Planning for perception and perceiving for decision: POMDP-like online target detection and recognition for autonomous UAVs

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
2, avenue Edouard Belin
FR-31055 TOULOUSE

Abstract

This paper studies the use of POMDP-like techniques to tackle an online multi-target detection and recognition mission by an autonomous rotorcraft UAV. Such robotics missions are complex and too large to be solved off-line, and acquiring information about the environment is as important as achieving some symbolic goals. The POMDP model deals in a single framework with both perception actions (controlling the camera's view angle), and mission actions (moving between zones and flight levels, landing) needed to achieve the goal of the mission, i.e. landing in a zone containing a car whose model is recognized as a desired target model with sufficient belief. We explain how we automatically learned the probabilistic observation POMDP model from statistical analysis of the image processing algorithm used on-board the UAV to analyze objects in the scene. We also present our "optimize-while-execute" framework, which drives a POMDP sub-planner to optimize and execute the POMDP policy in parallel under action duration constraints, reasoning about the future possible execution states of the robotic system. Finally, we present experimental results, which demonstrate that Artificial Intelligence techniques like POMDP planning can be successfully applied in order to automatically control perception and mission actions hand-in-hand for complex time-constrained UAV missions.

Introduction

Target detection and recognition by autonomous Unmanned Aerial Vehicules (UAVs) is an active field of research (Wang et al. 2012), due to the increasing deployment of UAV systems in civil and military missions. In such missions, the high-level decision strategy of UAVs is usually given as a hand-written rule (e.g. fly to a given zone, land, take image, etc.), that depends on stochastic events (e.g. target detected in a given zone, target recognized, etc.) that may arise when executing the decision rule. Because of the high complexity of automatically constructing decision rules, called policy, under uncertainty (Littman, Cassandra, and Pack Kaelbling 1995; Sabbadin, Lang, and Ravoanjanahary 2007), few deployed UAV systems rely on automatically-constructed and optimized policies.

When uncertainties in the environment come from imperfect action execution or environment observation, high-level policies can be automatically generated and optimized using Partially Observable Markov Decision Processes (POMDPs) (Smallwood and Sondik 1973). This model has been successfully implemented in ground robotics (Candido and Hutchinson 2011; Spaan 2008), and even in aerial robotics (Miller, Harris, and Chong 2009; Schesvold et al. 2003; Bai et al. 2011). Yet, in these applications, at least for the UAV ones, the POMDP problem is assumed to be available before the mission begins, allowing designers to have plenty of time to optimize the UAV policy off-line.

However, in a target detection and recognition mission (Wang et al. 2012), if viewed as an autonomous sequential decision problem under uncertainty, the problem is not known before the flight. Indeed, the number of targets, zones making up the environment, and positions of targets in these zones, are usually unknown beforehand and must be automatically extracted at the beginning of the mission (for instance using image processing techniques), in order to define the sequential decision problem to optimize. In this paper, we study a target detection and recognition mission by an autonomous UAV, modeled as a POMDP defined during the flight after the number of zones and targets has been online analyzed. We think that this work is challenging and original for at least two reasons: (i) the target detection and recognition mission is viewed as a long-term sequential decision-theoretic planning problem, with both perception actions (changing view angle) and mission actions (moving between zones, landing), for which we automatically construct an optimized policy ; (ii) the POMDP is solved online during the flight, taking into account time constraints required by the mission's duration and possible future execution states of the system.

Achieving such a fully automated mission from end to end requires many technical and theoretical pieces, which can not be all described with highest precision in this paper due to the page limit. We focus attention on the POMDP model, including a detailed discussion about how we statistically learned the observation model from real data, and on the *"optimize-while-execute"* framework that we developed to solve complex POMDP problems online while executing the currently available solution under mission duration constraints. The next section introduces the mathematical model

of POMDPs. In Section 3, we present the POMDP model used for our target detection and recognition mission for an autonomous rotorcraft UAV. Section 4 explains how we optimize and execute the POMDP policy in parallel, dealing with constraints on action durations and probabilistic evolution of the system. Finally, Section 5 presents and discusses many results obtained while experimenting with our approach, showing that Artificial Intelligence techniques can be applied to complex aerial robotics missions, whose decision rules were previously not fully automated nor optimized.

