Learning to Optimize Mobile Robot Navigation Based on HTN Plans

Thorsten Belker
Dept. of Computer Science
Univ. of Bonn, Germany
belker@cs.uni-bonn.de

Martin Hammel
Dept. of Computer Science
Univ. of Bonn, Germany
hammel@cs.uni-bonn.de

Joachim Hertzberg
Fraunhofer AIS
Sankt Augustin, Germany
hertzberg@ais.fraunhofer.de

Abstract

High-level symbolic representations of actions to control the working of autonomous robots are used in all hybrid (reactive and deliberative) robot control architectures. Abstract action representations serve several purposes, such as structuring the control code, optimizing the robot performance, and providing a basis for reasoning about future robot action.

The paper presents results about re-designing the RHINO navigation system by introducing an HTN plan layer. Besides yielding a more structured robot control software, this layer is used as a basis for optimizing the navigation performance by plan transformations. We show how a robot can learn to select plan transformations based on projections of its intended behavior. Our experimental evaluation shows that the overall robot navigation performance is increased by almost 42 % when using learned projective models to select plan transformations.

1 Introduction

Abstract plan or task layers have been used in robot control since SHAKEY's times [17], and they are essential in hybrid robot control architectures [15, 12]. In McDermott's terminology [14], a plan is that part of a robot control program, which the robot cannot only execute, but also reason about and manipulate. According to that broad view, a plan may serve many purposes in a robot control system: As [1] has it, "the use of plans enables these robots to flexibly interleave complex and interacting tasks, exploit opportunities, quickly plan their courses of action, and, if necessary, revise their intended activities".

In this paper, we describe a technique for using a plan layer in robot control for optimizing the performance in robot navigation by learning from past navigation experience. We start with the well-known RHINO software [22], building on top of it abstract navigation tasks from which plans are generated and executed. We use HTNs [16] as the plan format, as it allows standard plans to be generated very efficiently and makes it easy for the control system programmer to express prior knowledge about priorities and preferred decompositions of navigation tasks under certain circumstances. Moreover, the hierarchical nature of HTN plans makes them a handy substrate for dealing with execution failure by jumping to higher levels of abstraction within the current plan and pick alternative task expansion strategies. This has been one of the reasons for developing it for the archetype of HTN planners, NOAH [19].

Based on the declarative representation of robot navigation in the HTN format, the expected action of the robot on a given task can be projected into the future, allowing its performance to be estimated and, wherever possible, to be improved if alternative courses of action are available that can be chosen instead. This idea of plan transformation is inspired by the work of Beetz and McDermott [2]. As past experience from navigation tasks can of course be accumulated, we end up in a life-long learning framework in which the robot is able, based on the navigation plan format, to improve its navigation performance. Our results show performance improvements of 42 % on average.

The remainder of this paper is organized as follows. In the next section, we briefly recapitulate the RHINO navigation system and discuss its major shortcomings. Then we introduce – mostly by way of example – HTN planning and discuss how HTN planning can help to improve the RHINO system. After that, the technical core of the paper describes the procedure of optimizing navigation performance using learning techniques. We finally present empirical results of running the procedure, and conclude.

2 The RHINO Navigation System

The RHINO navigation system is well known for two reasons. First, its robust performance in two early tour-guide projects in the Deutsches Museum in Bonn [6] and the Smithsonian Institute in Washington D.C [21] and second, the consequent application of probabilistic algorithms to both map learning [22] and localization [8, 10].

Figure 1 depicts the main components of the RHINO sys-

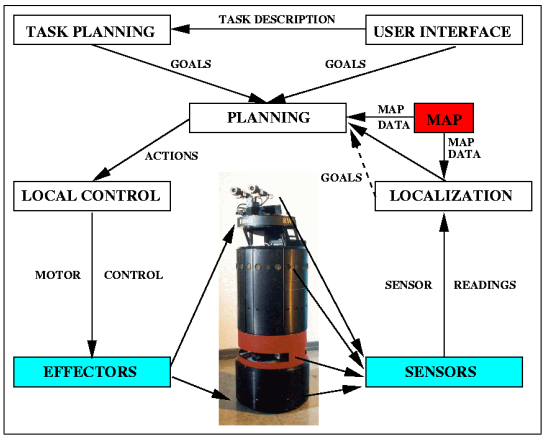

Figure 1: *Components of the* RHINO *system.*

tem and their interaction. The localization component tries to fit the sensor readings of a mobile robot into a model (a grid map) of the robot's working environment to estimate the robot's current position or pose within the environment and reduce uncertainty caused by unreliable dead-reckoning. A robust self-localization can be achieved using Bayes' rule to integrate noisy odometry information and noisy sensor information into a probability distribution of the robot's current position or pose within a known environment [8, 10]. The occupancy grids of the environment needed for localization can be learned [22]. While position tracking and navigation planning can be done concurrently, sensing actions might become necessary if the position uncertainty of the robot becomes too high. Burgard et al. suggest an entropy-based approach to compute sensing actions for active localization [7].

