POMDP-based online target detection and recognition for autonomous UAVs

Caroline P. Carvalho Chanel, F. Teichteil-Königsbuch, C. Lesire

ECAI 2012
Luc De Raedt et al. (Eds.)

doi:10.3233/978-1-61499-098-7-955

POMDP-based online target detection and recognition for autonomous UAVs

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

Abstract. This paper presents a target detection and recognition mission by an autonomous Unmanned Aerial Vehicule (UAV) modeled as a Partially Observable Markov Decision Process (POMDP). 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.

1 INTRODUCTION

Target detection and recognition by autonomous Unmanned Aerial Vehicules (UAVs) is an active field of research [18], due to the increasing deployment of UAV systems in civil and military missions. In such missions, high-level decision strategies of UAVs are usually given as hand-written rules (e.g. fly to a given zone, land, take image, etc.), which depend on stochastic events (e.g. target detected in a given zone, target recognized, etc.) that may arise when executing these decision rules. Because of the high complexity of automatically constructing decision rules under uncertainty [6, 10], often called policies in Artificial Intelligence planning, 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) [13]. This model has been successfully implemented in ground robotics [3, 15], and even in aerial robotics [7, 12, 1]. Yet, in these applications, at least for the UAV ones, the POMDP problem is assumed to be available before the mission begins, allowing system designers to have plenty of time to optimize the UAV policy off-line.

However, in a target detection and recognition mission [18], if viewed as an autonomous sequential decision problem under uncertainty, the decision problem is not known before the actual flight. Indeed, the number of targets, zones making up the environment, and positions of targets in these zones, are usually unknown beforehand. They 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 automatically analyzed and extracted online. 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. Section 2 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 with the 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.

2 FORMAL 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, $T : S \times A \times S \to [0; 1]$ is a transition function such that $T(s_{t+1}, a, s_t) = p(s_{t+1} \mid a, s_t)$, $O : \Omega \times S \to [0; 1]$ is an observation function such that $O(o_t, s_t) = p(o_t|s_t)$, $R : S \times A \to \mathbb{R}$ is a reward function associated with a state-action pair, and b_0 is the 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 [13].

Solving POMDPs consists in constructing a policy function π :

[1] Université de Toulouse – ISAE – Institut Supérieur de l'Aéronautique et de l'Espace; 10, av. Edouard Belin, FR-31055 Toulouse cedex 4;
[2] Onera – The French aerospace lab; 2, avenue Edouard Belin, FR-31055 Toulouse; name.surname@onera.fr

$\Delta \rightarrow A$, 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 [4, 16]:

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

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 [13], i.e, at the 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 $V_n(b) = \max_{\alpha_n^i \in V_n} b \cdot \alpha_n^i$. The corresponding optimal policy at this step will be $\pi_n(b) = a(\alpha_n^b)$.

Recent offline solving algorithms, e.g. PBVI [8], HSVI2 [14] and SARSOP [5], and online algorithms like RTDP-bel [2] and AEMS [9] 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. They update the value of V, which is represented by a set of α-vectors (except in RTDP-bel), using a backup operator for each $b \in B$ explored or relevant. Therefore, V is reduced and contains a limited number $|B|$ of α-vectors.

3 MULTI-TARGET DETECTION AND RECOGNITION MISSION

3.1 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, then landing next to this car. Due to the partially observable nature of the problem, especially 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 levels, landing) and perception actions (changing view angles 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), height levels (N_h), view angles (N_Φ), 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: $|S| = N_z \cdot N_h \cdot N_\Phi \cdot (N_z \cdot N_{models})^{N_{targets}} + T_s$, where T_s represents the terminal state.

For this application case, we consider 4 possible observations, i.e. $|\Omega| = 4$, in each state: {*no car 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 levels, changing view angles, landing. 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 *change_view* action changes the view angle when observing a given car, with two view angles $\Phi = \{front, side\}$. The $land$ action can be performed by the autonomous UAV all the time and in any zone. Moreover, the $land$ action completes the mission. So, the total number of actions can be computed as: $|A| = N_z + N_h + (N_\Phi - 1) + 1$.