Formal baseline framework: POMDP

A POMDP is a tuple $\langle S, A, \Omega, T, O, R, b_0 \rangle$ where S is a set of states, A is a set of actions, Ω is a set of observations, $T : S \times A \times S \rightarrow [0; 1]$ is a transition function such that $T(s_{t+1}, a, s_t) = p(s_{t+1} \mid a, s_t)$, $O : \Omega \times S \rightarrow [0; 1]$ is an observation function such that $O(o_t, s_t) = p(o_t|s_t)$, $R : S \times A \rightarrow \mathbb{R}$ is a reward function associated with a state-action pair, and b_0 is an initial probability distribution over states. We note Δ the set of probability distributions over the states, called *belief state space*. At each time step t, the agent updates its *belief state* defined as an element $b_t \in \Delta$ using Bayes' rule (Smallwood and Sondik 1973).

Solving POMDPs consists in constructing a policy function $\pi : \Delta \rightarrow A$, which maximizes some criterion generally based on rewards averaged over belief states. In robotics, where symbolic rewarded goals must be achieved, it is usually accepted to optimize the long-term average discounted accumulated rewards from any initial belief state (Cassandra, Kaelbling, and Kurien 1996; Spaan and Vlassis 2004):

$$V^{\pi}(b) = E_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t r(b_t, \pi(b_t)) \middle| b_0 = b\right] \quad (1)$$

where γ is the actualization factor. The optimal value V^* of an optimal policy π^* is defined by the value function that satisfies the bellman's equation:

$$V^*(b) = \max_{a \in A}\left[\sum_{s \in S} r(s, a)b(s) + \gamma \sum_{o \in O} p(o|a, b)V^*(b_a^o)\right] \quad (2)$$

Following from optimality theorems, the optimal value of belief states is piecewise linear and convex (Smallwood and Sondik 1973), i.e, at a step $n < \infty$, the value function can be represented by a set of hyperplanes over Δ, known as α-vectors. An action $a(\alpha_n^i)$ is associated with each α-vector, that defines a region in the belief state space for which this α-vector maximizes V_n. Thus, the value of a belief state can be defined as $V_n(b) = \max_{\alpha_n^i \in V_n} b \cdot \alpha_n^i$. And an optimal policy in this step will be $\pi_n(b) = a(\alpha_n^b)$.

Recent offline solving algorithms, e.g. PBVI (Pineau, Gordon, and Thrun 2003), HSVI2 (Smith and Simmons 2005), SARSOP (Kurniawati, Hsu, and Lee 2008) and symbolic PERSEUS (Poupart 2005), and online algorithms as RTDP-bel (Bonet and Geffner 2009) and AEMS (Ross and Chaib-Draa 2007) approximate the value function with a bounded set of belief states B, where $B \subset \Delta$. These algorithms implement different heuristics to explore the belief state space, and update the value of V, which is represented by a set of α-vectors (except in RTDP-bel), by a backup operator for each $b \in B$ explored or relevant. Therefore, V is reduced and contains a limited number $|B|$ of α-vectors.

Multi-target detection and recognition mission

Mission description

We consider an autonomous Unmanned Aerial Vehicle (UAV) that must detect and recognize some targets under real-world constraints. The mission consists in detecting and identifying a car that has a particular model among several cars in the scene, and land next to this car. Due to the nature of the problem, especially partially observability due to the probabilistic belief about cars' models, it is modeled as a POMDP. The UAV can perform both high-level mission tasks (moving between zones, changing height level, land) and perception actions (change view angle in order to observe the cars). Cars can be in any of many zones in the environment, which are beforehand extracted by image processing (no more than one car per zone).

The total number of states depends on many variables that are all discretized: the number of zones (N_z), the height levels (H), the view angles (N_Φ), the number of targets ($N_{targets}$) and car models (N_{models}), and a terminal state that characterizes the end of the mission. As cars (candidate targets) can be in any of the zones and be of any possible models a priori, the total number of states is:

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

where T_s represents the terminal states.

For this application case, we consider 4 possible observations, i.e. $|\Omega| = 4$, in each state: {*car not detected, car detected but not identified, car identified as target, car identified as non-target*}. These observations rely on the result of image processing (described later).

As mentioned before, the high level mission tasks performed by the autonomous UAV are: moving between zones, changing height level, land. The number of actions for moving between zones depends on the number of zones considered. These actions are called $go\_to(\hat{z})$, where $\hat{z}$ represents the zone to go to. Changing the height level also depends on the number of different levels at which the autonomous UAV can fly. These actions are called $go\_to(\hat{h})$, where $\hat{h}$ represents the desired height level. The land action can be performed by the autonomous UAV at any moment and in any zone. Moreover, the land action finishes the mission. We consider only one high level perception action, called *change_view*: change view angle when observing a given car, with two view angles $\Phi = \{front, side\}$. So, the total number of actions can be computed as: $|A| = N_z + H + (N_\Phi - 1) + 1$.

Model dynamics

We now describe the transition and reward models. The effects of each action will be formalized with mathematical equations, which rely on some variables and functions described below, that help to understand the evolution of the POMDP state.

