Planning and Executing Joint Navigation Tasks in Autonomous Robot Soccer

Sebastian Buck, Michael Beetz, and Thorsten Schmitt

Munich, University of Technology, Germany
{buck,beetzm,schmittt}@in.tum.de

Abstract. In this paper we propose a hybrid navigation planning and execution system for performing joint navigation tasks in autonomous robot soccer. The proposed system consists of three components: an artificial neural network controller, a library of software tools for planning and plan merging, and a decision module that selects the appropriate planning and execution methods in a situation-specific way. The system learns by experimentation predictive models for the performance of different navigation planning methods. The decision module uses the learned predictive models to select the most promising planning method for the given navigation task.
In extensive experiments using a realistic and accurate robot simulator that has learned the dynamic model of the real robots we show that our navigation system is capable to (1) generate fast and smooth navigation trajectories and (2) outperform the state of the art planning methods.

1 Introduction

In order to perform plays competently, teams of autonomous robots playing robot soccer must be capable to perform joint navigation tasks given by a target state for each robot. In this paper we describe how the navigation system of the AGILO RoboCup team [3] solves the joint navigation problem. The distinctive characteristics of the AGILO navigation system are the following ones. First, the system uses a recursive neural network controller as its basic execution component. This neural network controller automatically learns how to best transform the robot's current dynamic state into a given target state. Second, the navigation system employs a library of single robot navigation planning and plan merging methods that it can select, combine, and apply to a given navigation task. Third, the navigation system automatically learns to predict which methods are best for which kinds of navigation tasks. Exploiting this knowledge it can apply the most promising planning method of its toolbox to the navigation tasks it is to perform. This way the navigation system performs better than any of the individual planning methods that it is using.
One of the key problems in the design of a multi robot navigation system is that different multi robot navigation planning methods make different assumptions about the kind of navigation problems they are to solve and the capabilities of the robots they are to control. Because these assumptions are implicit and not

A. Birk, S. Coradeschi, and S. Tadokoro (Eds.): RoboCup 2001, LNAI 2377, pp. 112–122, 2002.

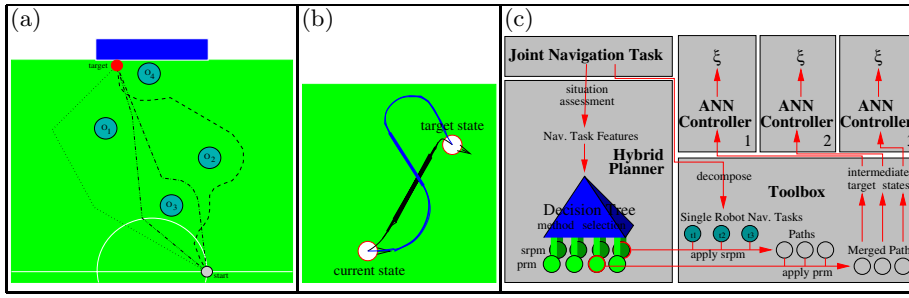

Fig. 1. Subfigure (a) shows navigation plans for one robot proposed by different path planning methods. Subfigure (b) shows two possible trajectories a controller can guide the robot. The orientation of start and target state is indicated by an arrow. The width of the trajectory indicates the robot's translational velocity. Subfigure (c): The planning and execution system: The *hybrid planner* computes the situation dependent navigation task features and uses a decision tree to determine an appropriate single robot planning method (srpm) and plan repair method (prm). In the *toolbox* first the srpm is applied to the decomposed single robot navigation tasks. Thereafter the obtained paths are merged by the prm. The intermediate target states of the merged paths are passed to the *artificial neural network (ANN) controller* of each robot. The controller then computes the low level command ξ to be executed.

well understood, selecting the **right** planning method for a given application domain and parameterize it optimally is very difficult.

Let us illustrate this point using a practical example. Figure 1(a) depicts a single robot navigation task in a typical game situation and the navigation plans proposed by different navigation planning algorithms. The figure illustrates that the paths computed by the different methods are qualitatively very different. While one path is longer and keeps larger distances to the next obstacles another one is shorter but requires more abrupt directional changes. The performance that the paths accomplish depends on many factors that the planning algorithms have not taken into account. These factors include whether the robot is holonomic or not, the dynamic properties of the robot, the characteristics of change in the environment, and so on.

