

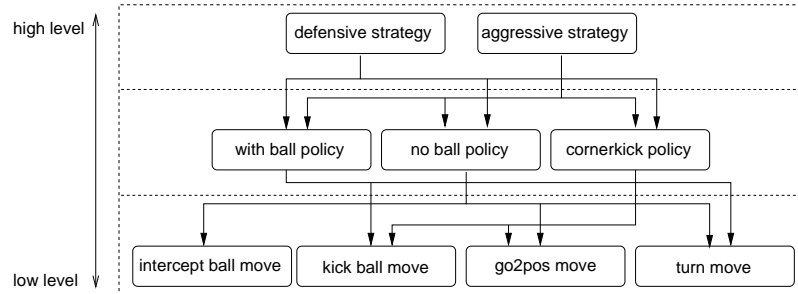
# Brainstormers 2003 - Team Description

M. Riedmiller, A. Merke, M. Nickschas, W. Nowak, and D. Withopf

Lehrstuhl Informatik I, Universität Dortmund, 44221 Dortmund, Germany

**Abstract.** The main interest behind the Brainstormers' effort in the robocup soccer domain is to develop and to apply machine learning techniques in complex domains. Especially, we are interested in reinforcement learning methods, where the training signal is only given in terms of success or failure. Our final goal is a learning system, where we only plug in 'win the match' - and our agents learn to generate the appropriate behaviour. Unfortunately, even from very optimistic complexity estimations it becomes obvious, that in the soccer domain, both conventional solution techniques and also advanced today's reinforcement learning techniques come to their limit - there are more than  $(108 \times 50)^{23}$  different states and more than  $(1000)^{300}$  different policies per agent per half time. This paper describes the new architecture of the Brainstormers team, the improved self-localization using particle filters and the extensions of the learning algorithm to simultaneously learn with and without ball behaviors.

## 1 New Architecture



**Fig. 1.** The old layer architecture

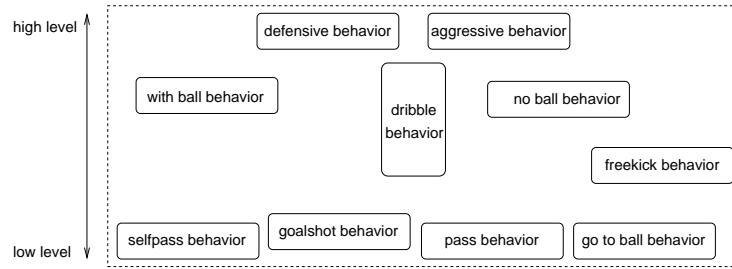
Until last year, the architecture of our team was a strictly hierarchical one, divided into strategies, policies and moves. The strategy is responsible for the high-level behavior of the team, e.g. play defensively or play aggressively. The policy acts as an intermediate level between strategy and moves. Typical instances include with- and without-ball-policies as well as standard-situations like goal- and

cornerkick or kickoff. The low-level skills are realised using moves like intercept-ball, goto-position and so on.

The idea behind that architecture was to repeatedly divide the team tactic into subtasks. To realise the expected behavior a strategy calls one or more policies, which in turn use moves to implement their tasks. But this was also one of the major drawbacks. Every new feature had to be integrated into one of the 3 layers, and it wasn't always clear which layer to chose for a new feature. And the control flow was strictly top-down, e.g. it was impossible for a policy to call a strategy (see figure 1).

This architecture was succesful, but after several years of changes and amendments it was difficult to see clear differences between the 3 layers of that architecture. There were some new features like dribbling, that didn't fit into the hierarchical concept.

Therefore we decided to alter the decision-making process, using behaviors inspired by behavior-based robot architectures. Our new architecture is still up to a certain degree a hierarchical one, but there are no restrictions like subdivisions into layers and strictly top-down control flow (see figure 2).



**Fig. 2.** The new behavior architecture

## 2 Improved Self-Localization Using Particle Filters

A player in simulation league receives visual information every cycle. Among these are the distances to landmarks with fixed positions around the field. Bearings to these landmarks can also be computed. A simple approach to localize the player is to intersect circles around the landmarks. The radius of these circles is determined by the sensed distance to the landmarks.

As the distances are quantized, there isn't a unique solution to that problem. By intersecting all pairs of 2 circles and then taking the mean value one can get a estimation of the players position. This method was used in our team until last year. One of the drawbacks was that the position estimation fluctuated very much.

Another interesting approach that only uses the bearings between the different landmarks has been introduced in [1]. If the bearings of the player to three landmarks are known, the position of the player can be determined. But this is only true if the measurements are noiseless. In a noisy environment as Robocup soccer server one has to integrate as many measurements as possible. In order to solve this overdetermined problem in a geometric way, a large system of non-linear equations has to be solved, which is not appropriate for players in Robocup, because it would be too time-consuming. Another possibility is to represent the landmarks as complex numbers, which leads to an overdetermined system of linear equations that can also be solved in linear time.

