IJCAI Workshop *Reasoning with Uncertainty in Robotics*, Edinburgh, Scotland, 30 July 2005

Symbolic Probabilistic-Conditional Plans Execution by a Mobile Robot

Abdelbaki BOUGUERRA and Lars KARLSSON

Department of Technology
Örebro University, SE-70182
Örebro; Sweden
http://aass.oru.se

Abstract

In this paper we report on the integration of a high-level plan executor with a behavior-based architecture. The executor is designed to execute plans that solve problems in partially observable domains. We discuss the different modules of the overall architecture and how we made the different modules interact using a shared representation. We also give a detailed description of the hierarchical architecture of the executor and how execution-time failures are handled.

1 Introduction

Carrying tasks through in real world environments presents a multitude of challenges to contemporary mobile robots. Most importantly is how to deal with the uncertainty inherent in on-board sensing, acting and lack of information. Acknowledging that classical planning, i.e. planning that assumes that the environment is static and the robot is omniscient, is not the best choice for reasoning in such environments, research has focused on developing planning approaches capable of reasoning under uncertainty and partial observability [6][2][8]. However, most developed planning approaches forget about plan execution and monitoring. Even with plans that incorporate contingencies of execution, there is still an uncountable number of unexpected events that might prevent executing the plan reliably or even invalidate it.

In this paper we report on the successful implementation of a high-level plan executor on top of a behavior-based mobile-robot control architecture. The executor is designed to handle probabilistic conditional plans and is not constrained by using a specific planner. In fact the executor can execute any plans fulfilling some representational constraints, mainly the syntax of the plans, and the specification of how to execute the plans actions. The execution system uses three hierarchical layers each with a specialized process. The top layer manages high level plans, user requests, and recovery when necessary. The middle layer has a more specialized process whose task is to execute the actions of the plan selected by the upper layer, whereas the third layer is responsible of low-level execution and monitoring. The overall system has been successfully used for research on sensor-based planning for mobile robots, most notably in the areas of perceptual anchoring [5] and active smelling [11]. One of the main strengths of our execution framework is the ability to act in partially observable domains as a result of using reasoning with symbolic planning under uncertainty. POMDP:s are by far the most used paradigm for decision making in partially observable domains, however the kind of symbolic planning that we use has certain advantages over using POMDP:s. First, symbolic planners are much faster then POMDP solving algorithms which makes replanning possible at execution-time, that is not the case with POMDP:s because policies have to be found to cover an exponential number of continuous belief states. Second, at execution time, no belief-state tracking is required, since belief states used by symbolic planners are distinguishable by their set of observations as opposed to implemented mobile-robot architectures executing POMDP policies which requires updating belief states online [15] [13], although in theory it is possible to approximate policies using finite state machines [7][12]. For an overview of planning under uncertainty including decision-theoretic planning and symbolic planning, the reader is referred to [3]. A survey of approximate methods used to solve POMDP:s can be found in [1].

One central issue that we address is how to respond to unexpected events at execution time. Our framework provides some degree of flexibility regarding failure recovery: with each low-level executable action, a very quick recovery strategy can be specified. If low-level execution is not recoverable, a more high-level deliberate recovery is launched to cope with the unexpected situation.

2 Architecture Overview

The plan executor and high-level planning were added as a deliberation layer on the top of the ThinkingCap behavior-based control architecture [14]. Figure 1 gives an overview of the overall architecture and how both layers are integrated. In the following subsections we briefly outline the different entities forming the complete system.

2.1 Behavior-based Architecture

The ThinkingCap (TC) robot-control architecture is composed of a fuzzy-logic controller and a navigational planner called B-Planner. TC controls the mobile robot using fuzzy behaviors expressed as sets of control rules. To generate navigation plans, B-Planner (for Behavior Planner) computes a set of context-behavior rules having the form `IF context THEN behavior`, where `context` is a formula of fuzzy predicates evaluated on the current world model. The context-behavior rules of a B-Plan are evaluated in parallel, influencing the overall robot behavior according to the blended value of their respective context.

TC uses a local perceptual space LPS to store information about the world around the robot expressed as object descriptors and perceptual data. The LPS provides data to the self-localization module used to compute the robot position in the different sectors of the map. The controller produces crisp control values (steering and velocity) through defuzzification of the result of the blended active fuzzy behaviors (according to their context calculated from the LPS).

2.2 High-level Planning

The kind of symbolic planning used within the overall system solves problems in partially observable domains i.e. observations reveal only part of the real state. Both planners PTLPLAN [9] and C-SHOP [4] that we used in our experiments use the same formalism for actions and world model. The action model makes it possible to reason about conditional non-deterministic effects with probability distribution over them. The description of effects might also include making observations. To represent uncertainty about the state of the world belief states are used. Both Planners take an initial belief state and a goal formula and return a plan with a certain probability of success.

