Using a contingency planner within a robot architecture

Alexandre Albore[1], **Eric Beaudry**[2], **Piergiorgio Bertoli**[3] and **Froduald Kabanza**[2]

Abstract. To achieve a given goal, a mobile robot must plan by predicting the possible evolution of the environment and the possible consequences of its actions. The use of actuators and sensors with limited precision and the presence of exogenous agents in the environment leads to nondeterministic predictions. However, most plan-based robotic frameworks ignore this nondeterminism at the planning time, by producing only deterministic plans, and replanning whenever the outcome of actions or the the environment's dynamics stray away from the assumed ones. The main motivation behind this choice has been the longstanding lack of effective planners handling nondeterminism; but recent advances in this area make it possible, and advisable, to exploit such systems to implement more robust plan-based robot behaviors. In this paper, we experiment the integration of a state-of-the art contingency planner in a robotic architecture, and discuss how such an integration improves the degree of flexibility and robustness of plan-based robot behaviors compared to the use of a deterministic planner.

1 Introduction

Humans can easily accomplish complex goal-oriented tasks, very often purely reactively, because of their very sophisticated cognitive, sensing and motion capabilities. In contrast, today robots rely on measurement devices with limited precision to sense their environment (e.g., sonar and laser) and actuators also with limited precision to navigate. This, combined with a limited model of the environment and its dynamics, make it necessary to plan even quite simple tasks that would be achieved otherwise instinctually by humans, such as moving on object from one position to another.

Determining how planned behaviors should be coordinated with purely reactive instinctual behaviors is one of the keys towards the design of artificial intelligent machines, such as robots. In particular, in this paper, we are concerned by the issues of when and how a mobile robot should deal with uncertainty by generating high-level contingent plans. Uncertainty stems from the use of sensors and actuators with limited precision, from exogenous agents in the environment (e.g., people moving objects or closing doors), and from incomplete knowledge about the environment (e.g., the robot may not know the arrangement of objects in a room until it gets to perceive them).

Given that a robot generates a plan of actions to achieve a goal based on a model of its own actions and a model of its environment, uncertainty during planning can be dealt with in two complementary ways. First, it can be ignored by making deterministic assumptions about the effects of actions and the dynamics of the environment, and assuming complete knowledge about the robot and environment state; that way, a deterministic plan (i.e., a sequence of actions) is generated and assumed valid as long as the assumptions hold during the execution; by monitoring the execution of the plan, the executive might detect violation of the deterministic planning assumptions, in which case the planner is re-invoked to generate a new plan fitting the new situation. This is often referred to as a classical planning framework [9].

Many plan-based robot platforms integrate such a classical planning framework for task level behaviors, that is, a level at which robot activities such as grasping an object, releasing an object, or going from one room to an adjacent one, are considered as primitive actions (e.g., see [4, 16, 2, 7, 11, 3, 13]). It is not unusual, however, to integrate such a deterministic task-level planner with some form of uncertainty reasoning at lower level behaviors. For instance, to determine the robot position (which is among features describing states at the level of the task-level planner), the executive can invoke a position localization component, which determines the robot position based on a map, a probabilistic model about its sensors and a probabilistic model about its actuators [18]. The action of moving from one room to another can be instantiated into a lower-level navigation process that invokes a path planner.

In this paper we are interested in validating the efficiency of contingency planning at the task level for mobile robots and determining what aspects of nondeterminism should be abstracted away at the planning time and better handled via re-planning as is the case in traditional approaches. We have developed a robot control architecture that integrates a particle filtering position localizer and the MBP nondeterministic planner [5] to plan task-level and path-level robot behaviours.

Our preliminary results show that this contingency planning strategy is feasible and that it brings some flexibility and robustness that are otherwise difficult to obtain with a deterministic task planner. Contingent plans smoothly combine localization and goal-directed actions, and as such, our approach naturally handles situations that require the ability to think about the main robot task (e.g., deliver a given number packages to given different destinations) while localizing and vice-versa. Such situations are not easily handled in previous traditional plan-based robot control platforms because they consider achieving goals and localizing the robot as two separate interleaved processes [16, 2, 7, 11, 3, 13].