State variables The world state is described by 7 discrete state variables. We assume that we have some basic prior knowledge about the environment:. there are two targets that can be each of only two possible models, i.e. $N_{models} = \{target, non-target\}$. The state variables are:

1. z with N_z possible values, which indicates the UAV's position;
2. h with H possible values, which indicates its height levels;
3. $\Phi = \{front, side\}$, which indicates the view angle between the UAV and the observed car;
4. Id_{target_1} (resp. Id_{target_2}) with N_{models} possible values, which indicates the identity (car model) of target 1 (resp. target 2);
5. z_{target_1} (resp. z_{target_2}) with N_z possible values, which indicates the position of target 1 (resp. target 2).

Transition and reward functions To define the model dynamics, let us characterize each action with:

- *effects*: textual description explaining how state variables change after the action is applied;
- transition function T;
- reward function R.

Concerning the notation used, the primed variables represent the successor state variables, and the variable not primed represent the current state. In addition, let us define the indicative function : $\mathbb{I}_{\{cond\}}$ equal to 1 if condition $cond$ holds, or to 0 otherwise; this notation is used to express the Bayesian dependencies between state variables. Another useful notation is $\delta_x(x')$ equal to 1 if $x = x'$, or to 0 otherwise; this notation allows us to express the possible different values taken by the successor state variable x'.

Based on previous missions with our UAV, we know that moving and landing actions are sufficiently precise to be considered deterministic: the effect of going to another zone, or changing flight altitude, or landing, is always deterministic. However, the problem is still a POMDP, because observations of cars' models is probabilistic ; moreover, it has been proved that the complexity of solving POMDPs essentially comes from probabilistic observations rather than from probabilistic action effects (Sabbadin, Lang, and Ravoanjanahary 2007).

Moreover, in order to be compliant with the POMDP model, which assumes that observations are available after each action is executed, all actions of our model provide an observation of cars' models. The only possible observation after the landing action is *non detected*, since this action does not allow the UAV to take images of the environment. All other actions described in the next automatically take images of the scene available in front of the UAV, giving rise to image processing and classification of observation symbols (see later). As the camera is fixed, it is important to control the orientation of the UAV in order to observe different portions of the environment.

action go_to($\hat{\mathtt{z}}$) This action brings the UAV to the desired zone. The dynamics is described next, but note that if the UAV is in the terminal state (T_s), this action has no effects and no cost (what is not formalized bellow).

- Effects: the UAV moves between zones.
- Transition function:

$$T(s', \mathtt{go\_to}(\hat{\mathtt{z}}), s) = \delta_{\hat{z}}(z')\delta_h(h')\delta_\Phi(\Phi') \\ \delta_{Id_{target_1}}(Id'_{target_1})\delta_{z_{target_1}}(z'_{target_1}) \\ \delta_{Id_{target_2}}(Id'_{target_2})\delta_{z_{target_2}}(z'_{target_2})$$

  which, according to the definition of function δ previously mentioned, is non-zero only for the transition where post-action state variables s' are all equal to pre-action state variables s, but the target zone z' that is equal to $\hat{z}$.
- Reward function: $R(s, \mathtt{go\_to}(\hat{\mathtt{z}})) = C_{z,\hat{z}}$, where $C_{z,\hat{z}} < 0$ represents the cost of moving from z to $\hat{z}$. For this moment we chose to use a constant cost C_z, because actual fuel consumption is difficult to measure with sufficient precision on our UAV. And also, because the automatic generation of the POMDP model does not take into account zone coordinates. Zone coordinates are needed for computing the distance between zones in order to model costs proportionaly to zones' distances.

action go_to($\hat{\mathtt{h}}$) This action leads the UAV to the desired height level. Like action go_to($\hat{\mathtt{z}}$), if the UAV is in the terminal state (T_s), this action has no effects and no cost.

- Effects: the UAV changes to height level $\hat{\mathtt{h}}$.
- Transition function:

$$T(s', \mathtt{go\_to}(\hat{\mathtt{h}}), s) = \delta_z(z')\delta_{\hat{h}}(h')\delta_\Phi(\Phi') \\ \delta_{Id_{target_1}}(Id'_{target_1})\delta_{z_{target_1}}(z'_{target_1}) \\ \delta_{Id_{target_2}}(Id'_{target_2})\delta_{z_{target_2}}(z'_{target_2})$$

- Reward function: $R(s, \mathtt{go\_to}(\hat{\mathtt{h}})) = C_{h,\hat{h}}$, where $C_{h,\hat{h}} < 0$ represents the cost of changing from height level h to $\hat{h}$. This cost also models the fuel consumption depending on the distance between altitudes. These costs are typically higher than costs for moving between zones. For the same reason as the previous action, we also chose to use a constant cost such that $C_z < C_h$.

action change_view This action changes the view angle of the UAV when observing cars. Due to environmental constraints, we assume that all cars have the same orientations in all zones (as in parking lots for instance), so that each view angle value has the same orientation for all zones. Like the previous actions, if the UAV is in the terminal state (T_s), this action has no effects and no cost.