2.3 Anchoring and Perception

The anchoring module provides an interface to perceptual information from the sensor systems of the mobile robot. It contains a number of functionalities for establishing the connection between the high-level symbolic representation and low-level perceptual representations such as video camera images. Typically, executing an action such as "(move-near gasbottle1)" requires the identification of what perceptual data correspond to the symbol "gasbottle1". On the symbolic level "gasbottle1" should have a symbolic description such as (shape gasbottle1 = bottle), (color gasbottle1 = red), which is matched against the available perceptual data. If a matching percept is found, the symbol is anchored to that percept. It sometimes happens that the anchoring module fails to find a specific object, either because no match is found, or there are several but partial matches. This is one important class of situations where recovery is needed [5]. The anchoring module can also provide information about already perceived objects, such as position and visual features extracted from the LPS.

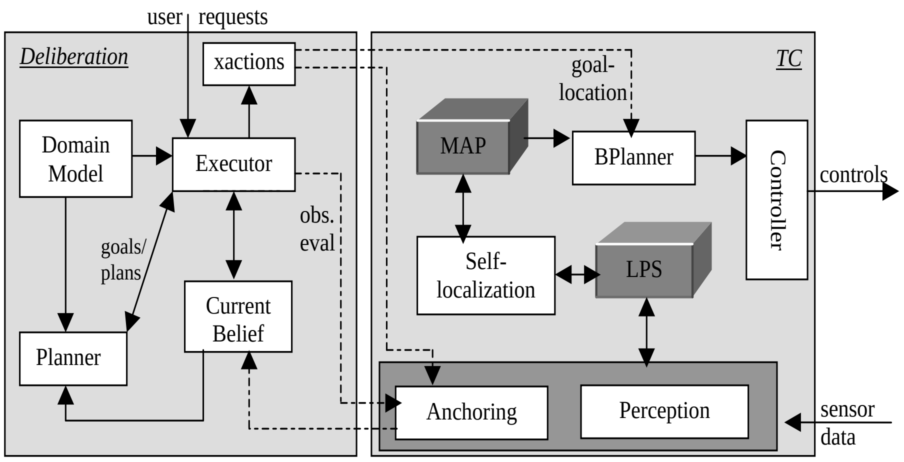

Figure 1: Global architecture

3 Shared Representation

Making the planner, the plan executor, and TC work together implies representing the different entities at the borderline between the different layers with a notation that all of them understand. More specifically is how to represent the robot's belief about the world, syntax and semantics of plans, the process of making observations and how plan actions should be executed by TC.

3.1 Belief States

The high-level plan executor is designed to execute and monitor plans that solve partially observable problems. Belief states are used to model partial observability about the state of the environment. The representation of belief states we use is similar to the one used by many symbolic planners such as C-SHOP [4], PTLPLAN [9], and C-BURIDAN [6], where a belief state is a probability distribution over elementary states that share the same set of observation fluents. For instance, the following belief state represents that, after making the observation of noticing a red light inside a room, the robot is either in room r_1 with 70% or in room r_2 with 30%.

$$\texttt{belief} = \langle \textsf{obs} = \{\textit{redlight}\}\,; \{0.3 : \texttt{(robot-in } r_2\texttt{)}; 0.7 : \texttt{(robot-in } r_1\texttt{)}\}\rangle$$

To be able to evaluate observation fluents at execution-time, a procedure must be defined and associated with each observation fluent. When executing a plan, the executor uses the observation fluent to determine the evaluation procedure associated with it. The procedure specifies how to make the observation by calling TC's fuzzy observation predicates such as `(open d)` that refers to the degree to which the door `d` is open. The procedure might also use the data stored in the LPS to evaluate observation fluents not computed by TC such as `(near obj1 obj2)` for evaluating whether `obj1` is near `obj2` based on metric data in the LPS about both objects.

