

Making a Robot Learn to Play Soccer Using Reward and Punishment

Heiko Müller², Martin Lauer¹, Roland Hafner¹, Sascha Lange¹, Artur Merke²,
and Martin Riedmiller¹

¹ Neuroinformatics Group, Institute of Computer Science and Institute of Cognitive Science, University of Osnabrück, 49069 Osnabrück, Germany

² Lehrstuhl Informatik 1, University of Dortmund, 44221 Dortmund, Germany
{Martin.Lauer,Roland.Hafner,Sascha.Lange,Martin.Riedmiller}@uos.de,
{heiko.mueller,artur.merke}@udo.edu

Abstract In this paper, we show how reinforcement learning can be applied to real robots to achieve optimal robot behavior. As example, we enable an autonomous soccer robot to learn intercepting a rolling ball. Main focus is on how to adapt the Q-learning algorithm to the needs of learning strategies for real robots and how to transfer strategies learned in simulation onto real robots.

1 Introduction

Although various machine learning techniques have been used successfully in robotics, the idea of reinforcement learning [16] has not been applied very often on real robots so far. This is surprising since the basic idea of learning how to control autonomous robots just by rewarding them for good behavior and punishing them for bad behavior is very promising for solving complex tasks for which no optimal solution yet exists.

In simulation environments, reinforcement learning techniques have already been applied successfully (e.g. [6]), but the step from simulation to real world application has posed many problems. While in simulation, all state variables of a control task are perfectly known, on real robots these values have to be estimated using noisy and unreliable sensory input. Hence, state estimation shows a much larger error.

Secondly, state spaces are typically very large. While the theory of reinforcement learning is based on the assumption of finite state spaces, in practice we are often faced with infinite vector-valued state spaces with ten or more dimensions. Hence, approximations [2] have to be used in order to learn efficiently and to represent the control strategy. These techniques might even disturb the convergence of the learning algorithms [12].

Furthermore, high dimensional state spaces require millions of training examples to be able to learn an optimal policy. These millions of examples typically cannot be generated with a real robot since the robot would breakdown. Again, simulators and model assumptions must be used to overcome this problem.

Finally, real applications typically violate the Markov property which is the basis for all learning algorithms. Due to latencies in sensory processing and actuator execution, we are faced with time gaps of more than $100ms$, in many applications even much more [3]. In contrast, control cycles of many software frameworks are typically below $50ms$, so that an action is not executed completely when the next action must be selected.

Asada et al. have learned successfully different tasks on real robots using reinforcement learning [1,17,19]. In particular, they evaluated the automatic construction of higher-order state descriptions in order to solve the delay problem [19]. But due to several simplifications in their problem formulations, the resulting strategies have not been competitive and often are far from optimal control.

Speaking more generally, all the problems mentioned above have prevented reinforcement learning approaches so far from being competitive on real autonomous robots. Within this paper, we present a reinforcement learning approach to an autonomous soccer playing robot. The learning task is to intercept a rolling ball. This task has been discussed for a simulated robot before [6] and has now been transferred to the special needs of real robots. The intercept policy finally learned was integrated into the soccer robot strategy of the RoboCup [9] middle size league team *Brainstormers Tribots*, which became world champion in 2006.

In section 2, we first introduce the basic concepts of reinforcement learning which are relevant for this work and then show how reinforcement learning can be used to learn strategies for real robots in section 3. There, we discuss how to overcome the problems that have prevented the application of reinforcement learning on real robots until now. Section 4 shows the experimental results of learning the intercept scenario in simulation and on the real robots. The paper closes with a summary of the main results in section 5.

2 Reinforcement Learning

2.1 Markov Decision Process

Reinforcement learning is based on the idea of an autonomous agent interacting with an unknown environment. The agent observes the environment and decides to choose one out of several possible actions for interacting with the world. Depending on the agent's choice, the environment will change its state. Furthermore, the agent gets a reward for each state-transition. This reward can also be negative, i.e. the agent is punished instead of rewarded. The agent's goal is to achieve as much accumulated reward as possible over time.

This basic idea of reinforcement learning is modeled as a Markov decision process (MDP) [16]. It consists of a finite set of states S of the environment, a finite set of actions A that the agent can choose, a probabilistic state-transition kernel $P = (p_{s,s'}^a)$ where $p_{s,s'}^a$ describes the probability of a transition from state s to s' if the agent chooses action a , and a reward function $r : S \times A \rightarrow \mathbb{R}$ that defines the reward provided to the agent, depending on the current state

and action. An important property of MDPs is that the transition probabilities are independent of past states and actions. This property simplifies theoretical analysis of reinforcement learning.