- Effects: the UAV switches its view angle (*front* to *side* and vice versa).

- Transition function:

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

- Reward function: $R(s, \texttt{change\_view}) = C_v$, where $C_v < 0$ represents the cost of changing the view angle. It is represented by a constant cost that is higher than costs of all other actions. Following our previous constant cost assumptions: $C_v \geq C_h > C_z$.

action land This action finalizes the UAV mission, leading the autonomous UAV to the terminal state. If the UAV is in the terminal state (T_s), this action has no effects and no cost.

- Effects: the UAV finishes the mission, and goes to the terminal state.
- Transition function: $T(s', land, s) = \delta_{T_s}(s')$
- Reward function:

$$R(s, \texttt{land}) = \mathbb{I}_{\{(z=z_{target_1})\&(Id_{target_1}=target)\}}R_l + \\ \mathbb{I}_{\{(z=z_{target_2})\&(Id_{target_2}=target)\}}R_l + \\ \mathbb{I}_{\{(z=z_{target_1})\&(Id_{target_1}=non-target)\}}C_l + \\ \mathbb{I}_{\{(z=z_{target_2})\&(Id_{target_2}=non-target)\}}C_l + \\ \mathbb{I}_{\{(z!=z_{target_1})\&(z!=z_{target_2})\}}C_l$$

where $R_l > 0$ represents the reward associated with a correctly achieved mission (the UAV is in the zone where the correct target is located) and $C_l < 0$ represents the cost of a failed mission. Note that: $R_l \gg C_v \geq C_h > C_z \ggg C_l$.

Observation model

POMDP models require a proper probabilistic description of actions' effects and observations, which is difficult to obtain in practice for real complex applications. For our target detection and recognition missions, we automatically learned from real data the observation model, which relies on image processing. We recall that we consider 4 possible observations in each state: {*no car detected, car detected but not identified, car identified as target, car identified as non-target*}. The key issue is to assign a prior probability on the possible semantic outputs of image processing given a particular scene.

Car observation is based on an object recognition algorithm based on image processing (Saux and Sanfourche 2011), already embedded on-board in our autonomous UAV. It takes as input one shot image (see Fig. 1(a)) that comes from the UAV onboard camera. First, the image is filtered (Fig. 1(b)) to automatically detect if the target is in the image (Fig. 1(c)). If no target is detected, it directly returns the label *no detected*. If a target is detected, the algorithm takes the region of interest of the image (bounding rectangle on Fig. 1(c)), then generates a local projection and compares it with the 3D template silhouettes on a data base of car models (Fig. 1(d)). The local projection only depends on the UAV height level, and camera focal length and azimuth as viewing-condition parameters. The height level is known at every time step, and the focal length and the camera azimuth are fixed parameters. Finally, the image processing algorithm chooses the 3D template that maximizes the similarity (for more details see (Saux and Sanfourche 2011)), and returns the label that corresponds or not to the searched target: *car identified as target* or *car identified as non-target*. If the level of similarity is less than a hand-tuned threshold, the image processing algorithm returns the label *car detected but not identified*.

o_i	$p(o_i\|s)$
car not detected	0.045351
car detected but not identified	0.090703
car identified as target	0.723356
car identified as non-target	0.140590

Table 1: Probability observation table learned from statistical analysis of the image processing algorithm answers using real data, with $s = \{z = z_{target_1}, Id_{target_1} = target, h = 30, z_{target_2} \neq z, Id_{target_2} = non-target\}$.