3.2 Conditional Plans

The symbolic planning system is required to generate plans with following syntax that the executor adopts for plan representation:

plan ::= `(` *action** *end-step* `)`
end-step ::= `:success` | `:fail` | `(cond` *branch** `)`
branch ::= `(`*obs-cond* *action** *end-step* `)`
action ::= `(`*action-name* *term**`)`
obs-cond ::= *fluent-literal* | `(and` *fluent-literal**`)`

action represents an instantiated domain action, *obs-cond* is a conjunction of fluent-value formulae defined over observations, and `:success`, `:fail` are used to denote predicted plan success and failure respectively. This grammar accepts plans that have the structure of a tree where nodes with one successor represent actions, and nodes with more than successor represent a conditional branching. The following plan is an example of plans accepted by the executor:

```
((go-near $d_1$)(check $d_1$)(cond (open $d_1$) :  (enter $r_1$) :success)
                                   (not (open $d_1$)) :  :fail))
```

Besides the syntax, the executor and the planner must agree on the semantics of the conditional plans. In our case, the semantics of a conditional plan can be interpreted as applying the first action of the plan in the initial belief state. The second action is applied in the resulting belief state of the first action, and repeatedly applying an action in the resulting belief state of its preceding action. If the application of an action results in more than one belief state, then the subsequent action must be a conditional plan `(cond` $(c_1\ p_1) \ldots (c_m\ p_m)$`)` with as many branches as resulting belief states. Each branch $(c_i\ p_i)$ represents a contingency plan p_i to be executed in the belief state whose observations satisfy the branch condition c_i. As mentioned in the introduction, since belief states at a certain execution time are uniquely identified by their observations, execution-time belief update is no longer required. Only the the observations fluents are evaluated and the branch that has its observation fluent formula verified in the real world is selected for subsequent execution. Of course at execution time, there must be only one belief state whose observations are verified in the real world.

3.3 Executable Actions

To be able to execute the actions of a high-level plan, the domain creator must specify with each high-level action the different executable actions `xactions` in terms of the robot control-architecture (in this case TC) functionalities. Typically, an executable action defines a procedure that calls TC's functions to produce behaviors that would achieve a specific low-level goal. The procedure also defines the monitoring process to be associated with the execution of the behaviors in order to make sure to respond to unexpected events and apply local recovery strategies if possible. The monitoring process must also give an indication of whether the `xaction` has been executed successfully or with failure, so that a deliberate recovery would be considered for the high-level action.

Example 1 In order to execute the high-level action `(enter` r_1`)` to enter room r_1, the execution part may consist of a procedure "`execute-enter (room)`" that 1) calls the B-Planner with the goal `(robot-in` r_1`)`, where the goal represents a fuzzy predicate, and 2) installs a monitor process for the generated B-Plan. In case of failure to achieve the b-plan goal, the monitor may call the B-Planner to replan for another navigation path to enter room r_1.

4 High-Level Execution

In this section we outline the main processes involved in executing a high-level conditional plan and its actions. The executor uses different data structures to manage the execution of multiple plans that can arrive asynchronously. After having generated a plan, the planner creates an execution context that includes the initial belief state, the goal, the plan itself, the last action executed, and the priority of the plan. The execution context is then placed in one of three queues waiting for execution. The queues are associated with classes of plans identified by their priorities. A plan can have either low, medium, or high priority.

Plan execution is performed hierarchically. At the top-level, there is a process responsible for selecting the plan with the highest priority for execution. It is also responsible of launching the recovery of plans when one of their actions fails to execute. At the second level, a more specialized process is used to control the execution of the actions of conditional plans, reporting the outcome of the action to the high-level process. The action execution process is mainly responsible of extracting the executable actions of the current plan action considered for execution, and launching the appropriate processes to achieve the executable actions and monitor their progress, see Fig 2.

4.1 Plan Execution Process

The plan-execution process is launched upon starting up the robot. While in state INIT, the process checks periodically for waiting plans, proceeding with the execution of the plan with the highest priority. The actual execution of a plan starts in state NEXT-ACTION, where the executor checks the type of the current action selected for execution. An action can be `:success`, `:fail`, a conditional plan, or a domain action. It is also in this state where plans with higher priority can interrupt the execution of the plan in execution. As mentioned earlier, the two special actions `:success` and `:fail` reflect predicted success and failure by the planner of the plan, respectively. If the plan reaches a predicted failure, then the process simply drops the plan. Reaching a predicted success state means that the plan has achieved its goals with success. Because the currently dropped or succeeded plan might have interrupted the execution of another plan with lower priority, the execution process checks subsequently whether there is an interrupted plan waiting for execution in order to restore its execution context and start it again (state RESUME).

