A multi-thread decisional architecture for real-time planning under uncertainty

Florent Teichteil-Königsbuch and **Patrick Fabiani**
ONERA-DCSD
2 avenue Édouard-Belin – BP 74025
31055 Toulouse Cedex 4 – France
{florent.teichteil,patrick.fabiani}@onera.fr

Abstract

We present a robotics architecture centered on the supervision function for an autonomous rotorcraft. It is made up of a reactive layer and of a deliberative layer, which run on two independent processors and with different temporal constraints. The deliberative layer is centered on the supervisor, which exchanges states and actions with a planning independent server. The latter is multi-threaded, so that it optimizes strategies while it interacts with the supervisor. This framework allows us to implement innovative anytime stochastic planning methods on-board a real autonomous rotorcraft.

Introduction

The operation of unmanned systems in partially known environments require higher levels of autonomy in order to guaranty safety whatever the situation complexity. Decision making tools, situation assessment tools or mission planning tools can improve the operator's situation awareness and help to manage the mission. However, the management of safety issues by a remote ground operator is prone to lead to undesired mission interruptions or situation mishandling, leading to apply unnecessary drastic measures or to fail to react to unexpected events. This requires safe and efficient capabilities of on-board situation assessment, planning and plan execution within the embedded robot architecture.

As a first step, it is crucial to demonstrate the feasibility of embedded decision making and plan execution capabilities in hostile, unknown or simply non cooperative environments. The benefits of using Artificial Intelligence planning techniques for real-world autonomous aircraft are studied on the case of an exploration problem in a Search and Rescue scenario in the ReSSAC project at ONERA. It turned out that even recentest stochastic planning methods can not be used to produce plans in a reasonable time compared with the dynamic of aircraft's actions. It lead us to develop an anytime multi-thread framework for stochastic planning methods, which is the subject of this article.

In the first section, we set out the ReSSAC project and the Search and Rescue scenario, which we worked on. In the second section, we present the architecture embedded on-board ReSSAC's autonomous rotorcrafts, by pointing out the decisional functions and their connections with the other functionalities. In the third section, we present the multi-thread planning system, which we developed for real-time decision-making under uncertainty, and its consequences for the embedded architecture in terms of plan execution. In the fourth section, we show results obtained during flight tests, which exhibit the relevance of our approach.

Acting and planning in an uncertain world

The ReSSAC project

The ReSSAC project at ONERA (Fabiani *et al.* 2007) lead us to study and demonstrate the feasibility of enabling uninhabited systems with embedded perception and decision making capabilities. Search and Rescue missions were taken as a case study. Two remote control Yamaha RmaX helicopters have been bought as experimental platforms. Both uninhabited helicopters were equipped with on-board avionics systems and a flight control architecture providing them with the capabilities of autonomous take-off, autonomous flight, autonomous navigation and autonomous landing in known areas and known environments. The on-board avionics, the ground station and data links have been developed with strong safety and security requirements. The flight control architecture was implemented so as to provide all the desirable reactive functions, namely the real time reaction capabilities of the aircraft. It is designated as the reactive layer. The decision making capabilities and vision capabilities do not need to run with the same reaction time and may require more resources in order to provide interesting results. These capabilities constitute the deliberative layer.

Exploration for a Search and Rescue mission

In the Search and Rescue scenario of the ReSSAC project, the aircraft is initially given a GPS position close to which the person to be rescued is supposed to be, a number of forbidden flight zones and an initial search region within which it should find a landing site. The vision and decision capabilities are used in order to gather information on the search region and replan the action strategy to be applied in the remaining of the mission accordingly. The reactive layer provides all the basic flight and motion functions for the achievement of the planned tasks.

Before flying in a partially known environment, possibly obstructed with obstacles, the aircraft must perform an exploration of the initial search region. It can select a number of zones of interest to be further explored within the initial search region. Each smaller zone can be explored and mapped through a sweeping trajectory of the aircraft. Depending on the height of the trajectory, the vision algorithms will perform differently. High obstacles can be detected at high altitudes, but smaller obstacles, or the estimation of the terrain slope, require the helicopter to fly at lower ground height, therefore between the higher obstacles.

It is therefore necessary to perform the exploration mission in several phases, including a re-planning of the mission in the middle.