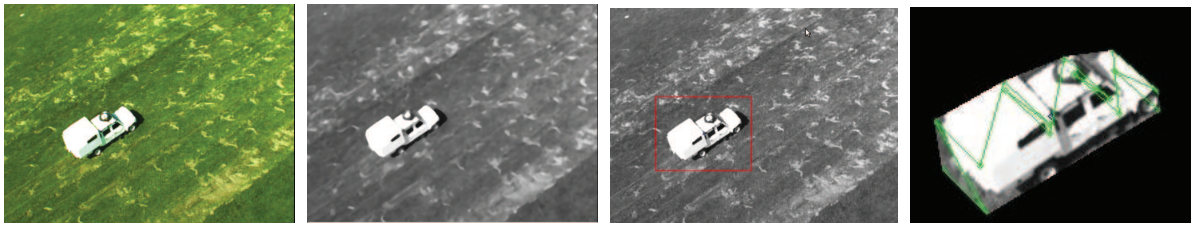
In order to learn the POMDP observation model from real data, we performed many outdoor test campaigns with our UAV and some known cars. It led to an observation model learned via a statistical analysis of the image processing algorithm's answers based on the images taken during these tests. More precisely, to approximate the observation function $O(o_t, s_t)$, we count the number of times that one of the four observations (labels) was an output answer of the image processing algorithm in a given state s. So, we compute $O(o_i, s) = p(o_i|s)$, where o_i is one of the 4 possible observations:

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

where N_{exp} represents the number of experiences, i.e. the number of runs performed by the image processing algorithm with respect to the different images, and o_n the label obtained at experience n. Note that we have access to more than 500 images for each state, so that $N_{exp} \gg 1$ and the statistical approximations may be good enough. Table 1 shows an example of observation probability obtained after learning in a given state.

Optimize-while-execute framework

Large and complex POMDP problems can rarely be optimized off-line, because of lack of sufficient computational means. Moreover, the problem to solve is not always known in advance, e.g. our target detection and recognition missions where the POMDP problem is based on zones that are automatically extracted from on-line images of the environment. Such applications require an efficient on-line framework for solving POMDPs and executing policies before the mission's deadline. We worked on extending the optimize-while-execute framework proposed



(a) Input image (b) Filtering (c) Car detection (d) Matching

Figure 1: Target detection and recognition image processing based on (Saux and Sanfourche 2011).

in (Teichteil-Konigsbuch, Lesire, and Infantes 2011), previously restricted to deterministic or MDP planning, to on-line solve large POMDPs under time constraints. Our extension is a meta planner that relies on standard POMDP planners like PBVI, HSVI, PERSEUS, AEMS, etc., which are called from possible future execution states while executing the current optimized action in the current execution state, in anticipation of the probabilistic evolution of the system and its environment. One of the issues of our extension was to adapt the mechanisms of (Teichteil-Konigsbuch, Lesire, and Infantes 2011) based on completely observable states, to belief states and point-based paradigms used by many state-of-the-art POMDP planners (Pineau, Gordon, and Thrun 2003; Ross and Chaib-Draa 2007). This framework is different from real-time algorithms like RTDP-bel (Bonet and Geffner 2009) that solve the POMDP *only* from the current execution state, but not from future possible ones as we propose.

We implemented this meta planner with the anytime POMDP algorithms PBVI (Pineau, Gordon, and Thrun 2003) and AEMS (Ross and Chaib-Draa 2007). AEMS is particularly useful for our optimize-while-execute framework with time constraints, since we can explicitly control the time spent by AEMS to optimize an action in a given belief state. The meta planner handles planning and execution requests in parallel, as shown in Fig. 2. At a glance, it works as described in the following:

1. Initially, the meta-planner plans for an initial belief state b using PBVI or AEMS during a certain amount of time (bootstrap).
2. Then, the meta-planner receives an action request, to which it returns back the action optimized by PBVI or AEMS for b.
3. The approximated execution time of the returned action is estimated, for instance 8 seconds, so that the meta planner will plan from some next possible belief states using PBVI or AEMS during a portion of this time (e.g. 2 seconds each for 4 possible future belief states), while executing the returned action.
4. After the current action is executed, an observation is received and the belief state is updated to a new b', for which the current optimized action is sent by the meta-planner to the execution engine.

This framework proposes a continuous planning algorithm that fully takes care of probabilistic uncertainties: it constructs various policy chunks at different future probabilistic execution states.

Furthermore, as illustrated in Fig. 2, planning requests and action requests are the core information exchanged between the main component and the planning component. Interestingly, each component works on an independent thread. More precisely, the main component, which is in charge of policy execution, runs in the execution thread that interacts with the system's execution engine. It competes with the planning component, which is in charge of policy optimization. The planning component runs in the optimization thread that drives the sub-POMDP planner.

Hence, due to thread concurrency, some data must be protected against concurrent memory access with mutexes: planning requests, and the optimized policy. Depending on the actual data structures used by the sub-POMDP planner, read and write access to the policy may be expensive. Therefore, in order to reduce CPU time required by mutex protection and to improve the execution thread's reactivity, we backup the policy after each planning request is solved.

