Plan Projection under the APPEAL Robot Control Architecture

Thorsten Belker   Martin Hammel   Joachim Hertzberg[*]
Department of Computer Science, University of Bonn
Römerstraße 164, D-53117 Bonn, Germany

Abstract. The paper presents APPEAL, a three-layer mobile robot control **A**rchitecture for **P**rojection-based **P**lanning, **E**xecution **a**nd **L**earning, with a focus on its execution layer. Besides task decomposition and failure recovery, it supports the projection of navigation plans based on learned models of navigation actions. Plan projection is applied to detect opportunities for improving the robot's navigation plan and transforming it accordingly. In the experimental section, we quantify the performance gains achieved by applying these techniques, both in simulation and on a robot. The savings are considerable.

1 Introduction

Symbolic task planners have been used in robot control since SHAKEY's [21] times. They allow a robot to achieve its tasks in a large number of situations without the need to explicitly specify its behavior for all possible situations and tasks. Hybrid robot control architectures [19, 16, 6] allow abstract, symbolic actions to be connected with the required reactive, behavioral facets of the robot control system. These architectures organize the slow deliberative planning process and the fast reactive control process in corresponding layers, namely, the deliberation and the behavioral layers, and connect them by the execution or sequencing layer. This layer is typically responsible for task decomposition and synchronization, execution monitoring, failure recovery, and resource management. RAP (Firby) [10], TCA/TDL(Simmons) [24], and ESL(Gat) [13] are high-level languages that are widely used for building sequencing layers.

The three-layer architecture APPEAL (**A**rchitecture for **P**rojection-based **P**lanning, **E**xecution **a**nd **L**earning) is the framework behind the work reported in this paper. APPEAL is an extension of the RHINO control architecture [8, 26]. On top of a behavioral layer for collision avoidance and local trajectory planning [5], an execution layer is built which implements a variety of functions, namely, localization, mapping, path planning, task decomposition, execution monitoring, and failure recovery [4]. The execution layer of APPEAL also provides a mechanism for projecting navigation plans based on learned models of the robot's actions. It supports the projection of navigation plans, which can be used to improve plan quality during plan execution. Plan projection and its application in APPEAL is the main focus of this

[*]Permanent address: Fraunhofer Institute of Autonomous Intelligent Systems (AIS), Schloss Birlinghofen, D-53754 Sankt Augustin, Germany

paper. The top layer, the deliberation layer, consists of a user interface with which a user can specify abstract navigation tasks. The deliberative layer may also contain a task or mission planner.

The idea of plan projection is to forecast the physical robot performance on the basis of the abstract, symbolic plan representation. This is not a simulation, and deliberately so. Plan projection tolerates all the abstractions and inaccuracies of the plan level domain description for the sake of efficiency of reasoning. Its purpose is to find with negligible effort possible alternative courses of action that look better in some respect than the currently available plan. If such a plan is found, it is swapped into the robot controller instead of the previous one.

We borrow the term plan projection from McDermott [18] and from its refinements later developed by Beetz [2]. Beetz's work on plan projection has focused on making the robot performance more robust by forestalling certain critical situations that some plan might lead the robot into. We focus on speeding up the execution of navigation plans, while conserving their expected robustness.

The remainder of this paper is organized as follows. The next section sketches the overall organization of APPEAL. Then we describe symbolic action plans as are used by the execution layer to represent navigation plans. Section 4 targets the mechanisms for projecting the robot's behavior. Section 5 presents the experiments carried out to evaluate the use of the projection mechanism for improving plan quality during execution, both in simulation and on a physical robot. Section 6 concludes.

2 A Sketch of the APPEAL Architecture

APPEAL is an extension of the RHINO control architecture. The RHINO architecture is well known for two reasons: First, its robust performance in two early tour-guide projects (in the *Deutsches Museum* in Bonn [7], and the Smithsonian Institute in Washington D.C. [25]) and second, the consequent application of probabilistic algorithms to both map learning [26] and localization [11].

The RHINO architecture is designed as a distributed system with modules for effector control, sensor interpretation, collision avoidance [12], map learning [26], localization [11], path planning [26], task planning [7, 25], and user interaction [23]. In the museum events, tasks were planned using GOLOG, a first-order logic programming language based on the situation calculus [17] and a structured reactive controller (SRC) [1]. Both systems are powerful for planning, but were mainly used to schedule user requests. GOLEX [14] was used for translating abstract actions into finite state machines and monitoring their execution.