The navigation planning component receives goal points from either a user interface, a task planner or – in the case of active localization – from a localization component. It generates a sequence of actions that can be executed by a local reactive control component. In most systems, navigation planning is considered as an instance of path planning [13]. However, RHINO's navigation planner computes navigation policies rather than paths. Navigation policies are functions that assign a navigation action to each (discrete) position or pose of the robot. From a navigation policy, a path to the goal can be computed efficiently for every state of the robot. This is advantageous when the robot often deviates from its pre-planned path, e.g. due to drift or unexpected obstacles. Thrun et al. propose to use a deterministic value iteration to compute a navigation policy for a fine-grained two-dimensional occupancy grid map of the robot's environment [22].

The local control component ensures a safe execution of high-level navigation actions. It translates these actions into sequences of motor control commands that can be directly executed by the mobile robot. It guarantees a safe navigation by reacting to unforeseen and dynamic obstacles. The dynamic window approach to collision avoidance [9] generates trajectories for local navigation tasks based on simple models of the robot dynamics and chooses the trajectories with the highest utility with respect to some given evaluation function. A recent extension of this approach [4] combines the dynamic window approach with local planning to compute a better evaluation function.

As the local navigation component can fail to execute a navigation action due to inaccurate motor control, unexpected obstacles, and inherent limitations of the local control component, the planning component should be able to handle failures at execution time and plan for their avoidance. We will introduce a symbolic navigation planner which can account for execution failures on both levels. By projecting the execution costs of alternative plans, the planning component can optimize the robot's average performance using plan transformations.

3 HTN Plans for Robot Navigation

The basis for the improvement of the robot's performance reported in the rest of this paper is the representation of abstract navigation actions in terms of HTN (Hierarchical Transition Network) plans for robot navigation. By introducing a plan layer into the robot control, we continue the rich tradition of hybrid robot control architectures [15, 12].

HTN planning originates from early work by Sacerdoti [19]. Since then, it has been used as a technique in several domain-independent and special-purpose planners. Milestones include the SIPE-2 system [23] and its various applications, as well as BridgeBaron [20], the winner of the 1997 computer bridge world championship. SHOP [16] is a modern, domain-independent HTN planner incorporating the BridgeBaron design knowledge.

HTN planning specifies a planning problem as a task network, i.e., a set of tasks together with constraints on the order in which they can be performed and restrictions on how variables may be bound. Tasks may be *elementary*, i.e., executable by the robot (in our case), 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 tasks 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 first, we call *plan stub* an HTN with an elementary task as the first task in its ordering. 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*. This principle is used in our case as the robot may start navigating even before a com-

plete solution plan has been found. The rationale is that we cannot assume that no unforseen events occur during plan execution (which would make plan suffixes unexecutable or irrelevant) and that 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 present an example, let us introduce the operator inventory for our navigation planner. We have the following schemata for elementary tasks:

SetTarget(x, y, d) sets the target point (x, y) for the low-level routine for collision-free drive control, which has to be approached up to a precision of d cm.

TurnTo(x, y) causes the robot to rotate on the spot until heading towards the point (x, y).

TurnToFree$()$ causes the robot to rotate on the spot until heading towards some free space.

MoveForward(d) causes the robot to move by d cm straight forward.

MoveBackward(d) causes the robot to move by d cm straight backward.

All elementary tasks must have an implementation in terms of low-level control routines of the robot, so that executing an elementary task means calling the respective routine. That does, of course, not guarantee that each and every elementary task instance, or its corresponding control routine, respectively, can be successfully executed. Failure is possible, as usual. This issue will be addressed below.

We have two types of compound tasks, the schemata of which are:

ApproachPoint(x, y, d) drives the robot to position (x, y) within an error radius of d. In terms of the expansion hierarchy, **ApproachPoint** is a middle-level task that serves for dealing with self-generated intermediate target points.

MDPgoto(x, y) drives the robot to the user-specified target point (x, y). It is the highest task in the expansion hierarchy. Over the time, it gets expanded into a sequence of **ApproachPoint** operators.