In addition, in real critical applications, end-users often want the autonomous system to provide some basic guarantees. For instance, in case of UAVs, operators require that the executed policy never puts the UAV in danger, what may happen in many situations like being out of fuel. Another danger may come from the lack of optimized action in the current system state, due to the on-line optimization process that has not yet computed a feasible action in this state. For that reason it is mandatory that the meta-planner provides a relevant applicable action to execute when queried by the system's execution scheme according to the current execution state. It can be handled by means of an application-dependent *default policy*, which can be generated before optimization in two different ways: either a parametric off-line expert policy whose parameters are on-line adapted to

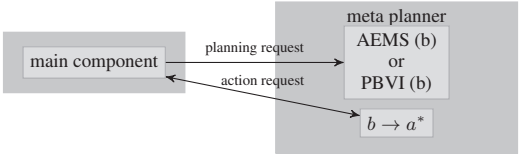


Figure 2: Meta planner planning / execution schema.

the actual problem; or a heuristic policy quickly computed on-line before computing the optimal policy. Simple but complete heuristic POMDP policies, for instance based on the QMDP approximation proposed by (Littman, Cassandra, and Pack Kaelbling 1995), can be quickly generated.

Experimental results

Up to now, we performed complete realistic "hardware in the loop" simulations, i.e. using the exact functional architecture and algorithms used on-board our UAV, a Yamaha Rmax adapted to autonomous flights, as well as real outdoor images. Real flights are being tested at the time we write this article. In this section, we present a deep analysis of results obtained during our realistic simulations.

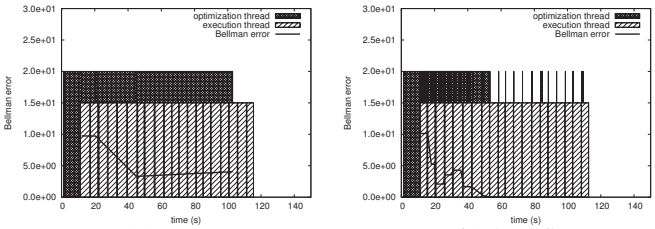
The instance of the problem considered has 2 height levels (30 and 40 meters), 2 view angles (front and side), 2 targets and 2 car models, and 3 zones, which leads to 433 states. Recall that we have 4 observation variables. The aim is to land next to the car whose model is presented in Fig. 1(d); however, the models of the cars is unknown at the beginning of the mission. The meta-planner on-line framework presented in the previous section is a good option for this problem because: (1) the number of zones is discovered in flight, making it impossible to solve the problem before the mission starts, and (2) the POMDP algorithms used – PBVI or AEMS – do not converge within the mission duration limit.

Note that PBVI and AEMS are point-based algorithms that approximate the value function for a set of relevant belief states. PBVI chooses the set of belief states by performing stochastic trials from the initial belief state. AEMS constructs a belief state tree beginning from the initial belief state and expanding this belief tree according to heuristic guidance means.

We consider two initial belief states that represent 2 different initial view angles and the fact that we do not know about the positions and the models of cars: b_0^1 (resp. b_0^2) is a uniform probability distribution over the 12 states $\{z = 1, h = 40, \phi = front, z_{target_1} \neq z_{target_2}, Id_{target_1} \neq Id_{target_2}\}$ (resp. $\{z = 1, h = 40, \phi = side, z_{target_1} \neq z_{target_2}, Id_{target_1} \neq Id_{target_2}\}$), The reward function is based on the following constants: $C_z = -5$, $C_h = -1$, $C_v = -1$, $R_l = 10$, and $C_l = -100$. The duration of an action is represented by a uniform distribution over $[T^a_{min}, T^a_{max}]$, with $T^a_{min} = 4s$ and $T^a_{max} = 6s$, which is representative of durations observed during preliminary test flights. We recall that we consider static targets.

The observations are characterized by the output of the image processing algorithm (Saux and Sanfourche 2011), which runs in parallel in a concurrent thread, and which is launched as soon as an action is performed. The simulator, which knows the real state of the world, takes an image from the data base and sends it to the image processing algorithm, which returns an observation to the execution component.