APPEAL keeps the distributed, modular design, but introduces a layering as depicted in Figure 1. The lowest, *behavioral,* layer contains modules for effector control, sensor interpretation and a behavioral system, which extends the dynamic window approach [12] by local planning [5]. The robot approaches local target points while avoiding collisions with unexpected or dynamic obstacles. The middle, *execution,* layer contains the modules for localization, map update, path planning and plan execution. The execution layer replaces the GOLEX module of the original RHINO system. The top, *deliberation,* layer contains the user interface. It may also contain a task planner. Currently no task planner is used as the scheduling of navigation tasks is performed by the execution layer and no other complicated reasoning tasks have to be carried out, neither in the tourbot domain nor in the office delivery domain.

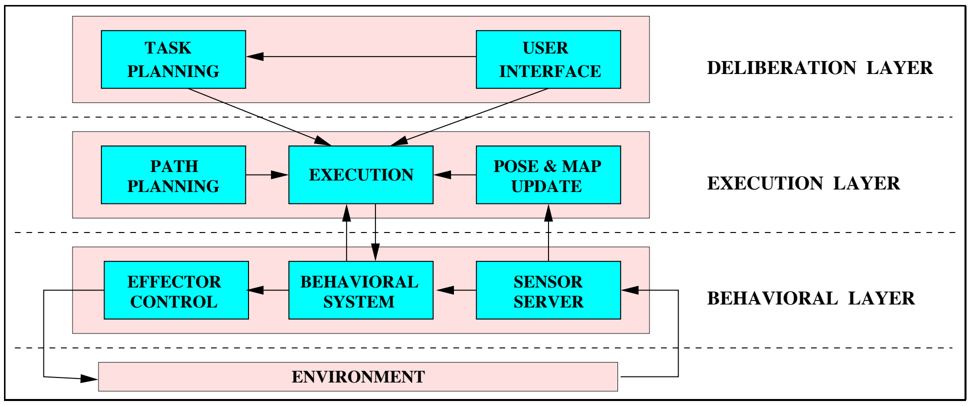


Figure 1: The layering of APPEAL: behavior, execution, and deliberation layers. Arrows denote the information flow.

3 Plans in APPEAL's Execution Layer

The execution layer of APPEAL integrates the pose estimation computed by the localization component, the continuously updated environment map and the status information of the behavioral system into a global world model. Based on the world model and the path planning module, the execution module can decompose user-specified tasks into tasks executable by the behavioral layer. The execution module also manages alternative plans to react to unforeseen execution failures and to optimize plan quality during execution as described in Section 4. Mapping and localization are taken from the RHINO architecture as published elsewhere [26, 11]. Our use of symbolic plans in APPEAL has recently been described [4]; we summarize it here to make the paper self-sufficient.

The execution module represents abstract navigation actions in terms of HTN (Hierarchical Task Network) [20] plans for robot navigation. HTN planning specifies a planning problem as a task network, i.e., a set of tasks together with constraints on their order and restrictions on variable bindings. Tasks may be *elementary,* i.e., executable by the robot, or *compound,* i.e., to be expanded into a task network. The expansion, in general, is not unique. Planning is performed by expanding compound tasks until only elementary ones remain. The resulting network of elementary tasks is a solution plan for the problem. In this paper, we use only total-order task networks and ground instances of tasks, handling linear propositional plans all the time. This simplification does, of course, not apply to HTN planning in general.

To introduce some terminology, we call *plan stub* an HTN with an elementary task in front. The strategy of stopping to reduce elementary tasks in an HTN as soon as a plan stub has been found, is called the *lazy expansion principle*. We apply it as the robot may start navigating even before a finished solution plan has been found. The rationale is that, first, we cannot assume that no unforeseen events occur during plan execution (which would make plan suffixes unexecutable or irrelevant) and that, second, plans may include sensing actions, the results of which may not be known at planning time. The lazy expansion principle reduces waste of planning time in these cases.