- Phase One
  - initial navigation and exploration of the search region: planned using our waypoints and itinerary planner generating a navigation map and systematic exploration (sweeping) trajectories in order to explore the initial search region at sufficient height above ground (50 meters in fact)
  - geometric decomposition of the search zone into obstacles and sub-zones of interest, further assessed in terms of probability of "goodness" for landing purposes
  - Mission re-planning by optimizing a conditional strategy (reactive decision rule under uncertainty) for the exploration of the ranked zones of interest, taking into account the probabilities of failure or success in finding a site appropriate for landing.
- Phase Two
  - Execution of the exploration strategy,
  - choice of a sub-zone to characterize and application of Phase Three until safe landing in the search zone, or safety landing in the safety zone, or safe return to base
- Phase Three
  - Test, by use of a laser telemeter, of the slope of the sub-zone eventually selected in Phase Two,
  - then if the slope is OK proceed with selection of a landing spot based on vision and automatic landing, otherwise resume exploration in Phase Two.

This approach clearly requires the system to be able to replan its mission according to the encountered situation and according to environment uncertainties.

Embedded architecture

As shown in Figure 1, our embedded architecture is divided into 2 parts: the deliberative layer and the reactive layer. Each layer runs on a specific computer. Both layers essentially interact with each other via data exchanges. The flight control functions are executed on the reactive layer under real time constraints: they have been validated separately and need to be able to run independently from the deliberative layer in order to permanently ensure the security of the flight – this is a requirement in order to obtain the official flight authorizations. The deliberative layer can thus be allowed to consume more memory and computing time resources, without penalizing the real time functions that are vital for maintaining the aircraft in flight. In other words, the main difference between the two layers, and the reason why it is more adapted to separate them is that the deliberative and reactive layers do not have the same time constraints, and this is highly desirable in order to allow the implementation of effective embedded decision making capabilities.

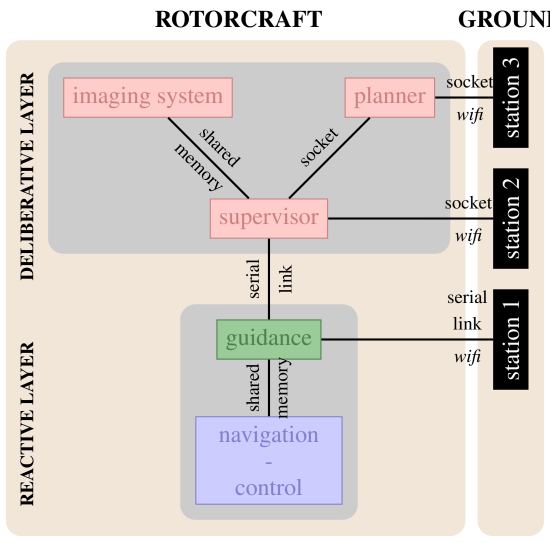

Figure 1: Embedded architecture: the mission is supervised by the supervisor, which is a client of the imaging system and the planner (servers activated on request)

Reactive layer: flight control architecture

The flight control architecture enables the ReSSAC rotorcraft to fulfill any mission described in terms of:

- flight plans associated with procedures (phases and flight states transition to comply to): flight plans are defined as an ordered list of "enriched waypoints", each waypoint being defined by its 3D location (x,y,z) in (GPS) earth reference coordinates, the desired velocity vector Vc and the desired heading of the aircraft at this point.
- or reactive action strategies, conditional to external events or perception. The SNAKE flight guidance strategy of the aircraft enables it to deal with forbidden flight zones and avoid unexpected local, even moving, obstacles.

A conditional reactive "decking" strategy was also developed and implemented for ship decking experiments, and flight tested using GPS-IMU measurements on both the aircraft and a moving pointing device simulating the motion (under swell) of a virtual ship deck in the air. The flight control architecture does not manage the mission reconfigurations such as a change in the flight plan.

Deliberative layer

In our deliberative layer we distinguish two main functions of decision making on one hand and perception on the other hand. The techniques applied for replanning and decision making are described in details in the following, but prior to this, we very briefly describe principle of the vision based perception capabilities of the aircraft.