The conclusion that we draw from this example is that the choice of problem-adequate navigation planning methods should be based on empirical investigations. For our investigations we develop a feature language which allows us to classify navigation tasks along dimensions that challenge planning methods. The remainder of this paper is organized as follows. An overview of the software architecture is given in section 2. Section 3 introduces our basic robot controller. Section 4 shortly describes the path planning algorithms used in our empirical investigation. Section 5 introduces a set of features that can be used to measure the characteristics of multi robot navigation problems along certain interesting dimensions. Section 6 then summarizes the results of our empirical investigation. We conclude with a review of related work and a discussion of the results.

2 The Structure of the Hybrid Navigation System

To plan *and* execute a joint navigation task we use a hybrid system containing a software robot controller, a toolbox for path planning, and a hybrid planner to select the appropriate algorithms for execution.
The hybrid navigation planning and execution system works as follows (see figure 1(c)). The system is given a joint navigation task that specifies a target state (position, orientation and velocity) for each robot of the team. The objective of the navigation system is to achieve a state where each robot is at its target state as fast as possible. The first step in the performance of the navigation task is the selection of the most appropriate planning tools. This is done by first assessing the given navigation task and the situation in which it is to be executed. The assessment assigns a navigation task a set of characteristics described in section 5.1. Thus features are then tested by a learned decision tree that assigns to each feature vector the most promising single robot path planning and plan merging method provided by the toolbox. In the second step, after the planning and plan repair methods have been chosen, the joint navigation task is decomposed into single robot navigation problems. The individual problems are then solved using the selected planning methods. Then, the individual plans are repaired in order to avoid negative interferences between the different plans. Finally, the toolbox extracts sequences of target states from the repaired plans and sends those sequences to the neural network controllers of the respective robots. The *robot controllers* are used for basic navigation to reach a given target state as fast as possible not considering any constraints. All constraints including moving obstacles, walls etc. are considered in the path planning algorithms of the *toolbox* which determine the intermediate target states.

3 A Recursive Neural Robot Controller

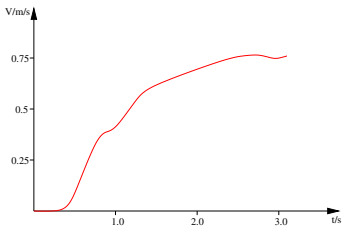

Fig. 2. The acceleration curve of a Pioneer I robot resulting from $\xi = (0, 0)$ for any $t < 0$ and $\xi = (0.75, 0)$ for $t \geq 0$. The dead time delay is about 300 ms.

The basic component of a navigation system is a controller that enables the robot to achieve specified dynamic states quickly.
Such a controller receives the target state (for example, the next state on a planned path) of a robot and returns low level commands that transform the current state into the target state as fast as possible. As shown in figure 1(b) there are different ways to solve this problem. To arrive at the target state different trajectories are possible. The dynamic state of a Pioneer I robot [11] can be summarized as a quintuple $\zeta = \langle x, y, \varphi, V_{tr}, V_{rot} \rangle$, where x and y are coordinates in a global system, φ is the orientation of the robot and V_{tr} (V_{rot}) are the translational (rotational) velocities. Using the Saphira software [7] one can set a command $\xi = \langle V_{tr}, V_{rot} \rangle$ at the frequency of 10 Hz where $V_{tr}(V_{rot})$ denote the target velocities in meters per second (degrees per second). But how to set them to quickly

reach the target state? We have measured a time delay of around 300 ms from the setting of ξ until the robot executes the command (see fig. 2). In order to take that delay into account we map commands ξ_i to the state change $\zeta_{i+3} \rightarrow \zeta_{i+4}$ as depicted in figure 3. In the remainder of the paper ζ_i and ξ_i denote a state and the respective command causing it.

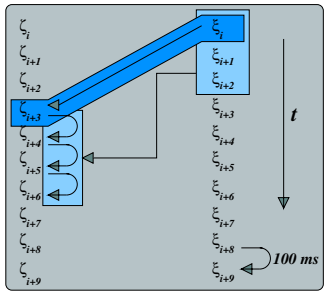

Fig. 3. Overcoming the dead time delay by assigning ξ to the state change 300 ms ahead.

A common approach (as proposed by [10] and described more thoroughly in [16]) is the use of fuzzy functions. This means to turn as long as the robot does not head towards its target state and to drive forward dependent on the orientation with respect to the target state.