The remainder of the paper is structured as follows. First, we briefly discuss our view of contingency planning, and the features for the MBP planner that we use for this purpose. To evaluate the merit of our approach, we define a reference robot architecture that uses a deterministic planning approach and that in a way subsumes many models of interleaving planning and executions in robotics as far as our discussion is concerned. Then, we describe our contingency-

---

[1] Universitat Pompeu Fabra, Barcelona, Spain, email: alexandre.albore@upf.edu

[2] Université de Sherbrooke, Quebec, Canada, email: {Eric.Beaudry,Froduald.Kabanza}@USherbrooke.ca

[3] ITC-Irst, Povo, Italy, email: bertoli@itc.it

based architecture that integrates MBP and position localization. Finally, we present and discuss the results from some tests, before closing with future and related work.

2 Contingency planning

Following [6], our approach to contingent planning is based on modeling a planning domain as a finite state machine, with an initial state uncertainty, non-deterministic action outcomes, and partial observability with noisy sensing. As a starting point of our investigations, we use qualitative notions of nondeterminism and noise. We do not recur to probabilities to characterize the likelihood of nondeterministic outcomes for robot actions or the degrees of believes of a robot with imperfect sensors or incomplete knowledge.

Definition 1 (Planning domain) *A non-deterministic planning domain with partial observability is a 6-tuple* $\mathcal{D} = \langle \mathcal{S}, \mathcal{A}, \mathcal{U}, \mathcal{I}, \mathcal{T}, \mathcal{X} \rangle$, *where:*

- $\mathcal{S}$ *is the set of* states.
- $\mathcal{A}$ *is the set of* actions.
- $\mathcal{U}$ *is the set of* observations.
- $\mathcal{I} \subseteq \mathcal{S}$ *is the set of* initial states*; we require* $\mathcal{I} \neq \emptyset$.
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \rightarrow 2^{\mathcal{S}}$ *is the* transition function*; it associates with each current state* $s \in \mathcal{S}$ *and with each action* $a \in \mathcal{A}$ *the set* $\mathcal{T}(s,a) \subseteq \mathcal{S}$ *of next states.*
- $\mathcal{X} : \mathcal{S} \rightarrow 2^{\mathcal{U}}$ *is the* observation function*; it associates with each state* s *the set of possible observations* $\mathcal{X}(s) \subseteq \mathcal{U}$, *with* $\mathcal{X}(s) \neq \emptyset$.

A contingency planning problem is a pair $\langle \mathcal{D}, \mathcal{G} \rangle$, where $\mathcal{G} \subseteq \mathcal{S}$ is a set of goal states. We consider conditional plans that branch on the basis of observations.

Definition 2 (Conditional Plan) *The set of conditional plans* Π *for a domain* $\mathcal{D} = \langle \mathcal{S}, \mathcal{A}, \mathcal{U}, \mathcal{I}, \mathcal{T}, \mathcal{X} \rangle$ *is the minimal set such that:*

- $\epsilon \in \Pi$*;*
- *if* $\alpha \in \mathcal{A}$ *and* $\pi \in \Pi$ *, then* $\alpha \circ \pi \in \Pi$*;*
- *if* $o \in \mathcal{U}$, *and* $\pi_1, \pi_2 \in \Pi$, *then* if o then π_1 else π_2 $\in \Pi$.

Intuitively, ϵ is the empty plan, $\alpha \circ \pi$ indicates that action α has to be executed before plan π, and *if o then* π_1 *else* π_2 indicates that either π_1 or π_2 must be executed, depending on whether the observation o holds. A plan can also be thought of as a finite state machine that executes synchronously with the environment.

Due to nondeterminism and partial observability, many different possible executions are possible for a given plan, following different branches of a plan based on the actual values of observations at run-time. A plan is called a *strong solution* to a planning problem if every possible execution does not fail, and terminates in a goal state [6].

Search for a contingent plan is performed taking into account the fact that only incomplete information about the domain state is available for a plan executor, and that decisions about how to act must be taken on the basis of such information. More specifically, a planning problem is solved by searching through the space of *belief states*. A belief state represents a set of plausible states, and models the knowledge a robot has on its actual state and that of the environment . The belief space is an and-or graph, in which the "or" component of the branching expresses the ability to choose different applicable actions from a state, and the "and" component conveys the constraint of considering every possible sensor readings.

Belief-level search can be computationally very expensive, since the belief space can be exponentially larger than the world-state space. Contingent planning is actually 2-ExpTime complete [15]. Nevertheless, effective technologies have recently been developed and adopted to deal with reasonably complex domains. Binary Decision Diagrams [1] are structures that allow the compact representation and the effective manipulation of large sets of states; canonicity of this representation allow for cosntant-cost equality check amongst beliefs, a key operation to enforce loop avoidance during the search. The MBP planner [5, 6] exploits BDDs to perform belief-level search, and has proved to be capable to deal very effectively with complex domains.