3.2 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. These equations help to understand the evolution of the POMDP state.

3.2.1 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 N_h possible values, which indicates its height level;
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).

3.2.2 Transition and reward functions

To define the model dynamics, let us characterize each action with:

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

Concerning the notation used, the primed variables represent the successor state variables, whereas the unprimed ones 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 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 as 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 are probabilistic ; moreover, it has been proved that the complexity of POMDP solving essentially comes from probabilistic observations rather than from probabilistic action effects [10].

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 *no car detected*, since this action does not allow the UAV to take images of the environment. All other actions described below 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. Its 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 (which is not formalized below).

- Effects: the UAV moves from a zone to another one.
- 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 $\hat{z}$. This cost models the fuel consumption depending on the distance between zones. To simplify, 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 zones' coordinates. The latter are needed for computing distances between zones, which are assumed to be proportional to costs.

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's height level is changed to $\hat{\mathtt{h}}$.
- 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 (from *front* to *side* and vice versa).
- Transition function:

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

- Reward function: $R(s, \mathtt{change\_view}) = C_v$, 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 completes 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 ends its mission, and goes to the terminal state.
- Transition function: $T(s', land, s) = \delta_{T_s}(s')$
- Reward function:

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

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

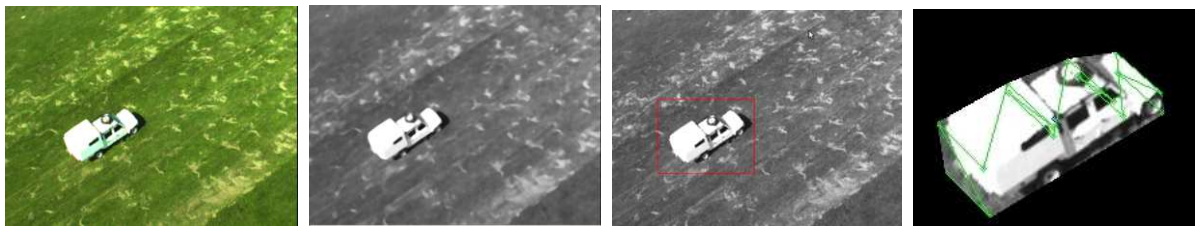
3.3 Observation model

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 deduced from an object recognition algorithm based on image processing [11], already embedded on-board our autonomous UAV. It takes as input one shot image (see Fig. 1(a)) that comes from the UAV on-board 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 car detected*. If a target is detected, the algorithm extracts 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 in a database of car models (Fig. 1(d)). The local projection only depends on the UAV's height level, camera focal length and azimuth as viewing-condition parameters. The height level is known at every time step, the focal length and the camera azimuth are fixed parameters. Finally, the image processing algorithm chooses the 3D template that maximizes similarity (for more details see [11]), 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*.

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 on the basis of 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 the following statistical estimation $\hat{p}(o_i|s)$, where o_i is one of the 4 possible observations:

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



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

Figure 1. Target detection and recognition image processing based on [11].

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. It applies the usual estimator of the mean of a Bernoulli distribution, which is equal to the probability of o_i against all other observations. It is proven to converge in probability to $p(o_i|s)$ as $N_{exp} \to \infty$. More than 500 images are available for each state ($N_{exp} \gg 1$) so that the statistical approximations may be good enough. We also computed intervals in which the true model lies with 99% confidence.

Table 1 shows an example of observation probability obtained after learning in a given state. We precompute tables for all possible states. No additional learning phase is performed during the real mission flight to improve the precision of the observation model. Note that, for the POMDP model, only $\hat{p}(o_i|s)$ is taken into account.

observation (o_i)	$\hat{p}(o_i\|s)$	$\sigma_{\hat{p}}$	$I = [\hat{p} \pm 3\sigma_{\hat{p}}]$
no car detected	0.0453	0.0091	[0.0183 ; 0.0723]
car detected but not identified	0.0907	0,0127	[0.0526 ; 0.1288]
car identified as target	0.7233	0,0199	[0.6636 ; 0.7830]
car identified as non-target	0.1405	0,0154	[0.0943 ; 0.1867]

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

4 OPTIMIZE-WHILE-EXECUTE FRAMEWORK

Large and complex POMDP problems can rarely be optimized offline, because of the 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 online framework for solving POMDPs and executing policies before the mission's deadline. We worked on extending the optimize-while-execute framework proposed in [17], previously restricted to deterministic or MDP planning, to solve on-line 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 [17] based on completely observable states, to belief states and point-based paradigms used by many state-of-the-art POMDP planners [8, 9].