To give an example, here are some elementary and compound tasks in navigation. Again, we refer to [4] for a more detailed description. Elementary tasks for the navigation domain include: **SetTarget**(x, y, d) sets the target point (x, y) for collision-free drive control, to be approached up to d cm precision. **TurnTo**(x, y) causes the robot to rotate on the spot until

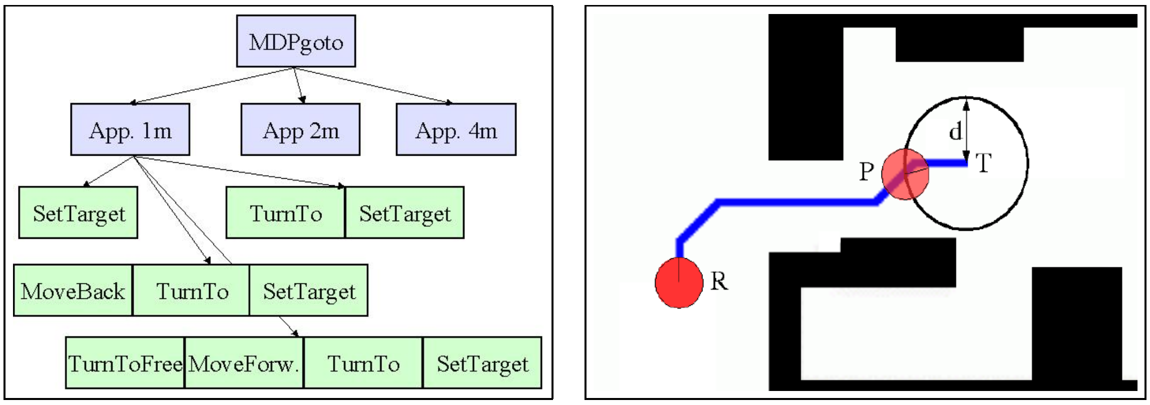


Figure 2: Left: Part of the task expansion hierarchy. Right: Projection of the position of a robot after successful completion of a navigation task. Arrows denote expansion hierarchies.

heading towards the point (x, y). **MoveForward/Backward**(d) causes the robot to move by d cm straight forward/backward.

Up next in the hierarchy is a task like **ApproachPoint**(x, y, d) which represents the choice of the next intermediate target point and which expands into a sequence of elementary tasks. Further up are tasks like **MDPgoto**(x, y) which is responsible for the path planning. Whenever a new **MDPgoto** task is inserted into the plan, the robot computes an optimal navigation policy for the given task based on MDP planning using a grid map of the environment. When the **MDPgoto** task is expanded into an **ApproachPoint**, an optimal path is computed from the policy, and a new target point on the path is computed that is either 1 m, 2 m, or 4 m ahead on the path depending on the selected expansion. Figure 2 (left) shows the part of the expansion hierarchy which is relevant for the experiments.

Further details are out of the scope here. However, keep in mind that all compound tasks have a default expansion and one or more alternative expansions, which may be used in turn if executing the most recent expansion has returned an error. For example, **ApproachPoint**(x, y, d) can, if its default expansion fails at some point, be expanded into the alternative sequences **TurnTo**(x, y), **SetTarget**(x, y, d), or **MoveBackward**(30), **TurnTo**(x, y), **SetTarget**(x, y, d).

Finding a working plan for some navigation mission is mostly easy, as many alternatives exist. However, these plans may differ considerably in quality, that is, in robot execution time. As the search for an optimal plan is computationally expensive, the robot should start execution as soon as a plan stub has been found. During execution, the robot may continue to search in the background for plans with higher expected performance. To support this kind of transformational planning, APPEAL's planner includes a mechanism for plan projection. The next section will discuss this mechanism for plan projection and its application to optimizing plan quality.