But as stated in [1], this approach yields poor results if the landmarks and the robot approximately lie on a circle. Unfortunately, this is true in the simulation league environments, where the players only have a small view cone.

To tackle that problem, we decided to use particle filters. This localization method has been used successfully with mobile robots ([2],[3]). Particle Filters try to represent the probability distribution  $p(l)$  (the probability that the robot is at position  $l$ ) by  $\mathbf{S} = \{ \langle l_i, w_i \rangle \mid i = 1 \dots n \}$ , a set of positions with appropriate weights, called particles. The particles constitute a discrete approximation of a probability distribution, therefore the weights should sum to one.

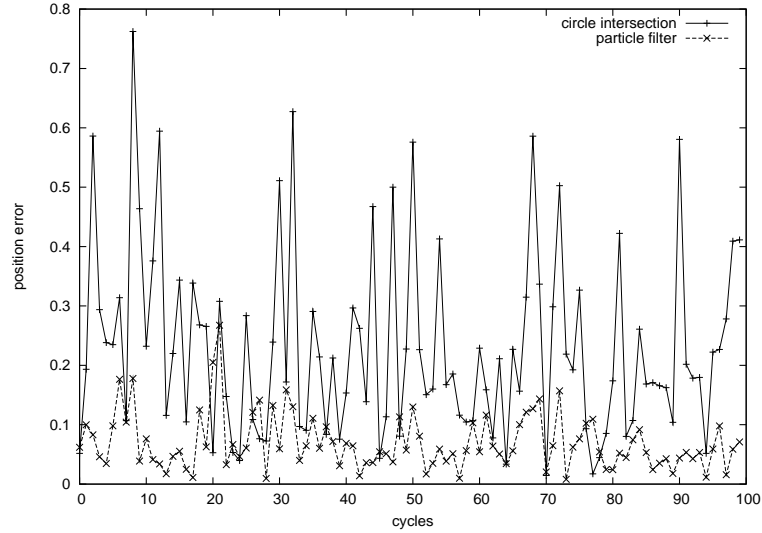
Let  $l$  be positions,  $a$  odometry readings and  $s$  sensor readings of the player. One has to know the movement model  $p(l'|a, l)$  and the sensor model  $p(s|l)$  to use the following algorithm. If the initial position the player is known, all particles positions are set accordingly and all weights are set to  $1/n$ . If the position is known, one uses  $n$  equally distributed random samples of positions on the pitch.

The algorithm works as follows:

1. Generate new particles: create  $N$  samples by drawing randomly from the particle set  $S$ , selecting each particle with probability  $w_i$
2. Integrate motion: according to the position  $l_i$  of the particles a new position  $l'$  is generated by drawing one sample from the odometry distribution  $p(l'|a_t, l_i)$
3. Integrate sensor readings: the particle weights are set to  $p(s_t|l_i)$
4. Normalization: normalize particle weights to guarantee  $\sum_{i=1}^n w_i = 1$

In our simulation league team, we used a normal distribution around the sensed movement as  $p(l'|a, l)$ . To compute  $p(s|l)$ , one can make use of the way the quantized distance measurements to the landmarks are generated from the real values. Using the formula from the soccer server manual, one can easily compute an interval that contains the true distance value. By doing this for all sensed landmarks, we get an area that contains the true position of the player. For all positions  $l$  that lie outside of this area  $p(s|l)$  is set to a value close to zero, for all other positions  $p(s|l)$  is set to 1.0.

To enable relocalization if the player is replaced on the field (e.g. in free-kick situation), the old circle-intersection method is still used as a reference. If this reference point is too far away from the mean position of all particles, a new particle with small weight is added at this reference point.



**Fig. 3.** Comparison of the two localization methods

A comparison of the old localization method and the new one using particle filters proved that the latter perform much better. The average position error decreased more than 60 percent. A plot of the position error over a sequence of 100 cycles is depicted in figure 2. The particle filter is not only most of the time more accurate than the intersect-circles method, but it also has less outliers, which is a very useful property.

### 3 Reinforcement Learning of Team Strategies

Many of the basic skills of our team like *intercept-ball*, *goto-position* and *kick* have been implemented using Reinforcement Learning approaches. These skills can be described as single-agent markov decision processes (MDP). For more information on learning single-agent skills see [4].

Our main research interest currently lies in the field of multi-agent markov decision processes (MMDP). Learning a team strategy can be described as such a MMDP with individual learners. As already stressed in [5] there is no guarantee that an optimal strategy is learned in such a case. In this section we describe the learning algorithm that was used to learn the positioning of players without ball and also the actions of the player with ball.

The key idea behind this approach is to learn a central value function  $V(s)$  for all players that describes how desirable a situation is. In other words,  $V(s)$  is a mapping from a state  $s$  to a value in  $[-1, 1]$ . A value close to 1.0 indicates that this situation is close to success (goal), a value near  $-1.0$  means that it is very probable to loose the ball in that situation.