Figure 3 shows the timelines for the meta-planner execution process. It represents the periods of time where the policy is optimized (optimization thread) or executed (execution thread) – both running in parallel –, as well as the evolution of the Bellman error during the mission. After a



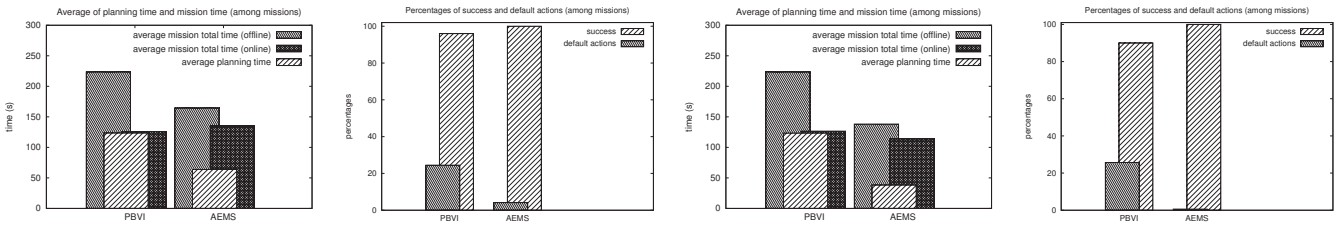
(a) PBVI (b) AEMS

Figure 3: Timelines for PBVI and AEMS implementations of the optimize-while-execute framework starting from b_0^1.

first bootstrap duration (where only the optimization thread is active), we can notice that the optimization process continues for a short time period. Then, small optimization chunks are still processed when new planning requests are sent to the planner, because the policy was previously not fully optimized in the current belief state during previous optimization chunks. The evolution of the Bellman error, reported for each planning request during optimization, emphasizes the evolution of the optimization process. In Fig. 3(a) the value function does not converge for all belief states in the relevant belief set, contrary to 3(b) where the optimization process has converged for the current (sliding) belief state. The reason is that AEMS is more efficient than PBVI, so that it has enough time to optimize the future possible belief states while executing actions. We can notice that the execution thread still goes on, but optimization chunks are very short because the Bellman error is already very small when beginning to optimize from each current belief state.

Figure 4 shows results for planning times and mission success percentages, for the 2 underlying POMDP solvers PBVI and AEMS driven by the optimize-while-execute framework: the average mission total time (on-line) represents the time until the end of the mission (i.e. limit time step); the average planning time represents the time taken by the optimization thread, that is very close to the mission total time for the PBVI algorithm, because it cannot converges during the mission time. These average results were computed over 50 test runs for each instance of the problem with a limit horizon of 20 steps ; each test run was a complete mission (optimization and execution in parallel from scratch). To make a comparison, we drown an *offline* mission time that would correspond to optimizing the problem off line (still during the flight after the zones are extracted from the environment in-flight), then executing the optimized policy.

Figure 4 also presents the percentage of default actions and achieved goals. We aim at showing that, depending on the underlying algorithm used (PBVI or AEMS), the planning thread does not react as fast as expected, and more default actions can be performed. We recall that default policy used guarantee reactivity in case the optimized policy is not available in the current execution state. The default policy was quickly computed before computing the optimal policy. We chose a heuristic policy based on the QMDP approximation proposed by (Littman, Cassandra, and Pack Kaelbling 1995).



(a) Planning time for b_0^1 (b) Percentages for b_0^1 (c) Planning time for b_0^2 (d) Percentages for b_0^2

Figure 4: Averaged results for PBVI and AEMS implementations of the optimize-while-execute framework, starting either from belief state b_0^1 or from b_0^2.

The average number of achieved goals (the UAV has landed in the zone containing the car that has the correct target model) is close to 100%, what shows that our approach allows the UAV to achieves its mission very well on average. But, for this kind of partial observable problem where the real nature of the targets is not known, and where the observation model is not exactly due to imprecision of the observation model learned from the image processing algorithm, we think that if targets positions are not static it may be impossible to achieve the goal 100% of the time.

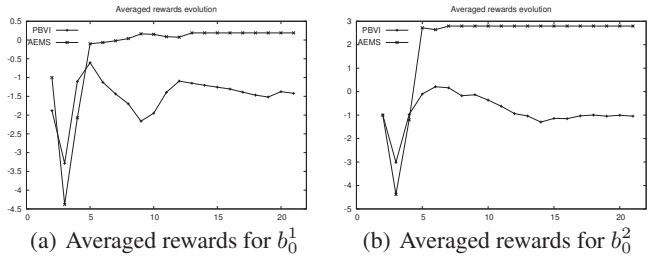
Figures 5(a) and 5(b) present the averaged return taken over 50 real policy executions, statistically computed as:

$$V^{\pi}(s_t) = \frac{1}{50} \sum_{50} \left[\sum_{k=0}^{t} \gamma^k r(s_k, \pi(b_k)) | b_0, s_k \right] \quad (3)$$

Note that the simulator uses its knowledge about the environent (i.e. the state s_t and all s_k), to attibuate the rewards while simulating. This equation allows us to show the accumulated rewards from the time step zero until time step t.

For PBVI, regardless of the initial belief state, the average return gathered during policy execution tends to be less important than for AEMS. We believe that this difference comes from the fact that PBVI is less reactive (efficient) than AEMS so that more default actions are performed, which are not optimal for the belief in which they were applied.