The behavior of an agent is described in terms of a policy $\pi : S \rightarrow A$. $\pi(s)$ is the action chosen by the agent in state s . Applying policy π in state s , the agent will get an immediate reward of $r(s, \pi(s))$ and the environment will randomly change to another state s' with probability $p_{s,s'}^{\pi(s)}$. If we continue applying policy π , we will observe a sequence of states and rewards. The goal of reinforcement learning is to find a policy that maximizes the expected sum of rewards over time:

$$\underset{\pi}{\text{maximize}} \quad E\left[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s, \pi\right] \quad (1)$$

where r_t denotes the reward that is achieved in the t -th step and $\gamma \in [0, 1)$ is a discount factor that is introduced to guarantee convergence of the infinite sum in (1). It is typically chosen close to 1³.

Using the Markov property of the MDP, we can unroll the sum in (1) and get a fixed point equation which is known as Bellman equation:

$$E\left[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s, \pi\right] = r(s, \pi(s)) + \gamma \sum_{s' \in S} \left(p_{s,s'}^{\pi(s)} E\left[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s', \pi\right]\right) \quad (2)$$

It has been shown [8] that for each MDP there exist policies π^* that maximize $E\left[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s\right]$ for all states $s_0 \in S$ at the same time. These policies are called *optimal policies*. Hence, the goal of reinforcement learning is to find an optimal policy.

2.2 Value Iteration

One way to calculate an optimal policy is first to determine the optimal expected reward and derive an optimal policy afterwards. The optimal expected reward is described in the form of a *value function* $V^* : S \rightarrow \mathbb{R}$. $V^*(s) = E\left[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s, \pi^*\right]$ denotes the accumulated expected reward if we apply an optimal policy π^* starting in state s .

Rewriting the Bellman equation (2) and taking into account that the optimal policy always chooses the action having the maximal expected reward, we get a fixed point equation for V^* :

$$V^*(s) = \max_{a \in A} \left(r(s, a) + \gamma \sum_{s' \in S} (p_{s,s'}^a V^*(s')) \right) \quad (3)$$

The right hand side of (3) can be interpreted as an operator that maps value functions onto value functions. It turns out that this operator is a contraction mapping in the space of value functions [4]. Thus, applying Banach's fixed point

³ variants of this optimization goal exist, see [4]

theorem, we can approximate the optimal value function V^* starting with an arbitrary initial value function V_0 and perpetually applying the operator defined by the right hand side of (3).

Furthermore, having once calculated V^* , we can derive an optimal policy π^* from V^* using a greedy evaluation scheme for the value function:

$$\pi^* : s \mapsto \arg \max_{a \in A} \left(r(s, a) + \gamma \sum_{s' \in S} (p_{s, s'}^a V^*(s')) \right) \quad (4)$$

The idea of using an iterative process to calculate the optimal value function and afterwards greedily deriving an optimal policy is implemented by a reinforcement learning algorithm called *value iteration* [4]. The value function is stored in the form of a table. In order to perform the update steps of value iteration, the transition probabilities $p_{s, s'}^a$ and the reward function $r(s, a)$ must be known.

2.3 Q-Learning

The main disadvantage of the value iteration approach is the fact that the state transition probabilities and the reward function must be known in advance. In real world applications of reinforcement learning, e.g. in autonomous robot control, these values are typically not known so that value iteration cannot be applied. To overcome this problem, a learning algorithm named Q-learning has been developed that is not based on this knowledge but that is able to implicitly estimate these values while interacting with the environment and observing transitions.

The main idea is to introduce value functions Q^* that depend on both the current state and the action that potentially might be chosen by the agent. $Q^*(s, a)$ models the expected reward of an agent that starts in state s , chooses action a first and acts optimally later on, i.e.:

$$Q^*(s, a) = r(s, a) + \gamma \sum_{s' \in S} (p_{s, s'}^a V^*(s')) \quad (5)$$

Using this definition, we can rewrite the Bellman equation (3) as:

$$Q^*(s, a) = r(s, a) + \gamma \sum_{s' \in S} (p_{s, s'}^a \max_{a' \in A} Q^*(s', a')) \quad (6)$$

which yields a fixed point equation for Q^* instead of V^* . The optimal policy can be derived directly from the Q^* -function using greedy evaluation:

$$\pi^* : s \mapsto \arg \max_{a \in A} Q^*(s, a) \quad (7)$$

If we knew the transition probabilities and the reward function, we could apply a dynamic programming-like algorithm similar to value iteration to approximate Q^* . In contrast, the *Q-learning* algorithm [20] does not assume these