In the experiments, we consider the following expansions. The task **MDPgoto**(x, y) can be expanded into **ApproachPoint**(t_x, t_y, c) with the target point (t_x, t_y) either 2 m (default), 1 m or 4 m ahead on the optimal path to the goal point (x, y) and with c=1 m, c=0.5 m, or c=2 m respectively. **ApproachPoint**(x, y, d) is either expanded into **SetTarget**(x, y, d) (default), into the sequence **TurnTo**(x, y), **SetTarget**(x, y, d), into the sequence **MoveBackward**(30), **SetTarget**(x, y, d), or into a sequence of **TurnToFree**, **MoveForward**, **TurnTo**, and **SetTarget**.

Within a plan, an *instance* of any task has all arguments fully instantiated, as we are dealing with purely propositional plans here. Moreover, all task instances have an additional argument saying whether they are PENDING, i.e., not yet executed (for elementary tasks) or expanded (for compound tasks), or EXPANDED in the case of compound tasks. Executed tasks are simply deleted from the current plan.

Representing a plan as a stack, here is an example. Assume the navigation target is the position $(1521.31, 1563.8)$ on some given floor map. This would be transformed into the one-task plan

MDPgoto$(1,$ PENDING$, 1521.31, 1563.8)$

where the first two arguments are the task instance ID and the status, resp., and the following ones are like in the task schema descriptions given above. (This pattern will re-appear in all other task instances to follow.)

Dealing with the top task of the stack means expanding it, in this case, since it is compound. Using the default expansion, this yields

ApproachPoint$(2,$ PENDING$, 1434.38, 1009.38, 100)$
MDPgoto$(1,$ EXPANDED$, 1521.31, 1563.8)$

and, expanding the **ApproachPoint** task,

SetTarget$(3,$ PENDING$, 1434.38, 1009.38, 100)$
ApproachPoint$(2,$ EXPANDED$, 1434.38, 1009.38, 100)$
MDPgoto$(1,$ EXPANDED$, 1521.31, 1563.8)$

As the topmost task is elementary, this is a plan stub, and according to the lazy expansion principle, this operator gets immediately executed by the robot, causing a physical robot drive action.

Assuming that all goes well, the control routine implementing the **SetTarget** task terminates successfully. This task pops out, and so does **ApproachPoint** in consequence. The **MDPgoto** task, however, is not yet finished, as its target point is not yet reached according to the robot's self-localization. In consequence, it gets re-expanded, yielding

ApproachPoint$(4,$ PENDING$, 1476.88, 1158.12, 100)$
MDPgoto$(1,$ EXPANDED$, 1521.31, 1563.8)$

the topmost task of which would get expanded into the respective **SetTarget** task and executed as before. If all keeps going well, this cycle of expand-execute-pop is repeated until the final target point is reached and the **MDPgoto** task pops out.

Failures to execute an elementary task are reported by the low-level control routines by raising exceptions of different types. Assume that executing the top task in the plan

SetTarget$(5,$ PENDING$, 1476.88, 1158.12, 100)$
ApproachPoint$(4,$ EXPANDED$, 1476.88, 1158.12, 100)$
MDPgoto$(1,$ EXPANDED$, 1521.31, 1563.8)$

results in an exception of type NO-ADMISSIBLE-TRAJECTORY, i.e., the low-level execution cannot find an unoccluded local path from the current position to the point $(1476.88, 1158.12)$, based on the recent sensor readings. As a result, the failed task would pop out and the next expansion alternative for the **ApproachPoint** task would be patched in, resulting in

MoveBackward$(6,$ PENDING$, 30)$
SetTarget$(7,$ PENDING$, 1476.88, 1158.12, 100)$
ApproachPoint$(4,$ EXPANDED$, 1476.88, 1158.12, 100)$
MDPgoto$(1,$ EXPANDED$, 1521.31, 1563.8)$

If this should fail again, there are more ways of expanding **ApproachPoint**. Only if all expansion alternatives for some compound task have been exhausted, there will be backtracking in the tree of possible expansions, or, if no more backtracking is possible, the execution of the main task fails and permanent failure is reported.

The two examples demonstrate that the HTN framework nicely supports the fast generation of default plans and the handling of exceptions caused by unexpected obstacles or inaccurate effectors. In addition, HTNs can be used to optimize the robot's navigation performance, that is, to minimize the expected execution time. In some situations, for example, it might be advantageous for the robot to turn to the target point before trying to approach it. In other situations, e.g. in a wide corridor, it might improve the robot's performance to have a target point that is more distant and allows the robot to drive faster. All these different courses of action can be represented as different expansions of the **ApproachPoint** or **MDPgoto** schema. As soon as an opportunity of improving the robot's navigation plan has been detected, the planner can backtrack in the expansion tree and select another expansion that seems to be more suitable for the current situation.