In contrast, we learn a direct mapping from the robot's current state (ζ_0) and the robot's target state (ζ_{target}) to the next command to be executed (ξ_0) using multi layer artificial neural networks [4] and the RPROP [13] algorithm: *Net*: $\langle \zeta_0, \zeta_{target} \rangle \mapsto \xi_0$. Considering the start state at $x = 0, y = 0, \varphi = 0$ in a local system we can reduce the input dimension to 7 by converting the target state's x, y, φ into that local system (that means we regard $\Delta x, \Delta y, \Delta\varphi$).

To collect training data we do several runs with ξ set to certain constant values[1] and recording the state changes resulting. Out of it we get a huge number of patterns $\langle\langle \zeta_i, \xi_i \rangle \mapsto \zeta_{i+1}\rangle$. These patterns are inverted to $\langle\langle \zeta_i, \zeta_{i+1} \rangle \mapsto \xi_i\rangle$. Successive patterns $\langle\langle \zeta_i, \zeta_{i+1} \rangle \mapsto \xi_i\rangle$ and $\langle\langle \zeta_{i+1}, \zeta_{i+2} \rangle \mapsto \xi_{i+1}\rangle$ can recursively be combined to $\langle\langle \zeta_i, \zeta_{i+2} \rangle \mapsto \xi_i\rangle$ as illustrated in figure 4(a). From one simple trajectory we get lots of useful training patterns as we can see in figure 4(b). Patterns are created not only from start and target state but from any two consecutive states or patterns of the tra-

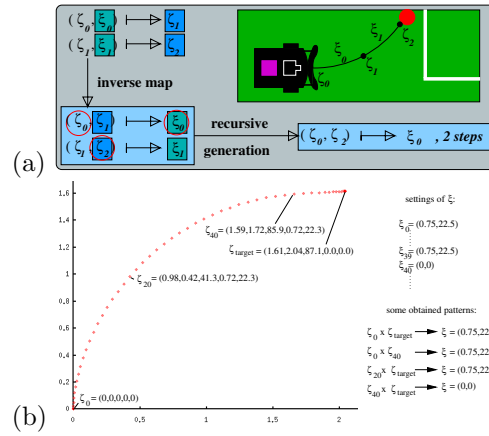

Fig. 4. Subfigure (a): Recursive generation of training patterns for the neural controller: Two successive state changes observed are inverted and combined to one new pattern. Subfigure (b): Driving 4 seconds with $\xi = (0.75, 22.5)$ and thereafter with $\xi = (0, 0)$ leads to numerous different patterns. The states from $\zeta_0 = (0, 0, 0, 0, 0)$ to ζ_{target} and 4 example patterns are plotted in the graph.

[1] one should always try to drive *fast* and not to increase, decrease and increase velocity again. Driving with only half speed on a straight line for example will teach the controller to do so even if it could reach the target state faster!

jectory. Recapitulatory we learn the dynamic driving behavior of Pioneer I robots and exploit it for navigation.

A Robot Simulator that Learns Models of Dynamic Behavior. An important means for developing competent navigation systems for robot soccer is a robot simulator that allows for controllable and repeatable experiments. For this reason we have developed a robot simulator that simulates how the dynamic state of the robot changes as the robot's control system issues new driving commands such as setting the target translational and rotational velocities.

The dynamic state of the robot is given as defined above. The robot control system issues driving commands $\langle V_{tr}, V_{rot} \rangle$. The dynamic model used by the simulator is acquired by learning the mapping $\Delta : \zeta_i \times \xi_i \mapsto \zeta_{i+1}$ from experience, that is recorded data from real robot runs. Our simulator learns this mapping using a simple multi layer neural network [4] and supervised learning with the RPROP algorithm [13]. Using this learning simulator we have learned the dynamics of Pioneer I robots. During data acquisition we have executed a wide variety of navigation scenarios that correspond to soccer plays. We have collected a total of more than 10000 training patterns from runs with a real Pioneer I robot. The accuracy for navigation tasks (including acceleration) is around 99% The accuracy decreases to about 92% in situations where both velocities, the translational and rotational one, are changed abruptly at the same time. These inaccuracies are caused by the lack of representative training patterns as well as the high variance in navigation behavior with maximal acceleration.

4 A Toolbox for Multi Robot Navigation Planning

The second component of our navigation system is a library of software tools for planning and repairing multi robot navigation plans. So far the planning methods library contains single robot navigation planning methods and methods for integrating the single robot plans through plan merging and plan repair afterwards.