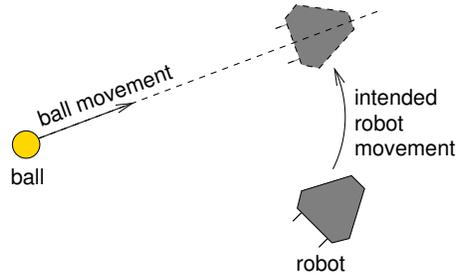


Figure 1. Intercepting a rolling ball. The robot must move to a point on the line that is given by the ball movement and must turn to face the ball. To intercept the ball, the difference in velocity between the robot and the ball must also be small.

values to be known in advance. Instead, by observing state transitions of the environment, the algorithm collects quadruples of predecessor state, action, reward and successor state and it approximates the Q^* function using a stochastic approximation scheme. The Q -function is updated incrementally every time a state transition is observed.

After having observed a transition from state s to state s' using action a and obtaining reward r , the Q -function is updated using the following rule:

$$Q(s, a) \leftarrow \alpha(r + \gamma \max_{a' \in A} Q(s', a')) + (1 - \alpha)Q(s, a) \quad (8)$$

where $\alpha > 0$ is a learning rate decreasing over time. Q -learning is guaranteed to converge towards the optimal Q^* -function as long as all combinations of states and actions are observed repeatedly and the decrease of the learning rate fulfills certain conditions [4].

The advantage of Q -learning in comparison to value iteration is the fact that the optimal policy can be learned only by interacting with the environment. No knowledge of the true transition probabilities or the reward function is necessary. Hence, it can also be applied to real-world tasks with unknown system dynamics. Similar to value iteration, the Q -function is typically stored in a table-based representation which can become very large if S and A are large.

3 Learning on a Real Robot

3.1 Robot Learning Task

The task discussed in this paper is the problem of an autonomous mobile soccer robot intercepting rolling ball, see Fig. 1. Intercepting means that the robot is moving to a certain point where it can interrupt the current ball movement and can get control of the ball. In order to do this, the robot must touch the ball with its front side and the difference in robot and ball velocities must be small, i.e. less than $0.6 \frac{m}{s}$. In this study, we are only interested in situations like the

one depicted in Figure 1 in which the ball moves towards the robot. Situations, in which the ball rolls away from the robot are of no interest.

The robot is assumed to be able to recognize its environment with a camera so that it can determine the ball's position and the ball velocity. The interpretation of the camera images and the estimation of the ball velocity are assumed to be done by existing sensor processing [11] and are not subject of this work. Furthermore, we assume that the robot is able to move in any direction without turning in advance.

For our experiments, we used an already existing soccer robot of the RoboCup middle size league team *Brainstormers Tribots* [7]. The robot is 80cm large, 40cm wide (see Fig. 2) and has a holonomic drive with three degrees of freedom, so that it can move in all directions and turn simultaneously. Its maximal velocity is $2.5 \frac{m}{s}$. The robot is driven by three electric motors that are individually controlled by PID-controllers. To recognize the ball, the robot is equipped with a catadioptric camera that allows a 360° view of its surroundings. With the help of this camera system, it can recognize objects up to a distance of 6m, e.g. a ball rolling on the ground.



Figure 2. A robot of the *Brainstormers Tribots* team that was used for our experiments

3.2 Modeling the Learning Task

In order to apply reinforcement learning algorithms to the intercept problem, we must describe the task in terms of an MDP. The state space of the intercept problem consists of the following variables:

- position of the robot (2-dimensional)
- orientation of the robot (1-dimensional)
- linear velocity of the robot (2-dimensional)
- angular velocity of the robot (1-dimensional)
- ball position (2-dimensional)
- linear velocity of the ball (2-dimensional)

In summary, the state space is described by 10 variables. Since the pose of the robot with respect to some global coordinate system is irrelevant for the intercept task, we can omit the robot pose variables and represent the ball position relative to the robot position. Hence, we are left with seven state variables. Since even this number of variables is still too large for reinforcement learning algorithms, we further simplified the problem using a standard orientation controller that always turns the robot such that it faces the ball. Therefore, this orientation controller becomes part of the environment. By aligning the coordinate system with the direction of the ball movement, we can omit the state variables that describe the direction of the ball movement and the angular velocity of the robot so that we are left with a five-dimensional state space. However, the state space remains infinite.