Perception is used in order to obtain information on the presence or absence of obstacle on the terrain, and if possible update an elevation map. A camera is mounted on the ReSSAC rotorcraft in "nadir" orientation, i.e. it is pointing downward, vertically when the rotorcraft's attitude is calm and horizontal. In such a configuration, the estimated distance, if obtained, corresponds to the altitude of the camera relative to the obstacles on the ground. The terrain map is based on an "a priori" numerical terrain model. This terrain model is bound to be embedded, which means that we are using a very coarse model today: triangulation with 40 meters long edges on average, with corresponding elevation errors in the middle of the triangles.

The vision algorithms on board the ReSSAC aircraft are used in order to build an elevation map using stereovision from motion. The basic principle of "stereovision from motion" is that the distance of observed features in the images is estimated on the basis of the way they move from an image to the next one in the video sequence. Pairs of images are correlated, in order to match a point in the first image with the corresponding point in the following image : the distance of that point from the camera is obtained by triangulation.

Choice of a supervision-centered architecture

Three main kinds of robotics architecture can be distinguished: modular architectures, planner-centered architectures, supervision-centered-architectures.

Modular architectures, like in (Muscettola *et al.* 2002; Verfaillie & Lemaître 2007), are composed of interconnected modules, which share a common abstract interface. Whereas such architectures define clean and clear connexions between functions, they do not allow to segregate the flight control functions (*functional level*) implemented in the reactive layer) and the higher level functions, including planning and image processing (*decisional level*) implemented in the deliberative layer). In such architectures, the separation between the functional level and the decisional level, required by safety authorities, is not clear. In (Fleury, Herrb, & Chatila 1994), a modular architecture is used only at the functional level.

In planning-centered architectures, like (Alami *et al.* 1998; Muscettola *et al.* 1998), the planner is at the top of the architecture. Such an architectural choice assumes that all high-level tasks are maintained by the planner, even if some of them do not need to be planned at all. In the mission of our autonomous rotorcraft, most of the tasks are *static procedures*: they are part of the operating rules of the aircraft, so that they are directly encoded in the supervisor's state machines. For instance, at the beginning of its mission, the rotorcraft has to takeoff. This task does not need to be planned because it is always part of the aircraft mission. On the contrary, the choice of the order in which the selected zones of interest must be further explored, in order to eventually land on it, is a complex deliberative process involving *dynamically adapted* tasks: it requires to optimize a compromise between the remaining flight autonomy, the probability to land on a zone after having explored it, and the probability to rescue the human when landing on a zone. Planning-centered architectures do not clearly distinguish static procedures from dynamically adapted and replanned tasks.

For this reason, our architecture is centered on the supervision function. The supervisor is the central function of our embedded architecture. It has four roles:

- encoding the scenario with finite state machines;
- coordinating image processing and planning;
- translating high-level tasks (go to a zone, explore it, land, takeoff) into low-level ones (move to a 3D point);
- sending the low-level translated tasks to the platform.

(Barrouil & Lemaire 1999) proposed a supervision-centered autonomous underwater vehicle architecture, where all the components are centered on the supervisor, even the basic motion control functions. One disadvantage of this architectural choice is that all basic functions must communicate via the supervisor, which is not only needless, but may also be contradictory with the respect of real-time constraints for the reactive layer functions. On autonomous aircraft, we need to clearly segregate the functional level (reactive layer) and the decisional level (deliberative layer), the latter being centered on the supervisor.

Planning as independent processes

The planner is a server that is activated on demand by the supervisor. The planner optimizes the solution to the submitted problem, and sends it back to the supervisor through the socket. The dialog between the supervisor and the planner is fitted to a multi-thread stochastic planning approach, presented in the next section.

Encapsulating the planning processes into servers has two advantages. On one hand, it simplifies the decisional functions by removing from the planner the tasks which do not need to be planned. On the other hand, it allows to solve several planning problems in parallel, without increasing the complexity of the architecture. For instance, the exploration of a search region may require its division into smaller search zones, with corresponding tasks decomposition performed by the planner. A subsequent planning problem can be submitted to the planner in parallel through a new socket as soon as the first zones are generated, in order to optimize (and start with) their exploration.