Finally, we counted the number of times that a $change\_view$ action was chosen by the policy, in order to evaluate the impact of the "perception action". For the initial belief state b_0^1 (i.e. $\Phi = front$), this action was chosen 2 times over 50 runs for PBVI, and also 2 times for AEMS. And, for b_0^2 (i.e. $\Phi = side$): 50 times with PBVI and 50 times for AEMS too. We believe that this behavior comes from the observation model, which is more discriminative when $\Phi = front$ than when $\Phi = side$: from an initial belief with $\Phi = front$, the policy optimization algorithm does not find interesting to change the observation view angle.



(a) Averaged rewards for b_0^1 (b) Averaged rewards for b_0^2

Figure 5: Average rewards for PBVI and AEMS implementations of the optimize-while-execute framework, starting either from belief state b_0^1 or from b_0^2.

Conclusion and future work

To the best of our knowledge, this paper presents one of the first *POMDP-based* implementations of a target detection and recognition mission by an autonomous rotorcraft UAV. Our contribution is threefold: (i) we model perception and mission actions in the same decision formalism using a single POMDP model; (ii) we statistically learn a meaningful probabilistic observation model of the outputs of an image processing algorithm that feeds the POMDP model; (iii) we provide practical algorithmic means to optimize and execute POMDP policies in parallel under time constraints, what is required because the POMDP problem is generated during the flight. We analyzed experiments conducted with a realistic "hardware in the loop" simulation based on real data, which demonstrate that POMDP planning techniques are now mature enough to tackle complex aerial robotics missions, on condition of using some kind of "optimize-while-execute" framework, as the one proposed in this paper.

At the time of writing this paper, we are embedding our decision-making components on-board the real UAV and beginning to conduct real outdoor flights. Many improvements can be considered for future research: analyzing the impact of different initial belief states on the optimized strategy; taking into account safety constraints imposed by civil aeronautical agencies when optimizing the strategy; building POMDP policies that are robust to imprecise observation models.

References

Bai, H.; Hsu, D.; Kochenderfer, M.; and Lee, W. S. 2011. Unmanned Aircraft Collision Avoidance using Continuous-State POMDPs. In *Proceedings of Robotics: Science and Systems*.

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

Candido, S., and Hutchinson, S. 2011. Minimum uncertainty robot navigation using information-guided POMDP planning. In *ICRA'11*, 6102–6108.

Cassandra, A.; Kaelbling, L.; and Kurien, J. 1996. Acting under uncertainty: Discrete Bayesian models for mobile-robot navigation. In *Proceedings of IEEE/RSJ*.

Kurniawati, H.; Hsu, D.; and Lee, W. 2008. SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. In *Proc. RSS*.

Littman, M.; Cassandra, A.; and Pack Kaelbling, L. 1995. Learning policies for partially observable environments: Scaling up. In *International Conference on Machine Learning*, 362–370.

Miller, S. A.; Harris, Z. A.; and 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.

Pineau, J.; Gordon, G.; and 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*. Ph.D. Dissertation, University of Toronto.

Ross, S., and 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)*, 2592–2598.

Sabbadin, R.; Lang, J.; and Ravoanjanahary, N. 2007. Purely epistemic markov decision processes. In *Proceedings of the 22nd national conference on Artificial intelligence - Volume 2*, 1057–1062. AAAI Press.

Saux, B., and 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)*, 466–470.

Schesvold, D.; Tang, J.; Ahmed, B.; Altenburg, K.; and 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., and Sondik, E. 1973. The optimal control of partially observable Markov processes over a finite horizon. *Operations Research* 1071–1088.

Smith, T., and Simmons, R. 2005. Point-based POMDP algorithms: Improved analysis and implementation. In *Proc. UAI*.

Spaan, M., and Vlassis, N. 2004. A point-based POMDP algorithm for robot planning. In *ICRA*.

Spaan, M. 2008. Cooperative Active Perception using POMDPs. *Association for the Advancement of Artificial Intelligence - AAAI*.

Teichteil-Konigsbuch, F.; Lesire, C.; and Infantes, G. 2011. A generic framework for anytime execution-driven planning in robotics. In *IEEE International Conference on Robotics and Automation (ICRA)*, 299–304.

Wang, J.; Zhang, Y.; Lu, J.; and Xu, W. 2012. A Framework for Moving Target Detection, Recognition and Tracking in UAV Videos. In Luo, J., ed., *Affective Computing and Intelligent Interaction*, volume 137 of *Advances in Intelligent and Soft Computing*. Springer Berlin / Heidelberg. 69–76.