est que chaque composant travaille sur un $thread$ indépendant. Plus précisément, le composant principal, qui est en charge de l'exécution de la politique, tourne sur le $thread$ d'exécution que interagit avec le moteur de l'exécution du système drone. Ceci s'exécute donc en parallèle du composant de planification, qui est en charge de l'optimisation des politiques. Le méta planificateur qui dirige le sous-planificateur de POMDP (PBVI ou AEMS) tourne sur le $thread$ d'optimisation.

Dans des applications réelles, les utilisateurs exigent que certaines garanties de base soient respectées par les systèmes autonomes. Par exemple, dans le cas des UAV, l'opérateur exige que la politique exécutée ne puisse jamais mettre l'UAV en danger, ce qui peut arriver dans des situations comme être en panne de carburant. Un autre danger peut survenir par le manque d'action optimisée pour un état courant d'exécution, si le processus d'optimisation n'a pas encore calculé une action réalisable dans cet état. Pour cette raison, il est indispensable que le méta planificateur puisse fournir une action pertinente et applicable à exécuter lorsque interrogé par le composant principal du système en fonction de l'état d'exécution courant.

Le manque d'action optimisée peut être résolu via une politique par défaut qui dépend de l'application. La politique par défaut peut être générée avant optimisation de différentes façons : soit une politique experte paramétrique calculée hors ligne dont les paramètres sont adaptés en ligne à la problématique réelle, soit une politique sous-optimale calculée très rapidement en ligne grâce à des heuristiques avant le calcul de la politique optimale. Des heuristiques simples mais complètes pour le calcul de politiques pour le POMDP, comme par exemple une approximation QMDP proposé par Littman *et al.* (1995), peuvent être très rapidement générées.

5 Résultats expérimentaux

Jusqu'à présent, nous avons réalisé des simulations réalistes et complètes (*hardware in the loop*), i.e. en utilisant l'architecture fonctionnelle embarquée exacte et les algorithmes utilisés à bord de notre UAV un Yamaha Rmax adapté au vol autonome, ainsi que de véritables images prises en plein air. Des vols réels sont mis à l'essai au moment où nous écrivons cet article. Dans cette section, nous présentons une analyse approfondie des résultats obtenus lors de nos simulations réalistes.

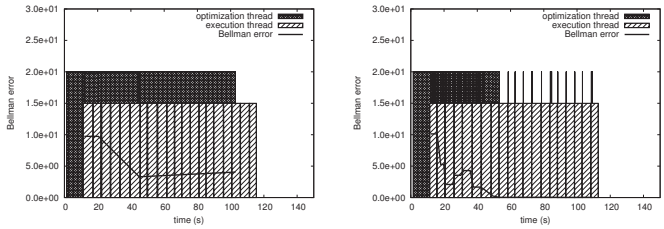
L'instance du problème considéré a 2 altitudes de vol (30 et 40 mètres), 2 angles de vue (latéral et frontal), 2 cibles, 2 modèles de voiture dans la base de donnés, et 3 zones, ce qui nous ramène à 433 états. Nous rappelons que nous avons 4 observations possibles. L'objectif est d'atterrir à côté du modèle de voiture montré dans la Fig. 1(d). Par contre, les modèles des voitures qui sont disposées dans l'environnement ne sont pas connus à l'avance.

Le cadre du méta planificateur présenté dans la section précédente est une bonne option à ce genre d'application car : (1) le nombre de zones est découvert pendant le vol, rendant impossible la résolution du problème avant vol ; et, (2) les algorithmes de résolution utilisés pour le POMDP – PBVI et AEMS – ne convergent pas dans le durée limite de la mission (20 pas de temps).

Notez que PBVI et AEMS sont des algorithmes $point-based$ qui approchent la fonction de valeur pour un ensemble dit pertinent d'états de croyance. PBVI choisi l'ensemble d'états de croyance en effectuant des tirages stochastiques depuis l'état de croyance initial. AEMS construit un arbre d'états de croyance depuis l'état de croyance initial et réalise l'expansion de cet arbre de croyance selon l'orientation de l'heuristique utilisée.