An anytime multi-thread planning system

Uncertainties and plan representation

In artificial intelligence planning, two approaches emerged to deal with environment's uncertainties. The first approach is reactive and is based on so called *classical planning* (Lemai & Ingrand 2004; Damiani, Verfaillie, & Charmeau 2005). It supposes that actions are locally deterministic: it locally produces *plans*, i.e. finite sequences of actions that lead from an initial state to a goal state. If an action leads to an unexpected state, the planner launches a new replanning process, starting at the new current state. This approach is very efficient because the situations explored to find a solution, are limited by the deterministic hypothesis of actions' effects. Nevertheless, the plan is not robust to uncertainties, in the sense that it does not take into account the consequences of actions failures. For instance, if the flight autonomy decreases more than it was foreseen during the planning process, the mission could fail even after several replannings. Within a supervision-centered architecture, the deterministic planner sends the entire plan to the supervisor, which is in charge to execute it and to eventually launch a replanning process.

The second approach relies on *stochastic planning*, which takes into account the environment's uncertainties in the planning process. It produces a *policy* or *conditional plan*, a mapping from every possible states to an optimal action to perform. A popular framework to construct policies is based on Markov Decision Processes (MDPs): it optimizes the average of cumulated rewards (or penalties) along the mission probabilistic trajectories (Puterman 1994). As a result, the policy is robust to environment's uncertainties, since its optimization anticipates the action failures. On the other hand, complete optimization of MDPs are often intractable for realistic problems, because it explores all possible situations and is also exponential in the number of state's features (Boutilier, Dean, & Hanks 1999). Therefore, local heuristic search methods have been recently proposed to reduce the number of situations to explore (Aberdeen & Buffet 2007; Teichteil & Fabiani 2005; Feng, Hansen, & Zilberstein 2003), as we will see in the following.

We do not know about any other real autonomous systems embedding a MDP-like planning system, but ReSSAC. We have devoted special efforts on the way to execute policies produced by MDPs. Even with heuristic local search methods, policies are far bigger in space than (deterministic) plans. Contrary to deterministic approaches, we could not send the entire policy through the socket between the planner and the supervisor. However, policies are mapping from states to actions: we only send the current state through the socket in XML format, then the planner sends back the action to perform, read from the policy (see Figure 2). If the current state is not reachable, the planner locally reoptimizes the MDP from the new current state.

For effectiveness reasons, the data and the policy of the planner are encoded as Algebraic Decision Diagrams (ADDs). They represent numeric data (probabilistic effects of actions, optimized value of the policy) mapped on logic data (feature-based state space and action space) in a very compact encoding (R.I. Bahar *et al.* 1993; Hoey *et al.* 1999). As shown in Figure 2, XML states and XML actions are translated into ADDs between the planner communication interface and the policy.

Multi-thread anytime stochastic planning

Optimization of MDPs is a long and consuming process because of its polynomial complexity in the number of state space (exponential in the number of state features). Real autonomous rotorcraft missions are strongly constrained by the flight time, so that it is impossible to wait for the planner to optimize the complete mission. Nevertheless, as presented in the next subsection, recent stochastic planning approaches allow to cut the complete optimization process in small incremental optimization units (Teichteil & Fabiani 2005; Feng & Hansen 2002). An applicable policy is available

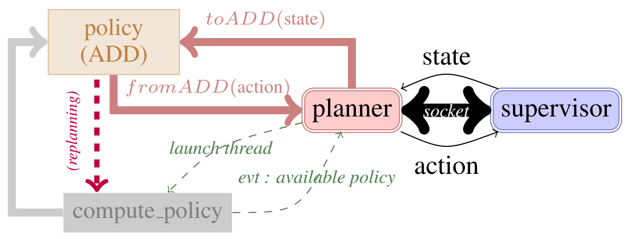

Figure 2: Anytime multi-thread planning: the planner optimizes the MDP within a background task. It is questioned by the supervisor as soon as a feasible policy is available.

between each optimization unit. Therefore, we can easily turn these incremental methods into anytime stochastic planning approaches, thanks to multi-threading programming: one thread interacts with the supervisor (including XML to ADD translation and policy reading), while another thread optimizes the MDP in background. Both threads are synchronized on the availability of the thread-protected policy between each optimization unit (see Figure 2).