3 Architecture

3.1 A Reference Deterministic Robot Planning Architecture

To validate the benefits of our contingency-based robot planning architecture, one possibility would be to compare it to different robot architectures that use deterministic, and there exist a large number of them [14, 16, 11, 10]. Making comparisons with different robot architectures would be quite difficult because each architecture comes with a different programming environment. Here we take a rather different, more economic approach. We define a reference architecture, implementing a basic model of interleaving deterministic planning and robot execution and reflecting the main aspects of the different architectures that are relevant for our discussion. The reference architecture, shown in Figure 1, is based on a two-layered planning component, on a set of behavioral modules and on a robot-position localizer. These components run concurrently, asynchronously, and each of them plays a dedicated role.

The top level planning component is a task-level deterministic planner. It takes as input an initial state and goal state (i.e., a robot mission, viewed as a planning problem). The initial state include features characterizing the current robot position as provided by the localizer. The planner can be based on any of the families of automated planning algorithms in AI planning [9]. Here we opted for an HTN planner, which can be easily programmed to follow specific task completion procedures if needed.

The HTN planner produces a plan that is a sequence of actions, where actions correspond to primitive robot tasks, such as grasping an object in reach, releasing an object, or going from one room to an adjacent one. An action is taken in charge by the executor by coordinating several dedicated processes (also called behaviors). For instance, "move to room" actions also involve the calculation of a path to follow by using a path-planner, then navigation-level processes to move the robot along the path, including obstacle avoidance, position tracking and moves between adjacent cells along a path.

These low-level behaviors are coordinated by using a priority-based arbitration mechanism to manage conflicts among behaviors affecting the same control command simultaneously, in a manner typical of other behavior-based control architectures [14, 12]. Each behavior participating in the accomplishment of a plan action is synchronized with the other behaviors, each getting the possibility to execute an atomic instruction every 100 millisecond; the atomic instruction may be a control command to the robot (e.g., set speed, turn rate); conflicts among simultaneous control commands are handled by the arbitrer.

Finally, the atomic instruction elected for execution is translated into actual commands to the actuators of the robot platform, and executed as such. During execution of the actions, robot sensors are probed to give feedback to the low-level command loop, to track the robot position, and to monitor the accomplishment of a task.
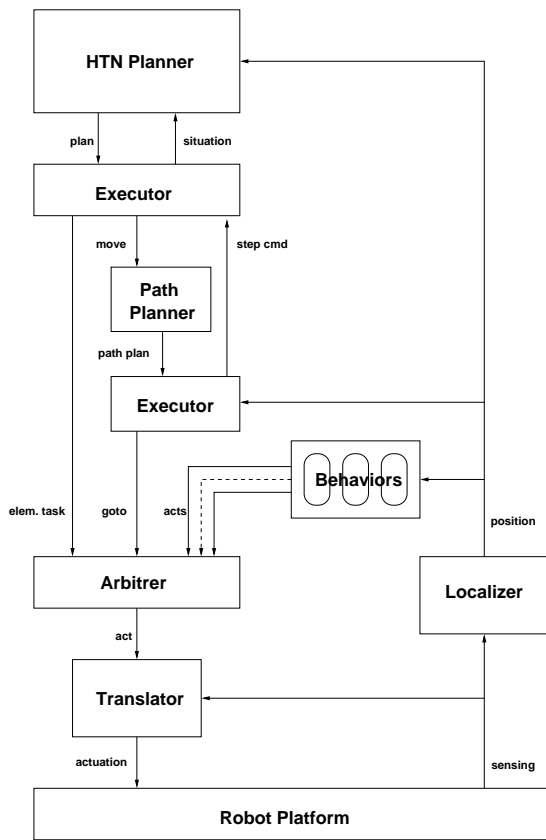
Figure 1. A reference hybrid robot control architecture