4 Plan Projection in APPEAL

Plans in APPEAL can be projected on any level of detail, that is on any level of the task expansion hierarchy. Plan projection on the level of **ApproachPoint** tasks is equivalent to the task of selecting intermediate target points for the optimal path to the goal. Plan projection in this case means to search for an optimal sequence of **ApproachPoint** tasks. We formulate the search problem by specifying the initial state, the goal test, the successor state function, and

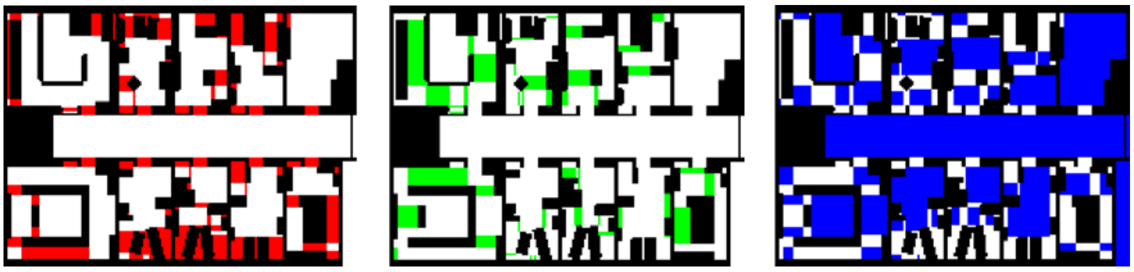


Figure 3: An environment segmentation based on the main axis clearance map with (a) narrow passages, (b) passages, and (c) free passages.

the cost function.

The initial state is the state of the robot at the time the search process starts. Each state that is sufficiently close to the **MDPgoto**'s goal state is a goal state in the search process. The robot's state after executing **ApproachPoint**(x, y, d) is estimated by computing the optimal path to the target $T = (x, y)$ and following the path to the first position $P = (x', y')$ on the path that is at most d cm away from T, facing T. This process is visualized in Figure 2 (right).

We assume that **ApproachPoint** tasks do not fail completely. Due to the exception handling performed in APPEAL, complete action failures are very rare. In the experiments, an unrecoverable failure occurred with a probability of less than 0.001. Please refer to [3] for a discussion of how plan projection can be modeled as Markov Decision Process in the case of frequent execution failures.

The prediction of the costs of executing an action a in state s is based on learned models of the action, more precisely, on the expected time the robot needs to execute a in s. As the time the robot needs to execute the navigation task depends on the shape and length of the path to the projected state P, we use a set of features derived from the path. Besides the path length, the path curvature (ratio of the Euclidian distance to the target and the path length) and the initial angle of the robot towards the path, the feature set contains features derived from the main axis clearance histogram of the environment map [3]. The features *NarrowPassageCounter*, *PassageCounter* and *FreePassageCounter*, for example, count the number of grid cells on the path that belong to a narrow passage, a passage or a free passage respectively. Figure 3 shows the segmentation of the robot's map into narrow passages (a), passages (b), and free passages (c).

For the learning we have applied model trees [15, 22], an extension of regression trees where the leafs contain linear predictions rather than constant predictions. Model trees implement piecewise linear regression models with main-axis parallel boundaries. We have decided to use tree-based induction methods for the function approximation as they provide in addition to a value prediction an explanation of the results, which can be translated into symbolic rules and is thus well accessible for human inspection.

Using this formalization of the plan execution problem, plan projection can be performed using any complete search algorithm. The first action on the shortest path that leads into a goal state is selected for expansion. To restrict the planning horizon, we allow only one possible action, the default action, for all nodes beyond the planning horizon.

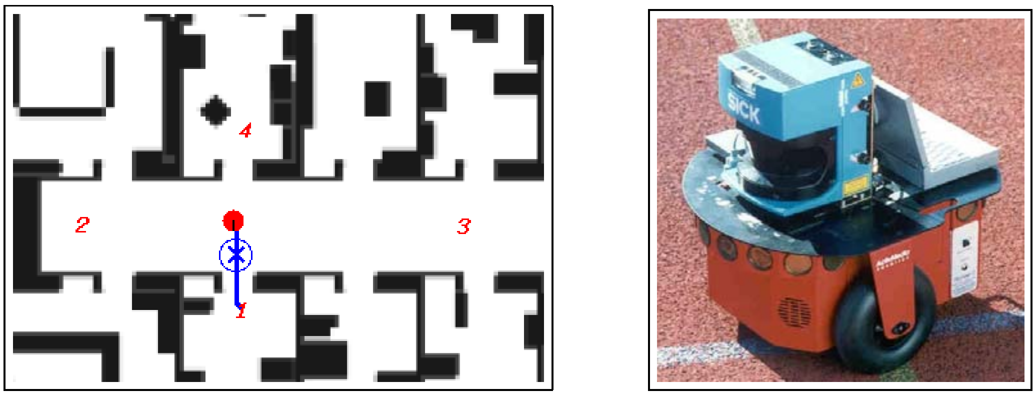


Figure 4: The four goal points for the real world experiments (left) carried out using a pioneer robot with laser range finder (right).