Nevertheless, the first optimization unit of such incremental stochastic planning algorithms is still a bit too long for time-constrained planning systems, whereas the other units are less and less longer. For that reason, we propose to firstly compute an applicable policy such that there exists at least one trajectory leading from the initial state to goal states, but without any optimization. This policy can be computed with a kind of logical version of dynamic programming:

- $\pi_0(s) = \begin{cases} \{\text{actions defined in goal states}\} & \text{if } s \in \text{goal} \\ \emptyset & \text{otherwise} \end{cases}$
- $\pi_{t+1}(s) = \{a : \exists\, s' \ ,\ T(s' \mid a, s) > 0 \text{ and } \pi_t(s') \neq \emptyset\}$

where $\pi_t(s)$ is the set of actions a that can be randomly chosen at step t in state s (eventually empty), and $T(s' \mid s, a)$ is the markovian state stochastic transitions from s to s' by applying action a. The dynamic programming process stops when there exists a step t such that: $\pi_t(\text{initial state}) \neq \emptyset$. This first policy computation unit only involves logical tests, so that it allows to obtain very quickly a first applicable action in the initial state.

Incremental stochastic dynamic programming

Many heuristic local search algorithms were developed to incrementally optimize an MDP, like sLAO* (Feng & Hansen 2002) or sfDP (Teichteil & Fabiani 2005). All these approaches alternate two stages in each optimization unit. The first stage is a computation of the set of states that are reachable by iteratively applying the current policy starting from the initial state. The second stage is a local optimization of the policy within the set of current reachable states.

These approaches differ in the expansion of the reachable states subspace. In sfDP, the expansion stage is controlled by the knowledge of the goal states (see Figure 3), expressed as a constraint linking state features:

$$at(base) \wedge (human\_rescued \vee (flight\_autonomy \leqslant 10\ mn))$$

Moreover, the initial policy computation, proposed in the previous subsection, requires the knowledge of goal states too. As far as we known, among incremental stochastic planning approaches, only sfDP uses goal states to control the states subspace expansion. Therefore, we implemented sfDP on board our autonomous rotorcrafts, augmented by the initial stochastic policy computation.

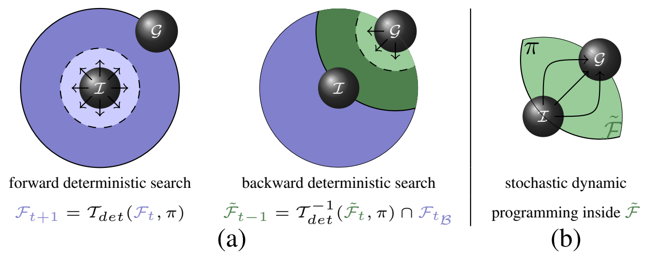
$\mathcal{G}$
$\mathcal{I}$
forward deterministic search
$\mathcal{F}_{t+1} = \mathcal{T}_{det}(\mathcal{F}_t, \pi)$
backward deterministic search
$\tilde{\mathcal{F}}_{t-1} = \mathcal{T}_{det}^{-1}(\tilde{\mathcal{F}}_t, \pi) \cap \mathcal{F}_{t_\mathcal{B}}$
(a)
π
$\tilde{\mathcal{F}}$
stochastic dynamic
programming inside $\tilde{\mathcal{F}}$
(b)

Figure 3: sfDP : (a) expansion of the reachable states subspace $\mathcal{F}$ by successively applying the current policy π from the initial states $\mathcal{I}$ to the goal states $\mathcal{G}$ (b) optimization of π. ($\mathcal{T}_{det}$: deterministic-made transitions)

Launching local replanning processes

The advantage of sfDP over other incremental methods is the knowledge of goal states, that reduces the number of states to explore during the local optimization stage. On the other hand, its drawback is the loss of optimality guarantee because not all states reachable from the initial state are explored, but only those leading to goal states. In particular, there is no guarantee that all trajectories of the current policy lead to the goal states, starting at the initial state: the states subspace expansion stops indeed as soon as at least one of these trajectory is found. As a result, the planner needs to launch replanning processes in two cases (see Figure 4):

(a) a low-probability state transition occurs during the policy execution, so that the new current state is outside the reachable states subspace ;

(b) no transitions are defined in the current state, when all stochastic outcomes are outside the reachable subspace.