Figure 2. A hybrid robot control architecture for contingency planning

its task, e.g. it will move the robot into areas where the model of sensors suggests it has best chances of being localized – while at the same time taking into account the requirements for achieving a goal. Also notice that, on the other side, in some cases the planner is able to generate plans for achieving goals, yet without knowing exactly the robot position, and relying only on a belief state about probable positions.
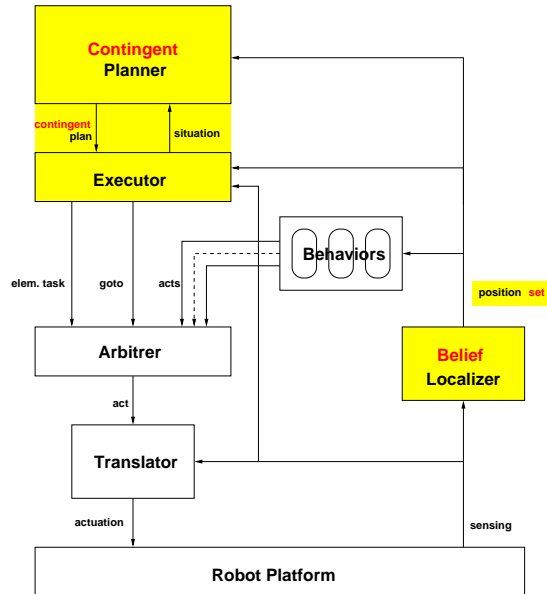
3.2 A Contingent Planning Robot Architecture

The new architecture we propose is outlined in Figure 2. Shaded portions highlight differences with the previous reference architecture. In the new architecture, the localizer does not return a single robot position. Rather, it returns a set of n most probable robot positions, that is, a *belief position*.

To appreciate the implications of this new approach, first note that both architectures use a particle filtering approach for tracking the robot position [18]. A particle filtering approach tracks a robot position by maintaining a set of probable robot positions (each position is called a particle). The set of particles represents a belief state about the robot position. In particular, the robot position is determined when a sufficient number of particles converge around one unique position; that is, when the uncertainty in the robot position is reduced below a given threshold or, conversely, when the confidence in the robot belief position is above a given threshold. In the reference architecture, since the planner cannot handle uncertainty, this situation must be enforced by the localizer, and just the most probable particle is considered.

In the new approach, since the MBP task planner is able to reason with incomplete states, it accepts a set of n probable positions given by the localizer. The number n has to be reasonably small to manage the planning complexity. Our approach relies on a confidence threshold and a cardinality bound to infer the (at most) n most probable position from the localizer's set of particles and pass them to MBP.

Notice that, since the task planner reasons about a belief position, and must proceed to deambiguate uncertainty enough to guarantee goal achievement, its actions are likely to support the localizer in

4 Experimental evaluation

We ran robot simulations and compared the behavior of the two architectures. We wanted to establish whether the ability to deal with uncertainty at the task-planning level improved the behavior of the robot in terms of goal achievement, and whether the additional computational complexity for this form of planning could be dealt with in practice.

Our current results are preliminary and concern navigation problems where a robot must drive across intersecting corridors. The simulation domain is illustrated in Figure 4. Each cell represents a possible robot position; the robot can be oriented in one of the four cardinal directions. The robot is represented as an arrow-shaped object to make its orientation clear. Walls are represented as thick lines, and of course cannot be traversed or walked upon.

Cells are 4-connected (the robot may navigate from one to the other following horizontal and vertical directions). The available actions for the robot are *step*, *turnLeft*, and *turnRight*. The first action allows the robot to move to the next cell in the faced direction, while the other two actions make the robot turn of $90°$ in the desired direction.

Our wheeled robot is equipped with laser sensors that provide information about the wall proximity: the robot can sense the presence of obstacles in the adjacent cells (in front, or at both sides), and as such, detect that there is a wall in front of it and/or on its side. The corresponding observations are *wallAhead*, *wallLeft*, and *wallRight*.
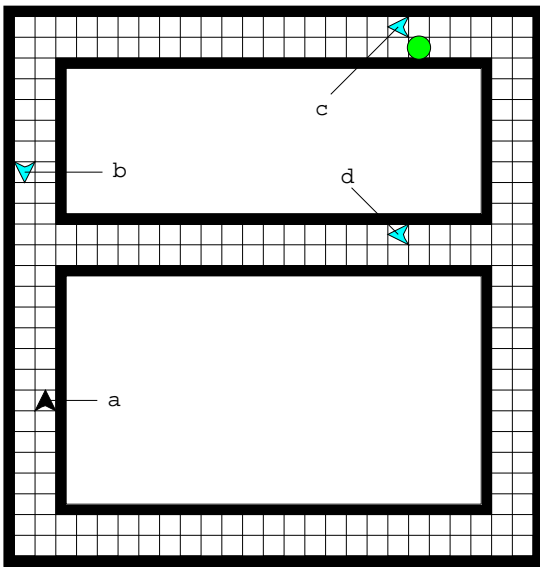
Figure 3. A simple robot domain

Figure 4. A useless plan generated by a classical planner.