Due to the holonomic drive of the robot, the complete action space contains accelerations between 0 and a maximal value in any direction. However, reinforcement learning approaches only allow finite and – typically – very small sets of actions. Therefore, we use a discretization of the complete action space, i.e. the robot is allowed to accelerate to one of eight directions organized in 45° angles, see Fig. 3. The amount of acceleration used is maximal with respect to the acceleration capabilities of the robot. After selecting one of the eight actions, the setpoints of the motor controllers are updated appropriately to achieve the desired acceleration. As mentioned above, the orientation of the robot is controlled by a standard controller so that we do not need to learn actions in order to turn the robot.

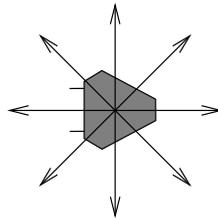


Figure 3. The eight directions of acceleration

To achieve an MDP we need to model the temporal processing of the robot control task carefully, since we have to guarantee that the Markov property is met. Due to delays in the motor controllers, inertia, and delays in camera

image processing, an action is not executed immediately but with a certain delay, and its consequences can only be observed after a certain amount of time. Measurements have shown that the execution of an action needs approximately $240ms$, which is much longer than the control cycle of our software framework which amounts to $40ms$.

If we had decided on a new action every $40ms$, the subsequent actions would not have been executed completely and the Markov property would have been violated. This problem has also been described in [3,5]. While there, a state prediction has been used based on physical motion models of the robot and the ball [10,11] to close the time gap between selection and execution of an action, here, we extended the time between two subsequent actions from $40ms$ to $240ms$. Hence, when a subsequent action is selected, the consequences of the prior action have already become part of the state variable. By doing so, we again meet the Markov property.

The reward function for the learning task was designed to achieve a robot behavior that is optimal in an intuitive sense, i.e. the robot should get possession of the ball as quickly as possible. Therefore, the reward function yields a reward of -0.2 for every step to punish the robot for wasting time and a reward of $+500$ if the robot gets control of the ball, i.e. it touches the ball at its front side and the difference between the velocity of the robot and that of the ball is no larger than $0.6\frac{m}{s}$. The discount factor is set to $\gamma = 0.92$ throughout all experiments.

3.3 Value Function Approximation

The state space of the intercept problem has been defined as a five-dimensional space. Hence, there are infinitely many states, and a table-based representation of value functions or policies is no longer possible. Moreover, observing transitions from each possible state is also not possible. Thus, we need to generalize over subsets of states to obtain policies for all states and use approximators to represent the value functions [2].

Although it has been shown that the convergence properties of reinforcement learning algorithms might be lost in some cases [12] and that only for very special situations can convergence be guaranteed [18], this approach has been investigated in many studies on reinforcement learning. In particular, linear function approximators with nonlinear features have been applied and have shown good performance.

Hence, we used two kinds of linear function approximators within this study, grid maps and lattice maps. Grid maps implement a piecewise constant function on the basis of some regular tessellation of the input space (see Fig. 4 left). The resulting function is a step-function. The height of each step is determined by the average value of all training examples located within the respective cell of the input space. To achieve an appropriate degree of precision, the number of cells must be very large and training examples must be located in each cell. Using grid maps is equivalent to a discretization of the input space.

Lattice maps are based on a tessellation of the input space into simplices using Kuhn triangulation [13]. The resulting function is defined individually for

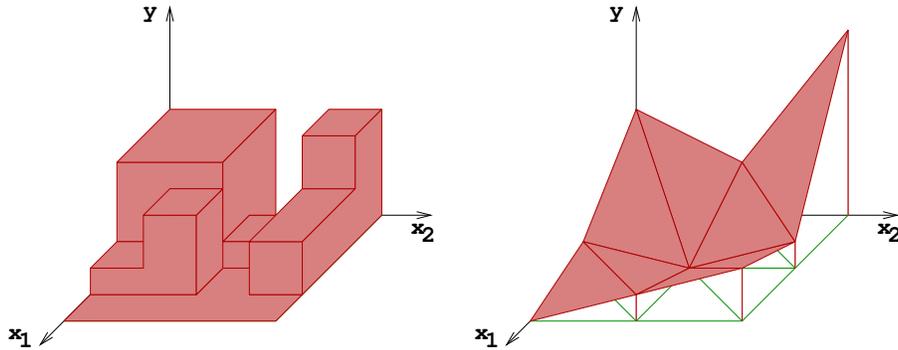


Figure 4. Example of a grid map (left) and a lattice map (right) over a two-dimensional feature space.