Nous considérons 2 états de croyance initiaux qui représentent 2 angles de vue différents initiaux et le fait que nous ne connaissons pas les positions et les modèles de voitures : b_0^1 (resp. b_0^2) est une distribution uniforme sur les variables états telles que $\{z = 1, h = 40, \Phi = front, z_{target_1} \neq z_{target_2}, Id_{target_1} \neq Id_{target_2}\}$ (resp. $\{z = 1, h = 40, \Phi = side, z_{target_1} \neq z_{target_2}, Id_{target_1} \neq Id_{target_2}\}$). La fonction de récompense est basée sur les constantes suivantes : $C_z = -5$, $C_h = -1$, $C_v = -1$, $R_l = 10$, et $C_l = -100$. La durée de chaque action est représentée par une distribution uniforme sur $[T^a_{min}, T^a_{max}]$, avec $T^a_{max} = 6s$ et $T^a_{min} = 4s$, ce qui est représentatif des durées observées pendant les vols préliminaires.

Les observations sont caractérisées par la sortie de l'algorithme de traitement d'image (Saux & Sanfourche, 2011), qui tourne en parallèle à l'optimisation dans le $thread$ d'exécution et qui est lancé a chaque



(a) PBVI (b) AEMS

FIGURE 3 – Frise chronologique pour les implémentations de PBVI et AEMS dans le cadre *optimize-while-executiong* en partant de b_0^1.

fois qu'une action est réalisée. Le simulateur, qui connait l'état réel de l'environnement considéré, prend une image de la base de données (toujours différente de la précédente si l'UAV ne change pas d'état) et envoie cette image à l'algorithme de traitement d'image, qui retourne un symbole d'observation au composant d'exécution.

La figure 3 montre les lignes de temps pour le processus d'exécution du méta planificateur. Ces lignes de temps représentent les périodes de temps où la politique a été optimisée ($thread$ d'optimisation) ou exécutée ($thread$ d'exécution) – les deux en parallèle –, ainsi que l'évolution de l'erreur de Bellman pendant la durée de la mission. Après une phase d'initialisation ($bootstrap$), où seulement le $thread$ d'optimisation est actif, nous pouvons noter que le processus d'optimisation se poursuit pendant quelque temps. Puis, des petits morceaux d'optimisation sont encore traitées lorsque des nouvelles requêtes de planification sont envoyées au méta planificateur, parce que la politique n'a pas été entièrement optimisée dans l'état de croyance concerné pendant les étapes d'optimisation précédentes.

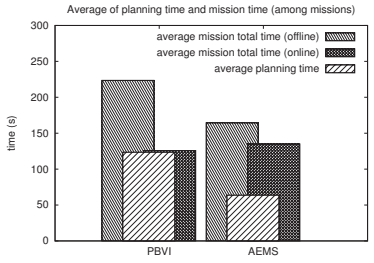
L'évolution de l'erreur de Bellman, rapportée à chaque requête de planification lors de l'optimisation, démontre l'évolution du processus d'optimisation. Dans la Fig. 3(a) la fonction de valeur ne converge pas pour l'ensemble complet des états de croyance concernés, contrairement à 3(b) où le processus d'optimisation a convergé pour l'état de croyance glissant. La raison est que AEMS est plus efficace que PBVI, du fait qu'il a suffisamment de temps pour optimiser les états futurs possibles de croyance pendant l'exécution d'action. Nous pouvons remarquer que le fil d'exécution se poursuit encore, mais les morceaux de temps d'optimisation sont très courts parce que l'erreur de Bellman est déjà très faible lorsque l'on commence à optimiser pour les états de croyance futurs.