Initially the robot is in position a , and it has to get to the position marked with a filled circle.

Notice that the initial position is ambiguous even considering all the sensors responses: because of the symmetry of the domain and the limited amount of information, many different starting positions can be confused with the real one. For instance, positions b, c, d convey the same observations of position a. This is a source of non-determinism for the navigation problem.

We first tested the deterministic planning architecture on this problem, running it 100 times - remember that given the same initial sensing, different position estimates can be given by the statistical localizer. Our results are that the goal has been reached only about 30% of the times. This is due essentially to the fact that the architecture "commits" to a single initial position estimate, which may be wrong, and builds a plan whose execution may not be able to detect that the robot is in a position different from the expected one, and that a different plan is needed. For instance, it may commit to position c, and then build a plan that simply turns left, steps, turns left again and makes a final step. As shown in Figure 4, executing this plan, the robot, whose actual initial position is a, would end up in a different position than the goal, namely a'. Still, during the execution, there would be no clue that leads to replanning: the observations gathered would be consistent with the expected ones, and any plan executor would still believe the robot started from c and got to c', i.e. to the goal.

We then tested the contingent planning architecture on the same problem, running it again 100 times, and having its belief localizer set up to return a belief containing up to 4 states.

As a result, we managed to achieve the goal all of the times, without the contingency planning generation significantly affecting the overall execution time of the architecture (MBP took in average 6.3 seconds to produce its plans, with no significant deviation).

The reason for this is explained by the fact that, in order to deal with contingencies, the plans generated by MBP smoothly integrate goal-directed actions with moves meant to localize, which may help detecting also situations unaccounted for within the current belief.
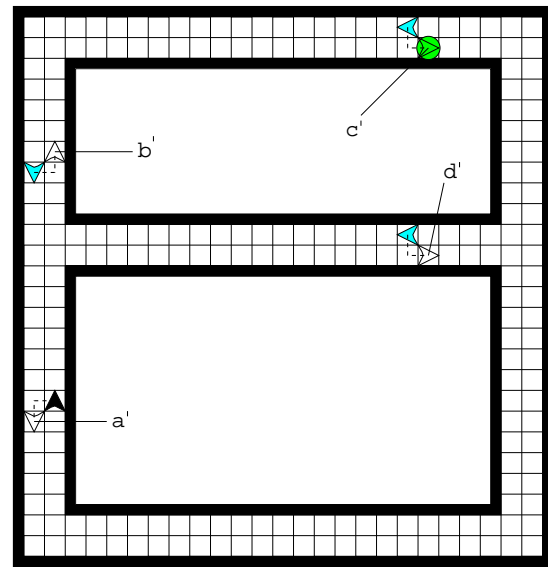
1 $17 \times step$
2 $turnLeft$
3 $step$
4 **if** $(\neg wallAhead)$ **then**
5 $turnLeft$
6 $22 \times step$
7 $turnLeft$
8 $step$
9   **if** $(\neg wallAhead)$ **then**
10 $9 \times step$
11 $turnLeft$
12 $4 \times step$
13   **done**
14   **else**
15 $turnLeft$
16 $5 \times step$
17 $turnLeft$
18 $step$
19 $turnLeft$
20 $6 \times step$
21   **if** $(\neg wallAhead)$
22 $turnRight$
23 $18 \times step$
24   **done**
25   **else**
26 $2 \times turnRight$
27 $step$
28   **goto** 12
29   **endif**
30   **else**
31 $2 \times turnRight$
32 $step$
33   **goto** 23
34 **endif**

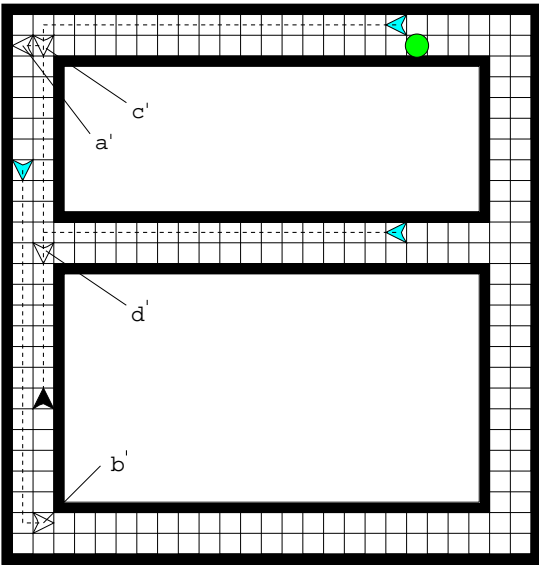**Figure 5.** Generated plan.