We implemented this meta-planner on top of the anytime POMDP algorithms PBVI [8] and AEMS [9]. 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. Our purpose is not to improve existing algorithms, but to incorporate them into a more flexible framework that allows us to on-line solve POMDPs under time constraints.

The approach relies on using a meta planner that conducts an anytime POMDP planner, and that benefits from the current action's execution time to plan ahead for next future belief states. The meta planner handles planning and execution requests in parallel, as shown in Fig. 2. At a glance, it works as follows:

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 is different from real-time algorithms like RTDP-bel [2] or AEMS that solve the POMDP *only* from the current execution state (belief state), but not from future possible ones as we propose. Indeed, 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.

To compute next beliefs states, we ask the anytime planner about the probabilistic effects of the action that is being run in the current belief state. As we use the POMDP framework, we consider observations as effects, and so we construct the next belief states for each possible observation. For example: in the current belief state, we get the optimized action, next we predict the time that the UAV will spend to perform this action (e.g. T_a). Then, we ask the planner about the possible next effects (e.g. at most 4 observations). And so, we compute next belief states and optimize the policy for each of them during a time proportional to the action's predicted duration and to the number of next possible observations (e.g. $T_a/4$ for each).

Furthermore, as illustrated in Fig. 2, planning requests and action requests are the core information exchanged between the main component and the meta-planner. Interestingly, each component runs in 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 meta-planner component, which runs in a separate optimization thread. The meta-planner component, which is in charge of policy optimization, drives the sub-POMDP planner.

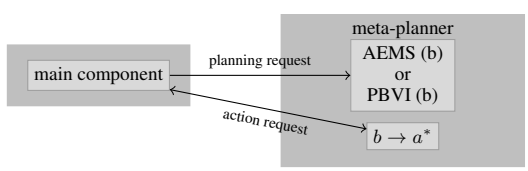


Figure 2. Meta-planner planning/execution schema.

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, which 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 schema, 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 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 [6], can be quickly generated.

5 EXPERIMENTAL RESULTS

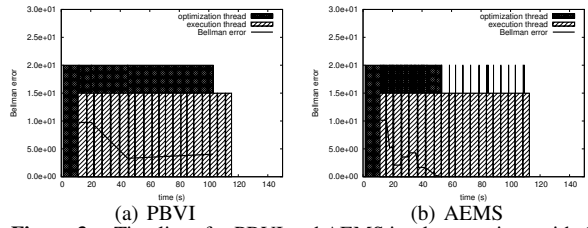
We performed complete realistic "hardware in the loop" simulations, i.e. using the exact functional architecture and algorithms used onboard our UAV, a Yamaha Rmax adapted to autonomous flights, as well as real outdoor images. Real flights have just been tested with success at the time we write this article. In this section, we present a deep analysis of results obtained during 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 possible observations. 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 beginning of the mission, and (2) the POMDP algorithms we used – PBVI or AEMS – do not converge within the mission duration limit.

Our experimental results are conducted on a small problem, but yet a real needed identification mission. The need for POMDPs in this kind of applications is indeed more related to the expressivity of probabilistic observation functions, than to the size of the problem. In such applications, problem size is not the crucial bottleneck. We believe that our approach will scale with bigger instances: in the optimize-while-execute framework, if the optimization algorithm does not provide an action in time, more default actions would just be performed. On the other hand, the longer an action lasts, the more the planner has time to improve the policy; thus, the scalability of our approach is also impacted by actions' actual durations.

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 the 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.