4 Learning to Optimize Navigation Performance

In the previous section, we have argued that HTNs support the optimization of navigation performance by opportunistic plan transformations. However, the detection of opportunities for plan improvements is often difficult and the specification of a good detector requires much insight in the robot's operation. In this section, we therefore propose a method that projects different plan stubs and selects the one which causes the lowest expected execution costs (time). Due to the declarativity of the HTNs, the execution of different tasks can be monitored and a prediction of the execution costs can be learned from past experience.

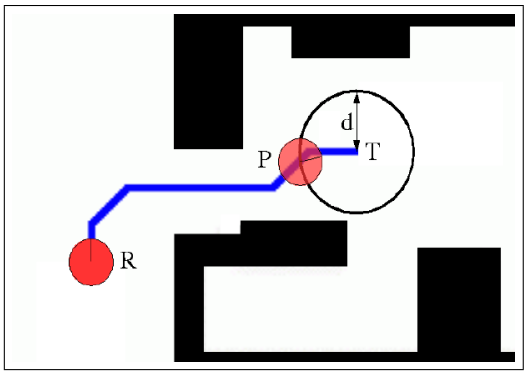

Figure 2: *Projection of the position of a robot after successful completion of a navigation task.*

The projection of the position and orientation of the robot after successfully executing **SetTarget**(x, y, d) is done 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. The robot is supposed to face T. The time the robot needs to arrive at position P has to be estimated based on previous experience of executing this kind of task. 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 (the 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 feature *crossesDoor*, for example, is true if the path crosses a region with low main axis clearance. The other primitive tasks are straight forward to project as the execution costs are either constant or only depend on the angle the robot has to turn. Compound tasks are projected by projecting their (default) expansion.

For the learning itself we have applied model trees [11, 18], an extension of regression trees [5] where the tree 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. Figure 3 shows a rule learned for the task of predicting the durations of a navigation task.

```
IF      (pathCurvature < 1.05) AND
        (NOT crossesDoor)      AND
        (pathLength ≥ 110.00   AND
        (pathLength < 130.00)
THEN    duration = 1/23.99 * pathLength
```

Figure 3: *One of the rules learned for the prediction task.*

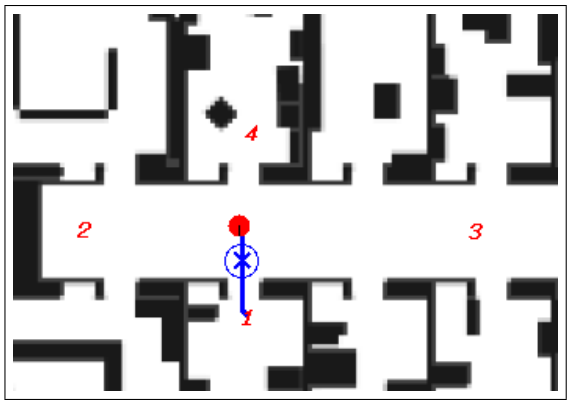

Figure 4: *The four goal points for the real world experiments.*

5 Experimental Results

In this section, two experiments will be presented that demonstrate the use of HTN plans for improving mobile robot navigation. The first experiment shows how plan transformations can be used to optimize navigation performance. While in this experiment hand-coded transformation rules are used, the second experiment shows how machine learning techniques can help to disburden the human programmer from specifying these rules.

5.1 Plan Transformations

In this experiment, the following three hand-coded transformation rules are used to improve the robot's navigation performance.

1. If the robot has its back to the target point and its clearance is low, then the robot turns towards the target point before approaching it.
2. If the robot's clearance gets small along the path fast, then the target point is set only one meter ahead.
3. If the robot's path has a low curvature and the clearance stays high along the path, then the target point is set four meters ahead.

These transformations are implemented using the alternative expansions of the **ApproachPoint** task (rule 1) and the **MDPgoto** task (rules 2,3). During execution of the plan, the robot starts to execute the default expansion of a given task immediately and does not wait until all alternative expansions have been considered. During execution the transformation rules are evaluated and applied if possible.