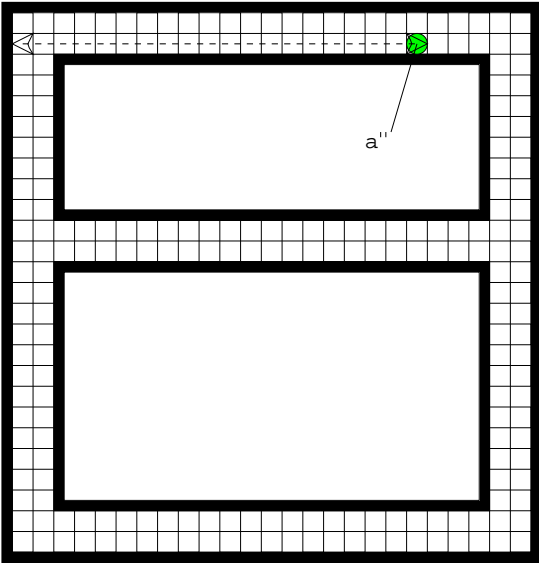**Figure 6.** First 19 navigation steps

Figure 7. Further steps to completion (starting from a).

As an example, consider the plan shown in Fig. 5, generated when the initial belief is made by the cells denoted by a, b, c, d. Such a plan acts first to remove the ambiguity about the position, and then aims at the goal. More precisely, after the first 19 actions (lines 1-3), the first observation determines whether the initial pose was a: only in a' the robot is facing a wall. In this case, the plan proceeds by executing lines 31,32,33,23,24, and gets to the goal, as shown in Fig. 7. If the pose initially is not in a, a different branch of the plan is followed. For instance, after executing lines 4-8, it is possible to understand whether the robot was initially in b (in which case, it would find itself facing a wall), and act consequently.

Now, assume that the robot was actually in a position outside the chosen initial belief (and compatible with the initial observations). For instance, assume it was in position e, see Fig. 8. Then, after the

first 21 steps, the plan executor would be able to distinguish that the starting position was none of a,b,c,d, and trigger replanning accordingly.
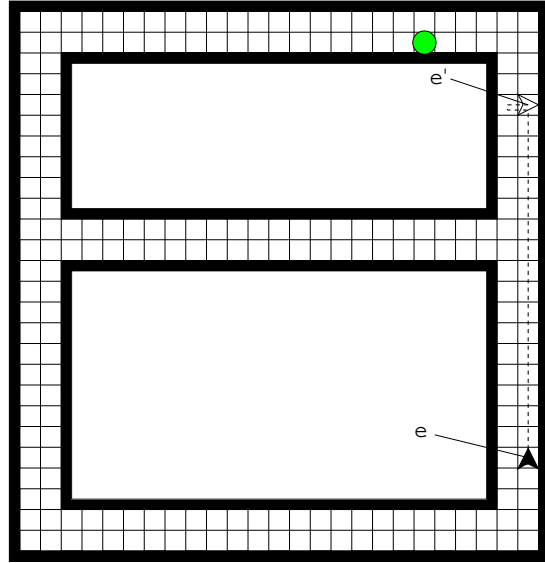

Figure 8. Replanning for an unexpected position.

5 Related Work and Conclusions

The integration of reactive robot behaviors and deliberative processes including reactive planning has been intensely studied. One emphasis in [16] was on defining a Task Description Language (TDL) amenable to an automated planning process. In [11], the focus is on integrating production rule-based robot reaction rules (specified using the Procedural Reasoning System) with an automated planner. In a similar vein, the ROGUE system integrate a behavior-based robot architecture with the PRODIGY planner [10]. Other approaches include [7, 13, 2] and some of them address the issue of monitoring and handling metric time constraints.