One issue in restarting the execution of an interrupted plan that might arise is the possibility not being able to start the execution of the interrupted plan, because its action to resume is not applicable in the the current real world situation resulting from the execution of an interrupting plan. To remedy partly to this problem, the process does not interrupt an executing plan unless it can find a chaining plan that ensures that the interrupted plan can be resumed when the interrupting plan finishes execution with success. It is worth noting that finding a chaining plan might be problematic since the interrupting plan can have more than one branch that leads to success. Generating the chaining plan would take into account this issue which results in a chaining plan with a possible branch for each possible belief state where the goals of the interrupted plan are satisfied. Upon resuming the execution of an interrupted plan, the process executes the chaining plan first, and then the rest of the interrupted plan. Obviously, this works only when the interrupting plan has successfully been executed, in case of failure, the process has to launch the planner to find a chaining plan that reaches the preconditions of the first action of the rest of the interrupted plan starting from the current situation.

If the current action is a conditional plan, the process checks the contingency condition for every branch in the real world, and chooses the branch whose condition is verified in the real world observations. Checking the branching conditions is done through calls to the procedures associated with the observation fluents that evaluate the condition fluents using the perceptual data provided by the anchoring module. If, on the other hand, the current action is an instantiated domain action, the process checks its preconditions in the current belief state. In case the preconditions are satisfied, another process is launched to execute the action as described in the next subsection. In case of discrepancy, the process calls special functions that build the current belief state so that more information is included about the current situation and then calling the planner to find a plan that achieves the preconditions of the failing action (state RECOVERY).

4.2 Action Execution Process

The execution of high-level actions is performed by a more specialized process whose states are outlined in Fig 2. Activating action execution at this level involves blocking the launching process i.e. the plan-execution process. High-level action execution starts by retrieving the `xactions` one at a time (state `xaction`). As we outlined before, there are specialized procedures defined with each `xaction` specifying the necessary steps to perform along with a monitoring process. Calling the specialized procedure of an `xaction`, results in blocking the action-execution process and launching the monitoring process of the `xaction`.

The monitoring process of an `xaction` can respond to failures by calling precomputed procedures or by calling the B-Planner to find another local B-plan. The implementation of an `xaction` monitoring process has also to to guarantee that the blocked action-execution process is notified about the outcome of the execution of the `xaction`. If the execution of the `xaction` is successful, then the action-execution process is awaken in the state XOK, otherwise it is awaken in the state XFAIL.

Awaking the action-execution process in state XFAIL is an indication of the inability of the robot to execute the `xaction` with success which leads to the failure of the high-level action. Therefore the plan-execution process is notified in turn that the execution of the current action has failed (state DISCREPANCY/FAIL). If, on the other hand, the monitor of the `xaction` reports to have executed `xaction` successfully, the action-execution process repeats the same steps with the remaining `xactions`. When all the `xactions` have been successfully executed (state SUCCESS), the action-execution process awakes the plan-execution process in the state NEXT-ACTION, so that the same steps can be performed with the next high-level action of the plan.

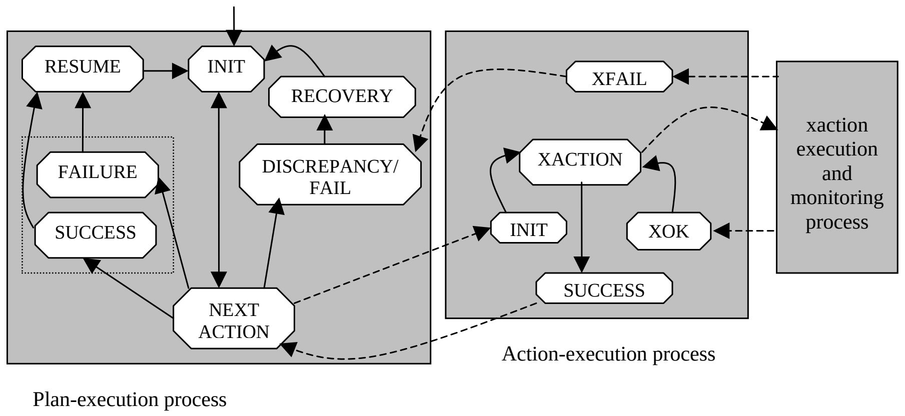

Figure 2: The different processes of the plan executor

5 Experiments

The system reported in this paper has been used on two Magellan Pro mobile robots for a number of experiments, in particular relating to the problems of anchoring with ambiguities, and the use of an electronic nose on a mobile robot. In the following, we briefly present some of those experiments. Although they were designed to test other specific capabilities of our system, they also serve well to illustrate what the executor can achieve.

The visual anchoring experiments, some of which were reported in [5], consisted of recovery planning in situations where the robot was supposed to find and approach an object but could not immediately determine which of several objects was the correct one. One series of experiments involved a number of gas bottles. The robot had the task to approach the marked gas bottle but the mark was not visible from the robot's current position. This resulted in a recovery plan where the robot inspected the different gas bottles from different angles and eventually detected the correct one. The success rates for these experiments were 87% of 45 runs, with setups of 2, 3 or 4 gas bottles. The maximal time required for the recovery planning was below 1.5 s for these experiments. More recent experiments involved using relational information to find objects (" the green can near to the red ball with a mark") with similar levels of success (80–93%) and performing recoveries from a sequence of problems occurring when the robot was to approach several different objects located in different corridors.