To evaluate the use of these transformations, the robot executes the sequence of four navigation tasks shown in Figure 4 ten times, both with and without transformations. The tasks are chosen such that the behavior of the robot in four different situations can be analyzed. Entering an office, leaving an office, navigation in the hallway, and navigation from one office to another. By applying the above mentioned transformation rules, the navigation performance of a PIONEER II robot with laser range finder (Figure 5) is on average improved by 30.88 %.

Figure 5: *The Pioneer II platform*

5.2 Learning to Predict Navigation Performance

Instead of specifying transformation rules by hand, the robot can transform plans based on projections of its behavior and a learned prediction of the time it takes the robot to execute a navigation task. The second experiment shows that this results in a navigation performance which is comparable to the behavior achieved with the transformation rules described above which we consider as expert knowledge.

To generate the data to learn the prediction function, the robot repeatedly executes a sequence of 10 navigation tasks in simulation by randomly selecting a possible reduction of the **MDPgoto** and **ApproachPoint** tasks.

Using a greedy error reduction splitting criterion, linear prediction functions, 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.

The learned rules are used to predict the performance of the robot. If an alternative reduction of the current task is projected to result in a better performance of the robot than the default reduction, the plan is transformed accordingly. This results in a navigation performance of the PIONEER II which is 41.88 % better than without using plan transformations and 8.4 % better than with hand-coded transformation rules. Both performance improvements are statistically significant with respect to a significance level of 0.95.

However, the performance gain differed considerably for the four tasks. While there was no significant performance gain for the first task (entering the office), the performance gain for the second task (leaving the office) was 51.37 %, for the third task (navigation in the hallway) 50.68 %. In the last task the robot on average performed even 65.97 % better. The results show that the transformations are especially useful for the task of leaving an office and fast navigation in the hallway.

Conclusion

Symbolic navigation plans allow control knowledge to be expressed and used in the robot controller in a structured way—that is the rationale behind all hybrid robot control architectures. The HTN framework allows for compact and efficient representation of such plans and alleviates to deal with execution failures in the lower level control routines in a transparent way.

The contribution of this paper is, first, to demonstrate the advantage of using an abstract HTN layer of navigation plans in a robot navigation system that has been used successfully over an extended period of time on various kinds of robots. Besides a transparent execution failure handling, such a layer facilitates opportunistic plan transformations. In the experiments, the navigation performance could be improved by 30.88 % using three hand-coded transformation rules. Second, we have shown the way for using machine learning techniques to predict the robot's performance in different situations. The learned models are essential for projecting the robot's behavior when executing a given plan. Plan projections can be used to trigger plan transformations whenever they promise an improvement of the robot's performance. Combing planning and learning in this way, the robot could improve its performance by 41.88 % on average. The learned rules thus outperform the hand-coded rules by 8.4%.

Future work will include the learning of transformation rules. The learning will be based on previously acquired projective model as described in this paper and therefore not require a human trainer.

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, J. Hertzberg, M. Ghallab, and M. Pollack. Preface. In M. Beetz, J. Hertzberg, M. Ghallab, and M. Pollack, editors, *Advances in Plan-Based Control of Robotic Agents*. Springer, LNAI 2466, 2002.

[2] M. Beetz and D. McDermott. Expressing transformations of structured reactive plans. In *Recent Advances in AI Planning. Proceedings of the 4th European Conference on Planning*, 1997.

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

[5] L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone. *Classification and Regression Trees*. Wadsworth, Inc., Belmont, CA, 1984.

[6] 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.

[7] W. Burgard, D. Fox, and S. Thrun. Active mobile robot localization. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, 1997.

[8] 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.

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

[10] D. Fox, W. Burgard, and S. Thrun. Markov localization for mobile robots in dynamic environments. *Journal of Artificial Intelligence Research*, 11, 1999.

[11] A. Karalic. Linear regression in regression tree leaves. In *Proceedings of ISSEK '92 (International School for Synthesis of Expert Knowledge)*, 1992.

[12] 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.

[13] J.-C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, 1991.

[14] D. McDermott. Robot planning. *AI Magazine*, 13(2):55–79, 1992.

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

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

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

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

[19] E.D. Sacerdoti. *A Structure for Plans and Behavior.* Elsevier/North Holland, 1977.

[20] S.J.J. Smith, D.S. Nau, and T. Throop. Success in spades: Using ai planning techniques to win the world champinonship of computer bridge. In *Proc. AAAI-98/IAAI-98*, pages 1079–1086, 1998.

[21] 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.

[22] 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 [12], 1998.

[23] D. Wilkins. *Practical Planning. Extending the Classical AI Planning Paradigm*. Morgan Kaufmann, San Mateo, CA, 1988.