La figure 4 montre des résultats comme le temps de planification et le pourcentage de missions réussies, pour les 2 sous-planificateurs PBVI et AEMS dirigés dans le cadre *optimize-while-executing* : le temps total moyen (en ligne) représente le temps jusqu'à la fin de la mission (i.e. nombre limité de pas de temps) ; le temps moyen de planification représente le temps pris par le $thread$ d'optimisation, qui est très proche du temps total de mission pour l'algorithme PBVI, parce que il ne converge pas pendant la durée de la mission. Ces moyennes ont été calculées à partir de 50 tests pour chaque instance du problème avec un horizon limite de 20 pas de temps ; chaque essai était une mission complète (optimisation et exécution en parallèle à partir de zéro). Pour faire une comparaison, nous montrons le temps d'une mission dite hors ligne, pour laquelle la politique serait optimisée d'abord pendant un certain temps (mais encore pendant le vol après l'extrait des zones de l'environnement en ligne), et puis la politique serait exécutée.

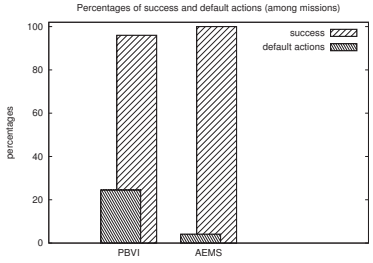
La figure 4 présente aussi le pourcentage d'actions par défaut utilisées et le pourcentage de missions réussies. Nous cherchons à montrer que, selon l'algorithme sous-jacent utilisé (PBVI ou AEMS), le $thread$ de planification ne réagit pas aussi rapidement qu'espéré, et plus d'actions par défaut sont réalisées. Nous rappelons que la politique par défaut est utilisé pour garantir la réactivité dans les cas où la politique optimisée n'est pas disponible dans l'état d'exécution courant. La politique par défaut a été rapidement calculé en début de mission avant de calculer la politique optimale. Nous avons choisi une politique heuristique basée sur l'approximation QMDP proposé par Littman *et al.* (1995). Le pourcentage de buts atteints (l'UAV a atterri dans la zone contenant le modèle de voiture recherché) est proche de 100%, ce qui montre que notre approche permet à l'UAV de réaliser statistiquement sa mission très bien.

Les figures 5(a) et 5(b) présentent les récompenses moyennes accumulées pendant le 50 tests d'exécution de la politique et stastiquement calculées par :

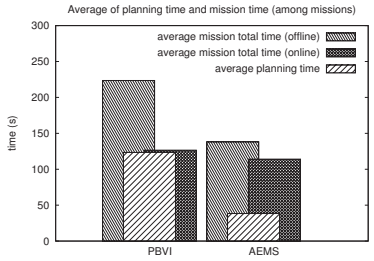
$$V^{\pi}(s_t) = \frac{1}{50} \sum_{50} \left[\sum_{k=0}^{t} \gamma^k r(s_k, \pi_k) | b_0, s_k \right] \tag{5}$$



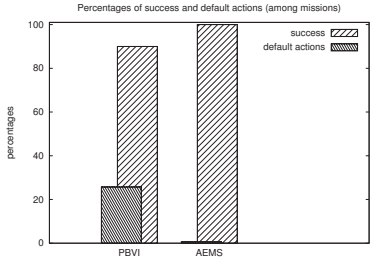
(a) Temps de planification pour b_0^1



(b) Pourcentages pour b_0^1



(c) Temps de planification pour b_0^2



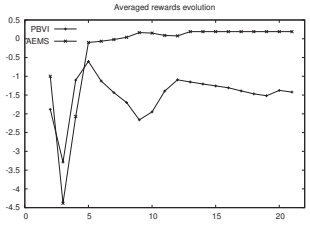
(d) Pourcentages pour b_0^2

FIGURE 4 – Résultats moyens pour les implémentations de PBVI et d'AEMS dans le cadre *optimize-while-executing* partant de b_0^1 et de b_0^2.