4.1 Single Robot Path Planning

The path planning methods we will use in our toolbox include the *Potential Field Method* [5,1,15], the *Shortest Path Method* [9,6], *Circumnavigating Obstacles* [14], and *Maximizing the Clearance* [8]. Buck et al. [2] discusses in a more detailed overview the employed algorithms, their features, and abilities.

4.2 Plan Merging and Repair

The algorithms for single robot path planning can be coupled with plan merging and repair methods. We will now address the question of how to combine the individual plans in order to obtain a good performance on the joint navigation tasks.

Waiting Simply combining the paths computed by each robot without taking further precautions entails the danger that two robots might collide if their paths

intersect. The simplest fix is to let one robot wait when two robots are to reach an intersection at about the same time until the other robot has crossed the intersection.

Path Replanning The remaining methods try to revise the individual plans such that no negative interferences will occur. Again we assign priorities to the robots according to the importance of their navigation tasks and ask the robots with lower priority to revise their plans to avoid conflicts with the paths of the higher priority robots. We have considered three different methods for path revision called *Defining Temporary Targets*, *Hallucinating Obstacles at Critical Sections*, and *Inserting New Obstacles*. The first one modifies the path by introducing additional intermediate target points. The second one hallucinates additional obstacles at the positions where collisions might occur. The third one simply considers the other robot at its respective position as a static obstacle. Advantages and drawbacks of these methods are discussed in [2].

5 A Hybrid Navigation Planner

The third component of our navigation system is the hybrid navigation planner. The hybrid navigation planner selects based on the particular navigation task and the situation the task is to be performed in the most appropriate single robot planning and plan repair method. The working horse of the hybrid navigation planner is a decision tree where pairs consisting of a single robot planning and a plan repair method are stored in the leaves. The branches of the tree are labelled with tests on the characteristics of the navigation problem. The semantics of the decision tree is the following. For any navigation task that satisfies all the tests on a branch b is the planning method proposed by the leaf of the branch b the most appropriate one. In our case the main reason for using a decision tree for classification is that it leads to understandable rules while a neural network for example does not (by the advantage of quantifying the planning methods).
In the remainder of this section we describe the feature language that we use for the characterization of navigation tasks and the automatic learning of decision trees for the selection of the most appropriate planning methods.

5.1 A Feature Language for Multi Robot Navigation Tasks

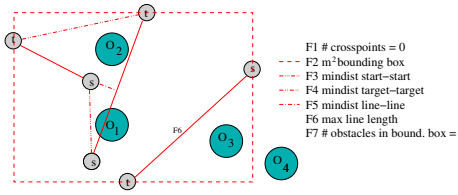

Fig. 5. Visualization of navigation task features that are used for classifying navigation tasks.

As mentioned in section 4 a large variety of different single robot navigation planning and plan merging methods exist, that have different strengths and weaknesses, and make different assumptions about the navigation problems at hand. This observation suggests that we need a language in which we can describe the characteristics of navigation problems

in a given robot control application and use these characteristics to select the appropriate new planning method.
In our investigations we will use 7 different features (see fig. 5). These features are: (1) the number of intersections between the line segments that represent the navigation tasks, (2) the size of the bounding box of the navigation tasks, (3) the minimal linear distance between different starting positions, (4) the minimal linear distance between different target positions, (5) the minimal distance between the line segments that represent the navigation tasks, (6) the maximum length of the linear distances of the individual navigation tasks, and (7) the number of obstacles in the bounding box of the joint navigation task.

5.2 Predicting the Performance of Planning Methods

A natural way for encoding a predictive model of the expected performance of different navigation planning methods in a given application domain is the specification of rules that have the following form:

if $c_1 \wedge ... \wedge c_n$ **then** fastest-method($\langle$ *srpm,prm*$\rangle$)

In this rule pattern the c_is represent conditions over the features that we use to classify navigation problems. The **then**-part of the rule asserts that for navigation problems that satisfy the conditions c_i the combination of the single robot planning method *srpm* together with the plan repair method *prm* can be expected to accomplish the navigation task faster then any other combination of single robot planning and plan repair method.
We have collected a training set of 1000 data records and used it for decision tree learning. From this training set the C4.5 algorithm [12][2] with standard parameterization and subsequent rule extraction has learned a set of 10 rules including the following one:

if there is one intersection of the navigation problems
$\wedge$ the navigation problems cover a small area ($\leq 10.7m^2$)
$\wedge$ the target points are close to each others ($\leq 1.1m$)
$\wedge$ the starting/target point distances are small ($\leq 5m$)
then fastest-method($\langle$*potential field,temp. targets* $\rangle$)