each vertex of the simplices and is interpolated between the vertices. By doing so, the resulting function is continuous and piecewise linear (see Fig. 4 right). Compared to grid maps, the number of simplices used for lattice maps might be much smaller than the number of cells in grid maps of the same precision. Hence, the memory requirements are smaller, the generalization performance is better and the number of training examples necessary is smaller. However, it has been shown that lattice maps may become instable using temporal difference updates and have a higher risk of not converging when compared to grid maps [12]. Fortunately, it has been possible to derive constraints under which the lattice maps remain stable and eventually converge [15]. To train the lattice maps, we used the Kaczmarz update rule [14] which turns out to be more stable than a gradient descent update.

3.4 Training the Robot

Since reinforcement learning is based on learning from experience, we needed to collect data from experiments with our robots. Unfortunately, all reinforcement learning algorithms need millions of examples to perform adequately. Collecting these examples on real robots is not possible since the robots would break. Therefore, we built a simulator tool that implements a simple physical model of the soccer scenario. It allowed us to generate a huge amount of training samples. However, since the simulation is only a crude image of the real soccer scenario, it is necessary to evaluate the results on real robots, as well.

Several experiments have been done on learning how to intercept a rolling ball. First, the policies were learned in simulation and afterwards evaluated in a simulation and on the real robot. Experiments were made using Q-learning combined with grid maps and lattice maps for approximation of the value function.

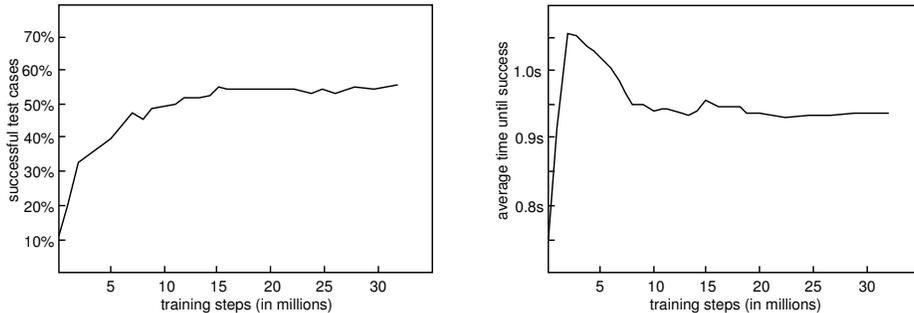


Figure 5. Left: success rate for Q-learning with grid map on test cases dependent on the number of training steps. Right: average time needed to get control of the ball in the successful test cases.

4 Experimental Results

4.1 Grid Map

In the first experiment, we analyze whether it is even possible to learn the intercept problem with Q-learning. We have chosen the grid map in this experiment to get rid of problems of non-convergence of function approximators. Unfortunately, it turns out that with a five-dimensional feature space the size of the grid map chosen must be very large in order to achieve an approximation with sufficient precision. Therefore, we restricted the experiments to a very limited working area: the ball had to be located within a wedge of $1.5m$ radius and 60° angle in front of the robot. The robot velocity was restricted to $1 \frac{m}{s}$, the ball velocity to $0.5 \frac{m}{s}$. However, the resulting grid partitioned the state space into 225,000 cells. Considering each of the eight possible actions, the Q-function was represented by 1.8 million cells in total. For each cell, we had to train one parameter of the function approximator.

The training was done in the simulator described in section 3.4. If the ball left the working area, a trajectory was finished and a new starting point for the ball within the working area was randomly chosen. The robot followed an ϵ -greedy exploration strategy by selecting with a probability of 0.8 the action that seemed to be optimal concerning the Q-function learned yet and with probability $\epsilon = 0.2$ a random action (exploration rate). The learning rate α decreased over time from 0.4 to 0.01.

Periodically, the performance of the learned policy was evaluated on a test set of 50,000 starting points uniformly distributed in the working area. The robot had a maximal time of 3 seconds to get control of the ball.