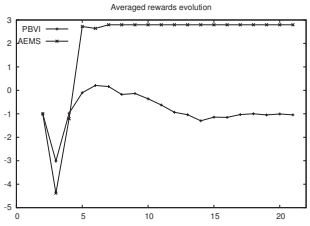
Notez que le simulateur utilise la connaissance de l'environnement (soit s_t, et tout les s_k), pour attribuer les récompenses pendant la simulation. De plus, cette équation nous permet de monter les récompenses moyennes accumulées à partir de l'instant zéro jusqu'à l'instant t.

Pour PBVI, indépendamment de l'état de croyance initial, les récompenses moyennes recueillies au cours des exécutions de politique tend à être moins important que pour AEMS. Nous croyons que cette différence vient du fait que PBVI est moins réactif (efficace) que AEMS. Ceci est montré par l'utilisation des actions par défaut, qui sont plus effectués pour PBVI et qui ne sont pas optimales pour la croyance dans lequel ils ont été appliqués.

Finalement, nous avons compté le nombre de fois que l'action $change\_view$ a été utilisée par la politique. Pour l'état de croyance initial b_0^1 (i.e. $\Phi = front$), cette action a été choisie 2 fois pour les 50 tests quelque soit le sous-algorithme utilisé. Et pour l'état de croyance initial b_0^2 (i.e. $\Phi = side$), cette action a été choisie 50 fois pour les 50 tests. Nous croyons que ce comportement provient du modèle d'observation, qui est plus discriminant lorsque $\Phi = front$ que quand $\Phi = side$: à partir d'une croyance initiale avec $\Phi = front$, l'algorithme d'optimisation ne trouve pas intéressant de changer l'angle de vue pour l'observation.



(a) Retour moyen pour b_0^1



(b) Retour moyen pour b_0^2

FIGURE 5 – Récompenses moyennes accumulées pour les implémentations de PBVI and d'AEMS dans le cadre *optimize-while-executing* partant de b_0^1 et de b_0^2.

6 Conclusion et travaux futurs

A de notre connaissance, ce document présente l'une des premières implémentations d'une mission de détection et de reconnaissance de cibles basé sur le modèle POMDP pour un UAV autonome. Nos études ont portée sur la modélisation du problème en tant que POMDP, ce qui nous permet de prendre en compte des actions de perception et d'accomplissement de mission dans un cadre unique ; et, sur l'apprentissage statistique du modèle probabiliste d'observation à partir des sorties de l'algorithme de traitement d'images. Ceci lancé sur un jeu d'images réelles prises en plein air. Ce modèle d'observation réaliste alimente le modèle POMDP.

Le coeur de notre contribution repose sur la proposition des moyens algorithmiques pratiques pour optimiser et exécuter des politiques POMDP en parallèle sous des contraintes de temps, ce qui est nécessaire dans le cas où le problème POMDP est généré pendant le vol.

Nous avons analysé des expériences réalisées avec des simulations réalistes et complètes (*hardware in the loop*) et basées sur des données réelles, qui démontrent que les techniques de planification POMDP sont maintenant assez matures pour s'attaquer à des missions de la robotique aérienne assez complexes, à condition d'utiliser un cadre *optimise-while-execute* tel que proposé dans le présent document.

Au moment où nous écrivons ce papier, nous embarquons nos composants de décision à bord de notre UAV autonome et nous commençons à effectuer des vols réels en plein air. De nombreuses améliorations peuvent être envisagées pour de futures recherches : (i) l'analyse de l'impact de différents états de croyance initiales sur la stratégie de décision ; (ii) la prise en compte des contraintes de sécurité imposées par les agences civiles aéronautiques lors de l'optimisation de la stratégie ; (iii) la prise en compte des incertitudes liées au modèle d'observation lors d'un calcul de la stratégie afin d'obtenir une politique robuste aux incertitudes pour les POMDP ; (iv) l'utilisation des POMDP factorisées, ou du modèle MOMDP (Ong *et al.*, 2009) qui permet de partitionner l'état de croyance en séparant la partie observable de la partie cachée, étant donné que l'incertitude de l'état de croyance repose sur l'identité de cibles.