In our approach a situation consists of ball position and velocity plus the position of all attacking teammates and all defending opponents. The number

of teammates and opponents that are used in the state representation for the value function has to be fixed beforehand.

The learning is done in epoches. In the beginning the value function is initialized randomly so the players pursue a random strategy. Now the players play according to that strategy until 5 succesful trajectories have been collected. If a trajectory was unsuccessful (e.g. loss of the ball) it is also stored. An example set  $E$  consisting of situations and rewards is generated from these 5 succesful plus maximal 5 unsuccessful trajectories.

The terminal state  $s_n$  of a trajectory gets a reward of 1.0 ( $V(s_n) = 1.0$ ) if it is a succesful terminal state and a reward of  $-1.0$  ( $V(s_n) = -1.0$ ) otherwise. The reward of the other states in the trajectory depends on the distance from the terminal state. Let  $s_1, \dots, s_n$  be the states encountered on a trajectory. The value of these states is then computed by:

$$V(s_i) = decay^{(n-i)} * V(s_n) \quad i = 1 \dots n - 1 \quad (1)$$

The parameter  $decay \in (0, 1)$  determines to what extent early states along the trajectories are responsible for the outcome of the trajectory. This idea is similar to the eligibility traces used by  $TD(1)$ .

The states together with the values are then used to train a function approximator. In our implementation we used a neural net that was trained with a variant of the backpropagation algorithm called RPROP ([6]).

The resulting net is then used in the next epoch to evaluate the different actions of a player. The action selection for a single player is done by generating an appropriate set of possible actions that the player could execute. The state after applying each of these actions is predicted using a model of the environment. The successor states are then evaluated using the value function and the best action is selected by choosing the corresponding action that leads to the succesor state with the highest value.

The action set of a player that owns the ball can be arbitrarily put together by selecting different types of actions like passes, dribblings, kick-and-rushes and so on. In our current implementation we use the following action types:

- passes directly to a teammate
- passes that a teammate can intercept before the opponent
- dribblings in one of three directions

This approach is very flexible and it is very easy to use new types of actions. The only thing that has to be done is to implement a model of the environment that predicts the successor state for these actions. The model for all of these actions assumes that the opponents don't move, only the teammates that are involved in the action are modelled. One could also think of a model that predicts the opponents behavior, but this would restrict the learned strategy to opponents that follow that modelled behavior. Therefore it is better to assume nothing about the opponents, to be able to generalize to unknown opponents.

The action set for a player without the ball is very simple and consists only of moving to different positions relative to the current player position. To prevent the players from moving arbitrarily on the field, we use the concept of home positions and home areas.

A player without ball can choose from the following actions (Actions that would move the player out of his home area are not allowed):

- go in one of eight directions from current position
- go to one of eight positions around the players home position
- go to home position

One limitation of that concept is that each acting players only considers its own action set to search for an optimal action. Theoretically a player would have to consider the joint action set  $a = (a_1, \dots, a_n)$  of all attacking players. As already mentioned in [5] this problem can be solved if the underlying MMDP is deterministic (see also [7]). As the soccer server environment is non-deterministic, one could change the soccer server to provide a deterministic environment and then hope that the optimal solution in this environment also works in the non-deterministic case. But as the action set size increases exponentially with the number of agents, this solution is intractable in practice.

## References

1. Betke, M., Gurvits, L.: Mobile robot localization using landmarks. *IEEE Transactions on Robotics and Automation* **13** (1997) 251–263
2. Fox, D., Burgard, W., Dellaert, F., Thrun, S.: Monte carlo localization: Efficient position estimation for mobile robots. In: *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI'99)*. (1999)
3. Fox, D., Thrun, S., Burgard, W., Dellaert, F.: Particle filters for mobile robot localization. In Doucet, A., de Freitas, N., Gordon, N., eds.: *Sequential Monte Carlo Methods in Practice*, New York, Springer (2001)
4. Riedmiller, M., Merke, A., Meier, D., Hoffmann, A., Sinner, A., Thate, O., Kill, C., Ehrmann, R.: Karlsruhe brainstormers - a reinforcement learning way to robotic soccer. In Jennings, A., Stone, P., eds.: *RoboCup-2000: Robot Soccer World Cup IV*, LNCS. Springer (2000)
5. Merke, A., Riedmiller, M.: Karlsruhe brainstormers - a reinforcement learning way to robotic soccer ii. In Birk, A., Coradeschi, S., Tadokoro, S., eds.: *RoboCup-2001: Robot Soccer World Cup V*, LNCS. Springer (2001) 322–327
6. Riedmiller, M., Braun, H.: A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In Ruspini, H., ed.: *Proceedings of the IEEE International Conference on Neural Networks (ICNN)*, San Francisco (1993) 586 – 591
7. Lauer, M., Riedmiller, M.: An algorithm for distributed reinforcement learning in cooperative multi-agent systems. In: *Proceedings of International Conference on Machine Learning, ICML '00*, Stanford, CA (2000) 535–542