Figure 5 shows the percentage of successfully solved test cases dependent on the number of training steps. After 15 million training steps, Q-learning reached its best performance with a success rate of 55%. From other experiments, we know that for at least 70% of all test cases it is possible to intercept the ball. However, since the test cases are generated randomly, there are cases in which

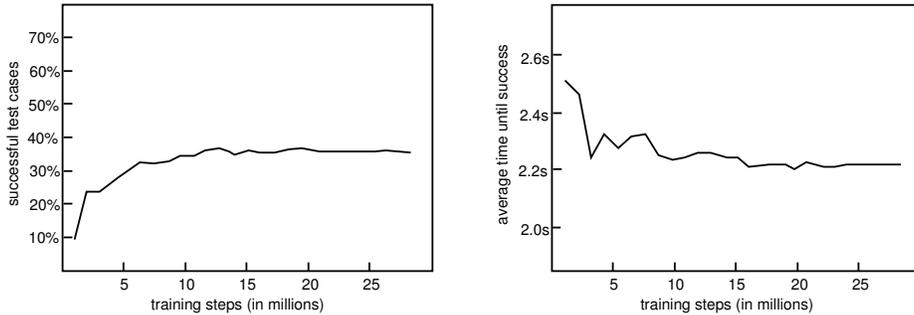


Figure 6. Left: success rate for Q-learning with lattice map on test cases dependent on the number of training steps. Right: average time needed to get control of the ball in the successful test cases.

no robot control strategy can be successful, i.e. a success rate of 100% is not possible. The right hand plot in Fig. 5 shows the time needed to get control of the ball in all successful test cases. By continuous optimization of the learned strategy the robot is able to decrease the necessary time from 1.05s to 0.95s on average.

4.2 Lattice Map

Although the experiments with the grid map showed the general possibility of learning the intercept problem using reinforcement learning techniques, the results are not convincing. Neither the measured performance nor the limited working area of this approach is suitable for the application of the learned strategy on a real robot. Therefore, we repeated the experiments with a lattice map function approximator instead of a grid map. Due to the better approximation performance of lattice maps, we could extend the working area to a wedge in front of the robot with radius $5m$ and an angle of 90° . The maximal robot velocity was increased to $2.5 \frac{m}{s}$, the maximal ball velocity to $3.5 \frac{m}{s}$.⁴

The lattice map had 161,568 grid points per action, with a total of 1.3 million parameters for all actions, which is smaller than in the grid map case although the working area is larger by a factor of 87.5. To avoid non-convergence of the lattice map during training, we did not consider whole trajectories but only single transitions from one state to another where only grid points were used as starting state. Using a simulator, this could be achieved easily.⁵

Figure 6 shows the results of the lattice map approach. The success rate is smaller than in the grid map case and achieves at most 36% after 13 million training steps. However, this can be explained by the fact that a larger working

⁴ Please note, although the ball initially may roll faster than the robot can move its velocity constantly decreases, thus possibly allowing a soft interception.

⁵ When *exploiting* the learned strategy, e.g. on the real robot, no such constraint has to be fulfilled.

area was used with ball velocities up to $3.5 \frac{m}{s}$ that are very hard to intercept for any strategy. To illustrate that the performance of the learned strategy is good, we introduced a second criterion to measure whether a strategy is able to touch the ball. Figure 7 shows the learning results using this ball contact criterion. The policy learned achieves a 70% success rate. In contrast, a simple strategy that always drives the robot into the direction of the ball with maximal velocity only achieves a success rate of 63%.

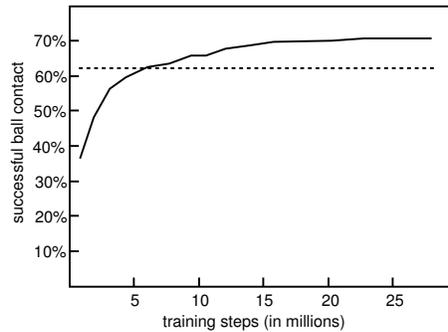


Figure 7. Success rate in test cases with respect to the ball contact criterion. The solid line shows the success rate for the robot behavior learned using Q-learning and a lattice map, the dashed line shows the success rate of a simple robot that always drives the robot into the direction of the ball with maximal velocity.

4.3 Learning for the Real Robot

The experiments described in section 4.1 and 4.2 have been made in a simulation environment without considering any delay in sensors and actuators which typically can be found on real robots (cf. section 3.2). To obtain results that are closer to reality and that can be transferred to a real robot, we repeated the experiments with the lattice map incorporating an artificial delay in the simulator.