A series of e-nose experiments [11] involved the e-nose as a complementary sensor modality. The experiments were performed by using a number of cups containing different substances and in order to find the correct cup, the robot first needed to visually search for candidate objects and then use the electronic nose to discriminate between these candidates (success rates 82 – 76% for 2–5 candidates). In another series of experiments [10] the robot patrolled a number of corridors, traveling in total more than 1.2 km, while maintaining and updating information about cans it encountered. Cans were sometimes added, removed or displaced. In addition, the robot was occasionally given odour samples and had to find cans with similar odours.

6 Conclusion

We have presented a hierarchical system to execute probabilistic conditional plans that solve problems in partially observable domains. One major advantage of using hierarchy of processes is the ability to reason about failure and recovery at different levels of detail and complexity. We also demonstrated how we integrated the proposed execution system on top of a behavior-based control architecture. Among the issues that were encountered in the process of integration, was the common representation that should be used to interface the executor and the planning system on one hand, and the executor and the control architecture on the other hand. We tackled this issue by adopting a simple set of rules that do not require too much effort from the planning domain writer.

References

[1] D. Aberdeen. A (revised) survey of approximate methods for solving partially observable markov decision processes. Technical report, National ICT Australia, Canberra, Austalia, 2003.

[2] P. Bertoli, A. Cimatti, M. Roveri, and P. Traverso. Planning in non-deterministic domains under partial observability via symbolic model checking. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence. (IJCAI)*, pages 473–478, 2001.

[3] J. Blythe. An overview of planning under uncertainty. *AI Magazine*, 20.

[4] A. Bouguerra and L. Karlsson. Hierarchical task planning under uncertainty. In *3rd Italian Workshop on Planning and Scheduling (AI*IA 2004)*. Perugia, Italy, 2004.

[5] M. Broxvall, L. Karlsson, and A. Saffiotti. Have another look: On failures and recovery planning in perceptual anchoring. In *Proceedings of the 4th International Cognitive Robotics Workshop (CogRob-2004)*, 2004.

[6] D. Draper, S. Hanks, and D. Weld. Probabilistic planning with information gathering and contingent execution. In *Proceedings of the 2nd International Conference on Artificial Intelligence Planning Systems (AIPS)*, pages 31–63, 1994.

[7] E. Hansen. Solving pomdps by searching in policy space. In *Proceedings of the 14th Annual Conference on Uncertainty in Artificial Intelligence (UAI-98)*, pages 211–219, San Francisco, CA, 1998. Morgan Kaufmann Publishers.

[8] L. Kaelbling, M. Littman, and A. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101:99–134, 1998.

[9] L. Karlsson. Conditional progressive planning under uncertainty. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 431–438, 2001.

[10] A. Loutfi and S. Coradeschi. Maintaining coherent perceptual information using anchoring. In *Nineteenth International Joint Conference on Artificial Intelligence (IJCAI05, to appear)*, 2005.

[11] A. Loutfi, S. Coradeschi, L. Karlsson, and M. Broxvall. Putting olfaction into action: Using an electronic nose on an multi-sensing mobile robot. In *Proceedings of IEEE/RSJ Int. Conference on Intelligent Robots and Systems, IROS04*, 2004.

[12] N. Meuleau, K. E. Kim, L. Kaelbling, and A. Cassandra. Solving pomdps by searching the space of finite policies. In *Proceedings of the 15th Annual Conference on Uncertainty in Artificial Intelligence (UAI-99)*, pages 417–426, San Francisco, CA, 1999. Morgan Kaufmann Publishers.

[13] J. Pineau, N. Roy, and S. Thrun. A hierarchical approach to pomdp planning and execution. In *Workshop on Hierarchy and Memory in Reinforcement Learning (ICML)*, June 2001.

[14] A. Saffiotti. *Autonomous Robot Navigation: a fuzzy logic approach*. PhD thesis, Faculté de Sciences Appliqueés, Université Libre de Bruxelles, 1998.

[15] V. Verma, J. Fernandez, and R. Simmons. Probabilistic models for monitoring and fault diagnosis. In R. Chatila, editor, *The Second IARP and IEEE/RAS Joint Workshop on Technical Challenges for Dependable Robots in Human Environments.* October 2002.