Références

BAI H., HSU D., KOCHENDERFER M. & LEE W. S. (2011). Unmanned Aircraft Collision Avoidance using Continuous-State POMDPs. In *Proceedings of Robotics : Science and Systems*, Los Angeles, CA, USA.

BONET B. & GEFFNER H. (2009). Solving POMDPs : RTDP-bel vs. point-based algorithms. In *Proceedings of the 21st international jont conference on Artifical intelligence*, IJCAI'09, p. 1641–1646, San Francisco, CA, USA : Morgan Kaufmann Publishers Inc.

CANDIDO S. & HUTCHINSON S. (2011). Minimum uncertainty robot navigation using information-guided POMDP planning. In *ICRA'11*, p. 6102–6108.

CASSANDRA A., KAELBLING L. & KURIEN J. (1996). Acting under uncertainty : Discrete Bayesian models for mobile-robot navigation. In *Proceedings of IEEE/RSJ*.

KURNIAWATI H., HSU D. & LEE W. (2008). SARSOP : Efficient point-based POMDP planning by approximating optimally reachable belief spaces. In *Proc. RSS*.

LITTMAN M., CASSANDRA A. & PACK KAELBLING L. (1995). Learning policies for partially observable environments : Scaling up. In *International Conference on Machine Learning*, p. 362–370.

MILLER S. A., HARRIS Z. A. & CHONG E. K. P. (2009). A POMDP framework for coordinated guidance of autonomous UAVs for multitarget tracking. *EURASIP J. Adv. Signal Process*, **2009**, 2 :1–2 :17.

ONG S., PNG S., HSU D. & LEE W. (2009). POMDPs for robotic tasks with mixed observability. In *Robotics : Science and Systems*, volume 5, Seattle, USA.

PINEAU J., GORDON G. & THRUN S. (2003). Point-based value iteration : An anytime algorithm for POMDPs. In *Proc. of IJCAI*.

POUPART P. (2005). *Exploiting structure to efficiently solve large scale partially observable Markov decision processes*. PhD thesis, University of Toronto.

ROSS S. & CHAIB-DRAA B. (2007). AEMS : An anytime online search algorithm for approximate policy refinement in large POMDPs. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-07)*, p. 2592–2598.

SABBADIN R., LANG J. & RAVOANJANAHARY N. (2007). Purely epistemic markov decision processes. In *Proceedings of the 22nd national conference on Artificial intelligence - Volume 2*, p. 1057–1062 : AAAI Press.

SAUX B. & SANFOURCHE M. (2011). Robust vehicle categorization from aerial images by 3d-template matching and multiple classifier system. In *7th International Symposium on Image and Signal Processing and Analysis (ISPA)*, p. 466–470.

SCHESVOLD D., TANG J., AHMED B., ALTENBURG K. & NYGARD K. (2003). POMDP planning for high level UAV decisions : Search vs. strike. In *In Proceedings of the 16th International Conference on Computer Applications in Industry and Enineering*.

SMALLWOOD R. & SONDIK E. (1973). The optimal control of partially observable Markov processes over a finite horizon. *Operations Research*, p. 1071–1088.

SMITH T. & SIMMONS R. (2005). Point-based POMDP algorithms : Improved analysis and implementation. In *Proc. UAI*.

SPAAN M. (2008). Cooperative Active Perception using POMDPs. *Association for the Advancement of Artificial Intelligence - AAAI*.

SPAAN M. & VLASSIS N. (2004). A point-based POMDP algorithm for robot planning. In *ICRA*.

TEICHTEIL-KONIGSBUCH F., LESIRE C. & INFANTES G. (2011). A generic framework for anytime execution-driven planning in robotics. In *IEEE International Conference on Robotics and Automation (ICRA)*, p. 299–304.

WANG J., ZHANG Y., LU J. & XU W. (2012). A Framework for Moving Target Detection, Recognition and Tracking in UAV Videos. In J. LUO, Ed., *Affective Computing and Intelligent Interaction*, volume 137 of *Advances in Intelligent and Soft Computing*, p. 69–76. Springer Berlin / Heidelberg.