5 Experimental Results

In this section, two types of experiments will be presented that demonstrate the use of plan projections for optimizing the robot's navigation performance. The first experiment is performed on a PIONEER II platform. It demonstrates that models learned in simulation can improve the robot overall performance even when transferred to a physical robot. The second experiment is carried out in simulation and tests the utility of plan projection on a larger number of tasks.

5.1 Experiment 1

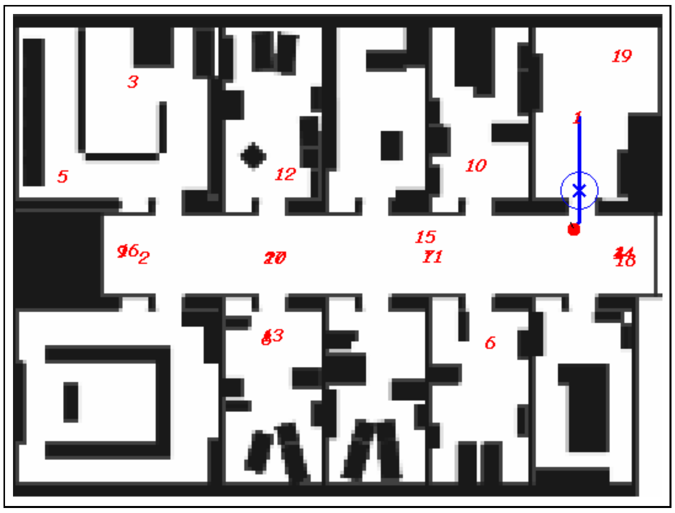
In the first experiment, a PIONEER II robot (Figure 4 right) is to execute the navigation tasks $1 \rightarrow 2$, $2 \rightarrow 3$, $3 \rightarrow 4$, and $4 \rightarrow 1$ for the four goals shown in Figure 4. We compare the performance of four different methods to select expansions of the **MDPgoto** task for these tasks.

In the following, the method of projection-based plan transformation as described above is denoted as **PROJ**. **PROJ(k)** is the same algorithm with a planning horizon of k. To evaluate the utility of the learned models, these methods are compared to two other standards. **DEFAULT** computes and executes the default plan without the application of any transformation. **CODED** applies a set of carefully hand-coded transformation rules to improve the robot performance.

To generate the data to learn the cost function for **PROJ** and **PROJ(1)**, the robot repeatedly executes the following procedure. It selects its next goal point randomly from the four goal points depicted in Figure 4 and executes the resulting **MDPgoto** task by randomly selecting possible expansions, i.e., intermediate target points.

Using a greedy error reduction splitting criterion, a single-variate linear prediction function, a depth-limit stopping criterion with a limit of 7, and a reduced-error post-pruning criterion, the model tree learner generates a set of 14 rules with one to six preconditions from about 7000 training examples.

To account for the high variance in the robot's behavior, the robot executes the sequence of four navigation tasks shown in Figure 4 ten times with each method. **PROJ(1)** performs on average 41.88 % better than **DEFAULT** and even 8.4 % better than **CODED**. Both perfor-



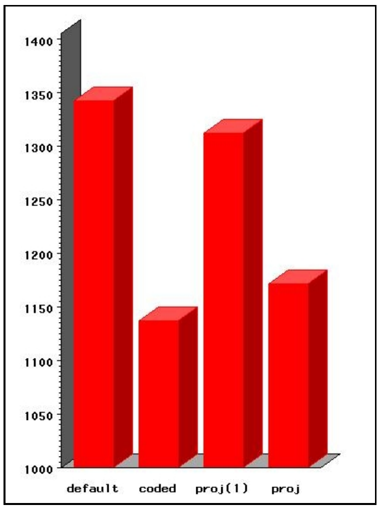


Figure 5: The twenty goal points used in the simulator experiment (left). The average time for the execution of the twenty tasks (right).