Observations are characterized by the output of the image processing algorithm [11], which runs in the execution thread. It is launched as soon as an action is performed. The simulator, which knows the real state of the world, takes an image from the database and sends it to the image processing algorithm, which returns an observation.



(a) PBVI (b) AEMS

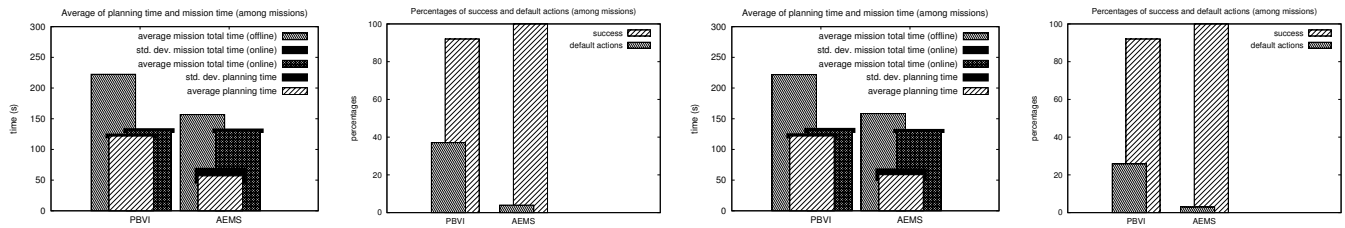
Figure 3. Timelines for PBVI and AEMS implementations with the optimize-while-execute framework starting from b_0^1.

Figure 3 shows the timelines for the meta-planner execution process. The x axis represents the time elapsed during the mission. Dark bars represent the portions of time where the policy is optimized (optimization thread) and light bars represent the portions of time where the policy is executed (execution thread) – both running in parallel. The curve shows the Bellman's error evolution during the mission. After a first bootstrap (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, which is monitored 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: after 50s the value function systematically converges before the current action execution has completed. 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 future belief state.

Figure 4 shows results for planning times and mission success percentages, using 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, which 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). As a comparison, we drown an *offline* mission time that would correspond to solving the problem off-line of execution (but still during the flight just after zones extraction from the environment), 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 may be performed. We recall that the default policy used guarantees reactivity in case the optimized policy would not be available in the current execution state. The default policy, implemented as a heuristic policy based on the QMDP approximation proposed by [6], was quickly computed before computing the optimal policy. The percentage of achieved goals (i.e. the UAV has landed in the zone containing the car that has the correct target model) is close to 100%, which highlights that our ap-

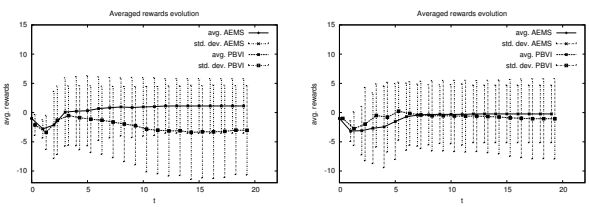


(a) Planning time for b_0^1 (b) Percentages for b_0^1 (c) Planning time for b_0^2 (d) Percentages for b_0^2

Figure 4. Averaged results for PBVI and AEMS implementations with the optimize-while-execute framework beginning from b_0^1 and b_0^2.

proach allows the UAV to achieve its mission very well on average.

Finally, figures 5(a) and 5(b) present the average over 50 real policy executions of the discount accumulated rewards, statistically computed as $V^{\pi}(s_t) = \frac{1}{50}\sum_{50}\left[\sum_{k=0}^{t}\gamma^k r(s_k, \pi(b_k))|b_0, s_k\right]$. Note that the simulator uses its knowledge of the environment (i.e. state s_t and all states s_k), to give out rewards while simulating the optimized policy. This equation allows us to show the accumulated rewards from time step zero until time step t. For PBVI, regardless of the initial belief state, the average rewards gathered during policy execution tend to be smaller 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.



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

Figure 5. Average rewards for PBVI and AEMS implementations with the optimize-while-execute framework starting from b_0^1 and b_0^2.