This rule essentially says that the potential field method is appropriate if there is only one intersection and the joint navigation problem covers at most one fourth of the field, and the target points are close to each others. This is because the potential field algorithm tends to generate smooth paths even for cluttered neighborhoods.
The accuracy of the ruleset for predicting the fastest navigation method is about 50% both for the training and the test set. A substantially slower algorithm was chosen only in very few cases. The inaccuracies of the rules have several reasons. First, the feature language as it has been introduced is not yet expressive enough. We expect that the accuracy of the rules can be substantially increased by adding

[2] For our experiments, we have used the public domain version of Quinlan's C4.5 algorithm

additional features such as the angle between the robot's current orientation and the direction to the target position. Second, in many navigation problems different methods achieved almost the same performance. In those cases we only selected the best one even when the margins where very narrow. Obviously, these data records are very noise sensitive. Third, in many runs collisions were caused by dynamic obstacles and have caused robots to get stuck. These runs resulted in outlier results that are not caused by the planning methods. Thus, for higher accuracy those runs have to be handled differently.
The conclusions that we draw from this experiment are that even with crude feature languages, without sophisticated data transformations and outlier handling a robot can learn useful predictive models for the performance of different navigation methods in a given application domain.

6 Experimental Results

In order to empirically evaluate our hybrid navigation planning and execution system we have performed extensive experiments using the simulator introduced in section 3. This simulator has enabled us to collect the necessary amount of realistic data, to make a simple quantitative comparison with respect to the average performance of the individual methods, and to find clusters of navigation tasks within the navigation tasks in the application that the individual methods solve well or poorly.
In the remainder of this section we describe two experiments. The first one is a quantitative comparison of the performance achieved by the different planning and plan repair methods provided by the toolbox. The results of this experiment suggest that it can, in general, not be expected that a single planning method can achieve heterogeneous navigation tasks equally well. In other words, we cannot expect navigation methods that dominate other ones in such complex application domains as robot soccer.
The second experiment shows that the hybrid navigation planner outperforms the individual planning methods on large collections of randomly sampled joint navigation tasks. This result suggests that it is indeed possible — even with such a simple feature language as ours — that a robot team can automatically learn predictive models of their planning methods that enable them to improve their performance both in a substantial and a statistically significant way.

6.1 Comparative Experiments

We set the number of obstacles to 4 (which is the team size of the mid size league). We carry out experiments with 3 robots (resulting from 3 field players in RoboCup). All our experiments underlie the same randomly generated situations: A robot starts at a randomly defined state in its configuration space and needs to get to a randomly defined target state. The obstacles move linearly from a randomly generated start point to a randomly defined target. If an obstacle reaches its target a new target is defined immediately. Obstacles move with a

Algorithm	Mean time values of 1000 train and 1000 test problems			
	TRAIN		TEST	
	μ/sec	significance level $P(\mu_{tree} < \mu)$	μ/sec	significance level $P(\mu_{tree} < \mu)$
Simple Potential Field	15.49	99.99 %	15.92	99.99 %
Shortest Path	13.36	99.99 %	13.14	99.99 %
Maximum Clearance	12.35	99.71 %	12.31	99.84 %
Viapoint	12.14	94.62 %	11.95	96.25 %
Decision Tree	11.64	50.00 %	11.44	50.00 %

Fig. 6. Results (mean time needed to solve a navigation task) of four evaluated algorithms and the trained decision tree. The significance level is based on a t-test.

random but constant velocity. Defining these preconditions we consider different dynamic behaviors of the obstacles without building a complex behavior model which would mean another laborious task.
Figure 6 pictures the mean value of the time resources required to complete a joint navigation task using the different planning methods introduced in section 4. The statistical data was acquired by performing 1000 randomly chosen navigation problems and performing the planning methods at a frequency of 10 Hz. The results show that based on the empirical data we cannot determine a single method that outperforms the other ones in a statistically significant way. This suggests that we should try to identify specializations of the navigation problems for which one planning method outperforms the other ones.

6.2 A Hybrid Navigation Planner

We have performed a bootstrapping t-test based on 1000 different joint navigation tasks (fig. 6) in order to empirically validate that the hybrid navigation planner performs better than the individual planning methods that we are using. Based on these experiments we obtained a 99.9% confidence in the test set (99.9% in the training set) that the hybrid method outperforms the potential field method (with its respective parameterization). The respective probabilities for the shortest path method are 99.9% (99.9%), for the maximum clearance method 99.84% (99.71%), and for the viapoint method 96.25% (94.62%). This means that our hypothesis that the hybrid planner dominates the other planning methods could be validated with statistical significance ($\geq 95\%$).