The replanning time generally decreases at each replanning, since the new initial states of each replanning process are nearer and nearer from the goal states.

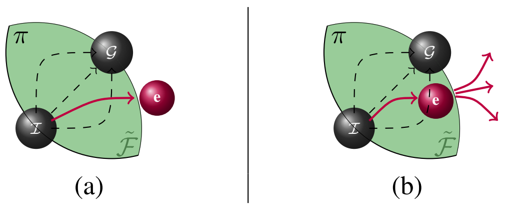

Figure 4: replanning cases: (a) the current state is outside the current reachable subspace (b) the local policy is not defined in the current state

Flight tests and return on experience

We realized real flight tests at Esperce in Southern France. The on-board planner implemented the multi-thread version of sfDP with the initial stochastic policy computation. In this section, we compare the performance of our anytime multi-thread framework with single-thread approaches like sfDP. We focus on four comparison criteria:

- *total optimization time*: sum of optimization times of planning and replanning processes;
- *number of replannings*;
- *maximum answering time*: maximum time to send an action to the supervisor after receiving of the current state;
- *landing zone*, obtained by applying the current policy from the rotorcraft's base.

In order to compare these criteria on the same basis, we recorded on disk a list of 10 zones that were extracted after image processing during a real flight. The planning problem contains 24 state variables: 1 binary variable `human-rescued`, 1 binary variable `on-ground`, 10 binary variables `explored(zone)`, 10 binary variables `landable(zone)`, 1 11-ary variable `at(zone)` or `at(base)`, and 1 288-ary variable `flight-autonomy` (1 hour divided into 288 intervals of 12.5 seconds). The size of the state space is also: $2 \times 2 \times 2^{10} \times 2^{10} \times 11 \times 288 = 13\,287\,555\,072$ states.

Figure 5 represents the comparison between the single-thread version of sfDP and its multi-thread version. All actions were simulated including their durations in order to test background optimization (in multi-thread mode) and replanning processes. Actions last 50 seconds on average. All tests were run on the real embedded processor dedicated to deliberative processes (1 Ghz Pentium). The last column reproduces the results that we obtained during the real flight. The second column corresponds to an optimal algorithm based on value iteration over the entire state space. It is proved for many years that such optimal algorithms are not efficient at all: we tested it to give an idea of the complexity of the planning problem, and to compare the landing zone that is computed by sfDP and an optimal algorithm (both landing zones are indeed identical). The optimal algorithm could not compute a policy after 1 hour, i.e. the maximum mission's duration, with more than 5 zones.

The aim of our tests is not to assess the efficiency of local heuristic stochastic planning algorithms like sfDP (see (Teichteil & Fabiani 2005; Feng, Hansen, & Zilberstein 2003) for such tests). We want to demonstrate that our multi-thread anytime approach produces applicable and non stupid policies in a very short time. Figure 5 shows that the maximum answering time of the multi-thread version of sfDP is negligible in comparison with actions' average duration (~ 50 s). The maximum answering time of the single-thread version is significantly bigger. With 10 zones, in single-thread mode, the current state is nearly never included in the reachable subspace of successive replannings, because the state space feature `flight_autonomy` decreases as much as the replanning time. As a result, a lot of replannings are necessary to apply a single action, what does not occur with our multi-thread anytime framework.

nb of zones	5	5	5	7	7	10	10
algorithm	optimal	ST	MT	ST	MT	ST	MT
total optim. time	1358	2.58	2.8	13.76	13.6	308.29	258.57
nb replannings	0	0	1	0	1	3	4
max. answer. time	1358	2.58	**0.21**	13.76	**0.29**	308.22	**5.75**
landing zone	Z1	Z1	Z0	Z5	Z0	Z0	Z1

Figure 5: Comparison between single-thread sfDP (ST) and multi-thread sfDP (MT) – time is given in seconds

Conclusion

We proposed a multi-thread decisional architecture for real-time planning under uncertainty. We don't known of any other MDP-like algorithms implemented on-board real autonomous aerial vehicles. Yet, MDP optimization algorithms can not be used "as is" to compute on-line time-constrained strategies: their computation time exceeds the acceptable answering time limit between the planner and the sensors. We propose to embed them in an anytime framework for solving real-time stochastic planning problems. The planner process is divided into two communicating threads synchronized on the current applicable policy. We compute the first stochastic policy such that there exists at least one trajectory leading from the initial state to goal states. Our tests on the real rotorcraft embedded processor show the effectiveness of our approach: not only the answering time is significantly reduced, but optimized policies can now be applied in time compatible with actions' duration. Another way to produce anytime strategies could consist in splitting an incremental optimization algorithm into small *independent* computation processes, successively run within a single thread, but this implementation might be painful in terms of data exchange between the computation processes.