It is not surprising to observe a significant standard deviation on statistically averaged rewards. Indeed, the cars are actually not in the same zones in different missions, wich implies that the UAV lands sooner or later depending on the zone in which the searched car is.

6 CONCLUSION AND FUTURE WORK

To the best of our knowledge, this paper presents one of the first *POMDP-based* implementations of 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, which 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: they demonstrate that POMDP planning techniques are now mature enough to tackle complex aerial robotics missions, assuming the use of some kind of "optimize-while-execute" framework, as the one proposed in this paper.

At the time of writing this paper, we have just embedded our decision-making components on-board the real UAV and began to conduct real outdoor flights. Possible future research improvements include: 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

[1] Haoyu Bai, David Hsu, Mykel Kochenderfer, and Wee Sun Lee, 'Unmanned Aircraft Collision Avoidance using Continuous-State POMDPs', in *Proceedings of Robotics: Science and Systems*, Los Angeles, CA, USA, (June 2011).

[2] Blai Bonet and Hctor Geffner, 'Solving POMDPs: RTDP-bel vs. point-based algorithms', in *Proceedings of the 21st international jont conference on Artifical intelligence*, IJCAI'09, p. 16411646, San Francisco, CA, USA, (2009). Morgan Kaufmann Publishers Inc.

[3] Salvatore Candido and Seth Hutchinson, 'Minimum uncertainty robot navigation using information-guided POMDP planning.', in *ICRA'11*, pp. 6102–6108, (2011).

[4] A.R. Cassandra, L.P. Kaelbling, and J.A. Kurien, 'Acting under uncertainty: Discrete Bayesian models for mobile-robot navigation', in *Proceedings of IEEE/RSJ*, (1996).

[5] H. Kurniawati, D. Hsu, and W.S. Lee, 'SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces', in *Proc. RSS*, (2008).

[6] M.L. Littman, A.R. Cassandra, and L. Pack Kaelbling, 'Learning policies for partially observable environments: Scaling up', in *International Conference on Machine Learning*, pp. 362–370, (1995).

[7] Scott A. Miller, Zachary A. Harris, and Edwin K. P. Chong, 'A POMDP framework for coordinated guidance of autonomous UAVs for multitarget tracking', *EURASIP J. Adv. Signal Process*, 2:1–2:17, (Jan. 2009).

[8] J. Pineau, G. Gordon, and S. Thrun, 'Point-based value iteration: An anytime algorithm for POMDPs', in *Proc. of IJCAI*, (2003).

[9] S. Ross and B. Chaib-Draa, '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)*, pp. 2592–2598, (2007).

[10] Régis Sabbadin, Jérôme Lang, and Nasolo Ravoanjanahary, 'Purely epistemic markov decision processes', in *Proceedings of the 22nd national conference on Artificial intelligence - Volume 2*, p. 10571062. AAAI Press, (2007).

[11] B.L. Saux and M. Sanfourche, '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)*, pp. 466–470, (2011).

[12] D. Schesvold, J. Tang, B.M. Ahmed, K. Altenburg, and K.E. Nygard, '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*, (2003).

[13] R.D. Smallwood and E.J. Sondik, 'The optimal control of partially observable Markov processes over a finite horizon', *Operations Research*, 1071–1088, (1973).

[14] T. Smith and R.G. Simmons, 'Point-based POMDP algorithms: Improved analysis and implementation', in *Proc. UAI*, (2005).

[15] M.T.J. Spaan, 'Cooperative Active Perception using POMDPs', *Association for the Advancement of Artificial Intelligence - AAAI*, (2008).

[16] M.T.J. Spaan and N. Vlassis, 'A point-based POMDP algorithm for robot planning', in *ICRA*, (2004).

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

[18] Jiabao Wang, Yafei Zhang, Jianjiang Lu, and Weiguang Xu, 'A Framework for Moving Target Detection, Recognition and Tracking in UAV Videos', in *Affective Computing and Intelligent Interaction*, ed., Jia Luo, volume 137 of *Advances in Intelligent and Soft Computing*, 69–76, Springer Berlin / Heidelberg, (2012).