A vs. B	DEF.	CODED	PRJ(1)	PRJ
DEF.	0.5000	0.9998	0.6392	0.9963
CODED	0.0006	0.5000	0.0068	0.2690
PRJ(1)	0.3487	0.9909	0.5000	0.9693
PRJ	0.0031	0.7266	0.0313	0.5000

Table 1: The significance probabilities for the pairwise comparison of the different policies. The significance probabilities are computed using a randomized paired t-test.

mance improvements are statistically significant with respect to a significance level of 0.95. Surprisingly, **PROJ** does not perform significantly better than **PROJ(1)** in this case.

5.2 Experiment 2

In the simulator experiment, the robot has to execute the sequence of navigation tasks depicted in Figure 5 (left). Each sequence is executed five times using the different policies. Besides comparing the average time that the robot needs to accomplish the sequence of navigation tasks, we compute the statistical significance of an observed performance using a randomized paired t-test [9, pp. 168-170].

The cost function is learned on the basis of 7139 examples. In contrast to the first experiment, we apply multi-variate linear regression as prediction function. From the examples, the robot learns a set of 11 rules to predict the time it needs to complete an **ApproachPoint** task.

Figure 5 visualizes the time the robot needs to accomplish the sequence of navigation tasks using the different selection policies. The planned execution policies as well as **CODED** outperform **DEFAULT**. **PROJ(1)** improves the performance only by 2.22%, while **PROJ** improves it by 12.70%. The coded rules improve performance by 15.30%.

Table 1 shows the significance probabilities for the pairwise comparison of the different methods, that is, the probability to obtain the observed pairwise differences under the assumption that algorithm A and algorithm B in fact perform equally well. The probabilities have been computed using a randomized paired t-test which, in contrast to the parametric t-test, does not assume that each sample is drawn from normal distributions and with equal variance. With respect to the well-established significance level of α=0.05, **DEFAULT** is outperformed by **CODED** and **PROJ**. At this level, the latter algorithms do not differ in performance, both outperforming **PROJ(1)**.

The performance improvements in this case are less impressive than in the real robot experiment. This might have multiple reasons. First, the real tasks might be more difficult than the average task in the simulator experiment. Second, though there are less tasks in the experiment, there are almost as many examples to learn from. The learning algorithm thus might learn a model which is adapted very well to the four navigation tasks, but does not generalize well to other navigation tasks.

6 Conclusion

One of the many purposes of symbolic plans in hybrid robot control architectures like APPEAL is to increase the robot performance. The focus of this paper was to examine how plan projection can increase robot performance. To demonstrate its usefulness, we have examined the application of transforming the robot's navigation plan in order to improve its performance.

In both experiments, the increase in performance was considerable. Using learned transformation rules on plan level, the performance in physical office navigation could be improved by 41.9% w.r.t. straightforward navigation behavior as in the basic RHINO software. In the simulator experiments, where the performance of the robot for a larger number of tasks was analysed, the performance improvement was less impressive. In this case, however, plan projection with unlimited planning horizon was shown to outperform projection based planning with a planning horizon of one.

Plan projection comes for free in a hybrid control architecture in the sense that it may run asynchronously in parallel to the robot controller, consuming otherwise un-used compute power on-board the robot whenever available, or even running on other processors. From these results and considerations, we conclude that plan projection is a powerful tool for helping a robot perform better.

Acknowledgments

T. Belker is partly supported by the German Research Foundation (DFG) under contract number BE 2200/3-1. J. Hertzberg is partly supported by the German Federal Ministry of Research (BMBF) by the project AgenTec (01AK905B).

References

[1] M. Beetz. Structured reactive controllers — a computational model of everyday activity. In *Proceedings of the Third International Conference on Autonomous Agents*, 1999.

[2] M. Beetz. *Concurrent Reactive Plans: Anticipating and Forestalling Execution Failures*, volume LNAI 1772 of *Lecture Notes in Artificial Intelligence*. Springer Publishers, 2000.

[3] T. Belker, M. Beetz, and A. B. Cremers. Learning of plan execution policies for indoor navigation. *AI Communications*, 15(1):3–16, 2002.