Compared to the experiments without delay, the success rates of intercepting decreased from 36% to 18%, the success rate of ball contacts decreased from 70% to 66%, while the average time needed to intercept the ball remained almost the same. The tremendous influence of delays also is illustrated by the fact that the success rate in ball contacts of the simple reference strategy that always drives the robot into the direction of the ball decreased from 63% to 31%. A lot of these failures were caused by the robot bumping too hard into the ball, thus violating the “soft interception” constraint and causing the ball to bounce away.

Using the simulator with delays enabled us to learn in simulation a strategy that can be transferred to the real robot. In order to do this, we integrated

the learned strategy into the software framework of our real robots. Information about the environment such as robot pose, robot velocity, ball position, and ball velocity are estimated and provided within a central world model by the existing software framework. The control of the motors is implemented there as well.

In contrast to simulation, the information available on the real robot is much noisier and less reliable. Figure 8 shows position estimates of the robot and the ball in simulation and on real robots for comparable situations. Certainly, the high noise level disturbs the intercepting strategy and increases the task complexity.

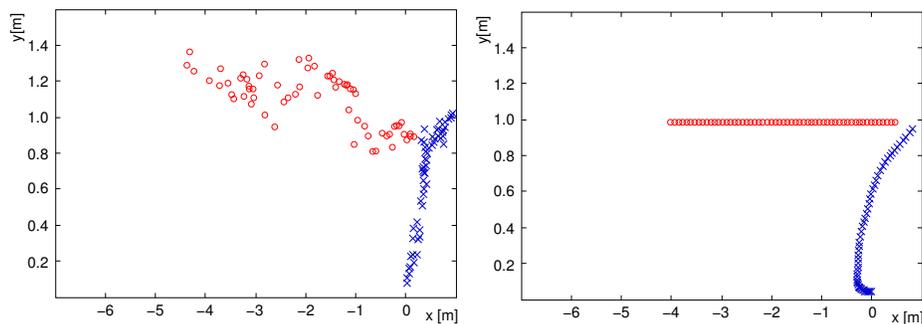


Figure 8. Comparison of the estimated robot (crosses) and ball (circles) positions in simulation (right) and on the real robot (left) for two comparable situations. The noise level is much higher on real robots than in simulation.

To test the performance of the intercept strategy learned, we performed 30 experiments in our laboratory where the ball was rolling down a ramp to achieve a certain velocity. Three different ball velocities ($0 \frac{m}{s}$, $1 \frac{m}{s}$ and $2 \frac{m}{s}$) and three different geometric configurations of initial robot and ball position were used. In 10 out of 30 experiments the robot succeeded in intercepting the ball according to the “soft interception” condition, constraining the difference of ball velocity and robot velocity in the moment of the contact. In most of the other experiments it moved towards the right direction but bounced the ball away.

To gain experience from the performance of the learned behavior in more complex and more realistic situations, we integrated it into the competition code of our RoboCup middle size league team the *Brainstormers Tribots*. The strategy of the robots is realized in a behavior-based architecture fusing principles of BDI-like architectures with ideas from the subsumption architecture. The whole strategy of the robots is realized by a number of several smaller submodules, named *behaviors*, and a multi-level hierarchical arbitration mechanism. Within a level of the hierarchy the arbitrator uses a priority ordering of the available behaviors and logical constraints connected to each of the behaviors to decide which behavior should become active in a particular situation.

The intercept behavior learned has been integrated into a larger submodule realizing all approaches to the ball. Whereas the learned behavior is activated only in situations where the ball rolls with a certain velocity and angle *towards* the robot, several other handcoded strategies solve the easier situations where the ball moves away from the robot or lays still on the ground.

We used the embedded learned strategy throughout the world championships 2006 in Bremen. On average, during a game of 30 minutes duration, the intercept behavior was used approximately 30 times per robot. The embedding handcoded behaviors were able to quickly get control of the slowly moving ball in many of the cases the interception strategy failed and bounced the ball away. Although statistics on the success rate of a single behavior in a complex real game is misleading, the entire robot behavior and especially the ability to approach the ball faster and more reliable than any other team was the most important factor in winning the RoboCup World Championship 2006 in the middle size league.

5 Discussion

So far, reinforcement learning has been applied so far primarily in simulated environments, while its application to real autonomous robots has been limited by a set of factors in which a simulated world differs from a real application, e.g. sensor and actuator delays which are in conflict with the Markov property, limited possibilities for training on the real robot due to hardware constraints, large state spaces and noisy state estimation procedures.