Acknowledgment

This work would not have been possible without the great and essential work of the ReSSAC team project: V. Fuertes, G. Le Besnerais, R. Mampey and A. Piquereau.

References

Aberdeen, D., and Buffet, O. 2007. Temporal probabilistic planning with policy-gradients. In *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling*.

Alami, R.; Chatila, R.; Fleury, S.; Ghallab, M.; and Ingrand, F. 1998. An architecture for autonomy. *International Journal of Robotics Research* 17(4):315–337.

Barrouil, C., and Lemaire, J. 1999. Advanced real-time mission management for an AUV. In *Proceedings of the SCI NATO Symposium on Advanced Mission Management and System Integration Technologies of Improved Tactical Operations*.

Boutilier, C.; Dean, T.; and Hanks, S. 1999. Decision-theoretic planning: Structural assumptions and computational leverage. *J.A.I.R.* 11:1–94.

Damiani, S.; Verfaillie, G.; and Charmeau, M.-C. 2005. A Continuous Anytime Planning Module for an Autonomous Earth Watching Satellite. In *ICAPS05 Workshop on Planning and Scheduling for Autonomous Systems*, 19–28.

Fabiani, P.; Fuertes, V.; Besnerais, G. L.; Mampey, R.; Piquereau, A.; and Teichteil, F. 2007. The ReSSAC autonomous helicopter: Flying in a non-cooperative uncertain world with embedded vision and decision making. In *A.H.S. Forum*.

Feng, Z., and Hansen, E. 2002. Symbolic heuristic search for factored markov decision processes. In *Proceedings 18th AAAI*, 455–460.

Feng, Z.; Hansen, E.; and Zilberstein, S. 2003. Symbolic generalization for on-line planning. In *Proceedings of the 19th Annual Conference on Uncertainty in Artificial Intelligence*. Morgan Kaufmann.

Fleury, S.; Herrb, M.; and Chatila, R. 1994. Design of a modular architecture for autonomous robot. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 3508–3513.

Hoey, J.; St-Aubin, R.; Hu, A.; and Boutilier, C. 1999. SPUDD: Stochastic planning using decision diagrams. In *Fifteenth Conference on Uncertainty in Artificial Intelligence*, 279–288.

Lemai, S., and Ingrand, F. 2004. Interleaving temporal planning and execution in robotics domains. In *Proceedings of the National Conference on Artificial Intelligence*.

Muscettola, N.; Nayak, P. P.; Pell, B.; and Williams, B. C. 1998. Remote agent: to boldly go where no ai system has gone before. *Artificial Intelligence* 103(1-2):5–47.

Muscettola, N.; Dorais, G.; Fry, C.; Levinson, R.; and Plaunt, C. 2002. IDEA: Planning at the core of autonomous reactive agents. In *Proceedings of the 3rd International NASA Workshop on Planning and Scheduling for Space*.

Puterman, M. L. 1994. *Markov Decision Processes*. John Wiley & Sons, INC.

R.I. Bahar; E.A. Frohm; C.M. Gaona; G.D. Hachtel; E. Macii; A. Pardo; and F. Somenzi. 1993. Algebraic Decision Diagrams and Their Applications. In *IEEE /ACM International Conference on CAD*, 188–191.

Teichteil, F., and Fabiani, P. 2005. Symbolic heuristic policy iteration algorithms for structured decision-theoretic exploration problems. In *ICAPS International Workshop on Planning under Uncertainty for Autonomous Systems*.

Verfaillie, G., and Lemaître, M. 2007. A generic modular architectural framework for the closed-loop control of a system. In *Proceedings of the 2nd National Workshop on Control Architectures of Robots*, 19–31.