An MDP framework could be used at the task level as well. However, although various general-purpose MDP planning frameworks exist, to our knowledge they still have to be tested into robot architecture at the task level. So far, they have been involved essentially at the navigation level (e.g., see [8]). Markov decision theoretic techniques have also been used in localizing the robot position as an alternative to particle filtering approaches [4].

Our focus in this paper was on effective task-level contingency planning for mobile robots. Task-level contingency planning has been addressed by many authors, but not focusing on integration with robot platforms. As we illustrated in this paper, the integration of contingent planning into mobile robot platforms brings some interesting opportunities, such as planning while localizing, bringing some promising results.

Our future work will deal with assessing the scalability of our approach, by testing on much larger robot domains (with hundreds of cells and more complex configurations) and real robots. Future work will also examine situations and contexts when the robot is better off by procrastinating planning until planning becomes effective (e.g., too much is unknown for the planned situation five minutes from

now, so we should postpone planning until we know more) and to develop principled strategies for planning for such situations. Another topic for future work is to include optimality criteria into the quality of generate plans. This will require either extending MBP planner to deal with optimality, or adapting another task-planning approach that has this feature, such as a Markov decision-theoretic planner [17].

REFERENCES

[1] S. B. Akers, 'Binary decision diagrams.', *IEEE Trans. Computers*, **27**(6), 509–516, (1978).

[2] J. Ambros-Ingerson and S. Steel, 'Integrating planning, execution and monitoring', in *Proceedings National Conference on Artificial Intelligence*, (1998).

[3] E. Beaudry, Y. Brosseau, C. Cote, C. Raievsky, D. Letourneau, F. Kabanza, and F. Michaud, 'Reactive Planning in a Motivated Behavioral Architecture', in *Proceedings National Conference on Artificial Intelligence (AAAI)*, pp. 1242–1249, (2005).

[4] M. Beetz, W. Burgard, D. Fox, and A. Cremers, 'Integrating active localization into high-level robot control systems', *Robotics and Autonomous Systems*, (1998).

[5] P. Bertoli, A. Cimatti, M. Pistore, M. Roveri, and P. Traverso, 'MBP: A Model Based Planner', in *Notes of ICAPS'03: system demo session*, (2003).

[6] P. Bertoli, A. Cimatti, M. Roveri, and P. Traverso, 'Planning under partial observability', *Journal of Artificial Intelligence Research*, **170**, 337–384, (2006).

[7] S. Chien, R. Knight, A. Stechert, R. Sherwood, and G. Rabideau, 'Using iterative repair to improve the responsiveness of planning and scheduling', in *Proceedings Fifth International Conference on Artificial Intelligence Planning and Scheduling*, (2000).

[8] T. Dean, L. Kaelbling, J. Kirman, and A. Nicholson, 'Planning under time constraints in stochastic domains', *Artificial Inteligence*, **76**, 35–74, (1995).

[9] M. Ghallab, D. Nau, and P. Traverso, *Automated Planning: Theory and Practice*, Morgan Kaufmann, 2004.

[10] K. Haigh and M. Veloso, 'Interleaving planning and robot execution for asynchronous user requests', *Autonomous Robots*, (1997).

[11] F. Ingrand and O. Despouys, 'Extending procedural reasoning system toward robot actions planning', in *Proceedings IEEE International Conference on Robotics and Automation*, (2001).

[12] K. Konolige, K. Myers, E. Ruspini, and A. Saffiotti, 'The Saphira architecture: A design for autonomy', *Journal of Experimental and Theoretical Artificial Intelligence*, **9**, 215–235, (1997).

[13] S. Lemai and F. Ingrand, 'Interleaving temporeal planning and execution in robotics domains', in *Proceeedings National Conference on Artificial Intelligence (AAAI)*, (2004).

[14] R. R. Murphy, *Introduction to AI Robotics*, MIT Press, Cambridge, MA, USA, 2000.

[15] J. Rintanen, 'Complexity of probabilistic planning under average rewards', in *IJCAI*, pp. 503–508, (2001).

[16] R. Simmons and D. Apfelbaum, 'A task description language for robot control', in *Proceedings Conference on Intelligent Robotics and Systems*, (1998).

[17] S. Thiébaux, C. Getton, J. Slaney, D. Price, and F. Kabanza, 'Decision-theoretic planning with non-markovian rewards', *Journal of Artificial Intelligence Research (JAIR). To appear.*, (2006).

[18] S. Thrun, D. Fox, and W. Burgard, 'Robust monte carlo localization for mobile robots', *Artificial Intelligence*, (2001).