By means of the ball intercepting problem, we have exemplified how these problems can be tackled and how strategies for real robots can be learned successfully. In order to do so, we have generated a modeling of the problem that is consistent with the theoretical needs of reinforcement learning but also covers the real world demands of autonomous robots. Using lattice maps as function approximators to represent the Q-function and combining them with the Q-learning algorithm and a simulator to generate training examples, successful policies could be learned.

In experiments using the simulator, we showed the principle applicability of reinforcement learning to the intercept problem and improved the modeling using lattice maps and an extended working area of the learned intercept routine. Finally, by transferring the intercept strategy learned to the real robot and integrating it into a larger software framework, we were able to test the behavior learned on a real robot and compare the results to the simulated test results. Moreover, we integrated the learned intercept strategy into the tournament code of our soccer robots and became world champion.

Acknowledgments

This work was supported by the German Research Foundation DFG SPP 1125.

References

1. M. Asada, S. Noda, S. Tawaratsumida, and K. Hosoda. Vision-based reinforcement learning for purposive behavior acquisition. In *Proc. of IEEE Int. Conf. on Robotics and Automation*, pages 146–153, 1995.
2. Leemon C. Baird. Residual algorithms: Reinforcement learning with function approximation. In *Proceedings of the 12th International Conference on Machine Learning*, pages 30–37, 1995.
3. Sven Behnke, Anna Egorova, Alexander Glove, Raúl Rojas, and Mark Simon. Predicting away robot control latency. In *RoboCup 2003: Robot Soccer World Cup VII*, pages 712–719, 2003.
4. D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1996.
5. Thomas Gabel, Roland Hafner, Sascha Lange, Martin Lauer, and Martin Riedmiller. Bridging the gap: Learning in the robocup simulation and midsize league. In *Proc. 7th Portuguese Conference on Automatic Control (Controlo 2006)*, 2006.
6. Thomas Gabel and Martin Riedmiller. Learning a partial behavior for a competitive robotic soccer agent. *Künstliche Intelligenz*, 20(2):18–23, 2006.
7. Roland Hafner, Sascha Lange, Martin Lauer, and Martin Riedmiller. Brainstormers Tribots team description. In *RoboCup-2006*, 2006.
8. R.A. Howard. *Dynamic programming and Markov processes*. MIT Press, 1960.
9. Hiroaki Kitano, Minoru Asada, Yasuo Kuniyoshi, Itsuki Noda, Eiichi Osawa, and Hitoshi Matsubara. RoboCup: A challenge problem for AI. *AI Magazine*, 18(1):73–85, 1997.
10. Martin Lauer. Ego-motion estimation and collision detection for omnidirectional robots. In *RoboCup 2006: Robot Soccer World Cup X*, 2006.
11. Martin Lauer, Sascha Lange, and Martin Riedmiller. Motion estimation of moving objects for autonomous mobile robots. *Künstliche Intelligenz*, 20(1):11–17, 2006.
12. Artur Merke and Ralf Schoknecht. A necessary condition of convergence for reinforcement learning with function approximation. In *Proceedings of the 19th International Conference on Machine Learning*, pages 411–418, 2002.
13. Remi Munos and Andrew Moore. Variable resolution discretization for high-accuracy solutions of optimal control problems. In *International Joint Conference on Artificial Intelligence*, pages 1348–1355, 1999.
14. Stephan Pareigis. Adaptive choice of grid and time in reinforcement learning. In *Advances in Neural Information Processing Systems 10*, pages 1036–1042, 1997.
15. Ralf Schoknecht and Artur Merke. Convergent combinations of reinforcement learning with linear function approximation. *Advances in Neural Information Processing Systems*, 15, 2003.
16. Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning : An Introduction*. MIT Press, 1998.
17. S. Suzuki, T. Kato, M. Asada, and K. Hosoda. Behavior learning for a mobile robot with omnidirectional vision enhanced by an active zoom mechanism. In *Proc. of Intelligent Autonomous System 5(IAS-5)*, pages 242–249, 1998.
18. J. N. Tsitsiklis and B. Van Roy. Analysis of temporal-difference learning with function approximation. In *Advances in Neural Information Processing Systems 1996*, pages 1075–1081, 1996.
19. Eiji Uchibe, Minoru Asada, and Koh Hosoda. Behavior learning for a mobile robot with omnidirectional vision enhanced by an active zoom mechanism. In *Proc. of Sixth European Workshop on Learning Robots (EWLR-6)*, 1997.
20. C.J.C.H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8:279–292, 1992.