7 Conclusions

In this paper we have shown that in complex multi robot navigation planning domains it is extremely difficult to predict the performance of different kinds of planning methods. This is because the different planning methods make different kinds of assumptions. The maximum clearance method, for example, assumes

that it is better to keep larger distances to the objects and follow a longer path with higher speed, on the other hand, the circumnavigation methods assume that it is better to pass obstacles at a closer distance. This, however, requires the robot to have more accurate control over its dynamics, for example to be equipped with an omnidirectional drive. Therefore, it is unclear for most application domains to which degree the navigation tasks match the assumptions of the different methods. Even worse, yet within a single application domain we can identify classes of navigation problems for which the algorithms' suitability varies.

In this paper, we have therefore proposed to select problem-adequate navigation planning methods based on empirical investigations, that is the robots should learn by experimentation to use the best methods. To support this development strategy we have provided a feature language for classifying navigation problems and software tools that enable the robots to automatically learn predictive models for the performance of different navigation planning methods in a given application domain. We have shown, in the context of robot soccer, that a hybrid planning method that selects planning methods based on a learned predictive model outperforms the individual planning methods. The results were validated in extensive experiments using a realistic and accurate robot simulator that has learned the dynamic model of the real robots.

Even though these results are conclusive we expect that the performance can be substantially improved by using a more sophisticated feature language for the classification of navigation problems and by devising more sophisticated preprocessing methods for the data elements used for learning. In particular, we would expect that a competent module for rejecting outliers would reduce the deviations in the data substantially and improve the predictability drastically. These extensions of our basic framework are subject of our future investigations.

References

1. J. Barraquand, B. Langlois, and J. Latombe: *Numerical potential field techniques for robot path planning.* IEEE Transactions on Systems, Man, Cybernetics, 22(2):224–241, March/April 1992.
2. S. Buck, U. Weber, M. Beetz, and T. Schmitt: *Multi Robot Path Planning for Dynamic Environments: A Case Study.* Accepted for publication at IEEE/RSJ IROS, 2001.
3. S. Buck, R. Hanek, M. Klupsch, and T. Schmitt: *Agilo RoboCuppers: RoboCup Team Description.* RoboCup 2000: Robot Soccer World Cup IV, Springer, 2000.
4. J. Hertz, A. Krogh, R.G. Palmer: *Introduction to the Theory of Neural Computation.* Addison-Wesley, 1991.
5. Y. K. Hwang and H. Ahuja: *A Potential Field Approach to Path Planning.* IEEE Transactions on Robotics and Automation, vol. 8, 1, 23-32, 1992.
6. K. Konolige: *A Gradient Method for Realtime Robot Control.* Proceedings of the IEEE/RSJ IROS, 2000.
7. 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.

8. J.-C. Latombe: *Robot Motion Planning*. Kluwer Academic Publishers, 1991.
9. J. Lengyel, M. Reichert, B. Donald, and D. Greenberg: *Real–time robot motion planning using rasterizing computer graphics hardware*. Proceedings of SIGGRAPH, pages 327–335, August 1990.
10. B. Nebel and T. Weigel: *The CS Freiburg 2000 Team*. Fourth International Workshop on RoboCup, Melbourne, Australia, 2000.
11. Pioneer Mobile Robots, Operation Manual, 2nd edition, Active Media, 1998.
12. R. Quinlan: *Induction of decision trees*, Machine Learning 1 (1), 1986
13. M. Riedmiller and H. Braun: *A direct adaptive method for faster backpropagation learning: the Rprop algorithm*. Proceedings of the ICNN, 1993.
14. A. Schweikard: *A simple path search strategy based on calculation of free sections of motions*. Engineering Applications of Artificial Intelligence, 5, 1, 1 - 10, 1992.
15. P. Tournassoud: *A strategy for obstacle avoidance and its application to multi-robot systems*. Proceedings of the IEEE ICRA, pp. 1224-1229, 1986.
16. T. Weigel: *Roboter-Fuball: Selbstlokalisierung, Weltmodellierung, Pfadplanung und verhaltensbasierte Kontrolle*. Master Thesis, University of Freiburg, Germany, 1998