[4] T. Belker, M. Hammel, and J. Hertzberg. Learning to optimize mobile robot navigation based on HTN plans. In *Proceedings of the International Conference on Robotics and Automation*, 2003.

[5] T. Belker and D. Schulz. Local action planning for mobile robot collision avoidance. In *IROS*, 2002.

[6] P. Bonasso, J. Firby, E. Gat, D. Kortenkamp, D. Miller, and M. Slack. Experiences with an architecture for intelligent, reactive agents. *Journal of Experimental and Theoretical Artificial Intelligence*, 9(1), 1997.

[7] W. Burgard, A.B. Cremers, D. Fox, D. Hähnel, G. Laemeyer, D. Schulz, W. Steiner, and S. Thrun. Experiences with an interactive museum tour-guide robot. *Artificial Intelligence*, 114(1-2), 1999.

[8] W. Burgard, A.B. Cremers, D. Fox, D. Hähnel, G. Lakemeyer, D. Schulz, W. Steiner, and S. Thrun. The interactive museum tour-guide robot. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI'98)*, 1998.

[9] P. Cohen. *Empirical Methods for Artificial Intelligence*. MIT Press, Cambridge, MA, 1995.

[10] J. Firby. Task networks for controlling continuous processes. In *Proceedings of the Second International Conference on AI Planning Systems*, 1994.

[11] D. Fox, W. Burgard, F. Dellaert, and S. Thrun. Monte carlo localization: Efficient position estimation for mobile robots. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence*, Orlando, FL, 1999.

[12] D. Fox, W. Burgard, and S. Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics and Automation Magazine*, 4(1), 1997.

[13] E. Gat. ESL: A language for supporting robust plan execution in embedded autonomous agents. In *Proceedings of the IEEE Aerospace Conference*, 1997.

[14] D. Hähnel, W. Burgard, and G. Lakemeyer. GOLEX - bridging the gap between logic (GOLOG) and a real robot. In *Proceedings of the 22nd German Conference on Artificial Intelligence (KI 98)*, 1998.

[15] A. Karalic. Employing linear regression in regression tree leaves. In *Proceedings of the tenth European Conference of Artificial Intelligence*, 1992.

[16] D. Kortenkamp, R.P. Bonasso, and R. Murphy, editors. *Artificial Intelligence and Mobile Robots: Case studies of successful robot systems*. MIT Press, Cambridge, MA, 1998.

[17] H.J. Levesque, R. Reiter, Y.Lesperance, F. Lin, and R. Scherl. Golog: A logic programming language for dynamic domains. *Journal on Logic Programming*, 31:59–84, 1997.

[18] D. McDermott. Transformational planning of reactive behavior. Research Report YALEU/DCS/RR-941, Yale University, 1992.

[19] R. Murphy. *Introduction to AI Robotics*. MIT Press, 2000.

[20] D.S. Nau, Y. Cao, A. Lotem, and H. Munoz-Avila. Shop: Simple hierarchical ordered planner. In *Proc. IJCAI-99*, 1999.

[21] N.J. Nilsson. Shakey the robot. Technical Report 323, SRI International, Menlo Park, California, 1984.

[22] J. Quinlan. Learning with contionous classes. In *Proceedings of the 5th Australien Joint Conference on Artificial Intelligence*, 1992.

[23] D. Schulz, W. Burgard, D. Fox, S. Thrun, and A.B. Cremers. Web interfaces for mobile robots in public places. *IEEE Robotics and Automation Magazine*, 7(1):49–56, 2000.

[24] R. Simmons. Structured control for autonomous robots. *Transactions on Robotics and Automation*, 10(1), 1994.

[25] S. Thrun, M. Bennewitz, W. Burgard, A.B. Cremers, F. Dellaert, D. Fox, D. Haehnel, C. Rosenberg, N. Roy, J. Schulte, and D. Schulz. Minerva: A second generation mobile tour-guide robot. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'99)*, 1999.

[26] S. Thrun, A. Buecken, W. Burgard, D. Fox, T. Fröhlinghaus, D. Hennig, T. Hofmann, M. Krell, and T. Schmidt. Map learning and high-speed navigation in rhino. In D. Kortenkamp, P. Bonasso, and R. Murphy, editors, *AI-based Mobile Robots: Case studies of successful robot systems*. MIT Press, 1998.