Approximating the Value Function for Continuous Space Reinforcement Learning in Robot Control

Sebastian Buck, Michael Beetz, and Thorsten Schmitt

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

Abstract

Many robot learning tasks are very difficult to solve: their state spaces are high dimensional, variables and command parameters are continuously valued, and system states are only partly observable. In this paper, we propose to learn a continuous space value function for reinforcement learning using neural networks trained from data of exploration runs. The learned function is guaranteed to be a lower bound for, and reproduces the characteristic shape of, the accurate value function. We apply our approach to two robot navigation tasks, discuss how to deal with possible problems occurring in practice, and assess its performance.

1 Introduction

Many autonomous robot skills are difficult and tedious to hand code. Preferably, such skills should be learned automatically by the robot — for instance, through experience-based and reinforcement learning techniques. Unfortunately, many of these learning tasks in the context of skill acquisition are very difficult to solve: their state spaces are high dimensional, variables and command parameters are continuous valued, and system states are only partly observable. This situation calls for pragmatic solutions to robot learning problems.

Learning to dribble in autonomous robot soccer provides a good case in point. Let us consider the following problem: A soccer robot in possession of the ball has to dribble around an opponent defender. The opponent robot wants to hinder it and tries to get the ball (Fig. 1). An appropriate state space for reinforcement learning [16] in this environment must include distances, angles, and velocities of both robots. Since acceleration usually is nonlinear discretizing the velocity values will be a problem. Determining the neighborhood of a discrete state in a, let us say, eight-dimensional state space becomes incredible complex. Moreover, the size of discrete states grows exponentially with the number of dimensions which means we cannot visit each state during exploration. Further discretization may lead to extremely discontinuous behavior.

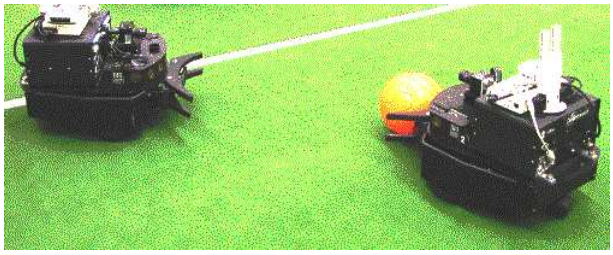

Figure 1: *Soccer robots in a game situation.*

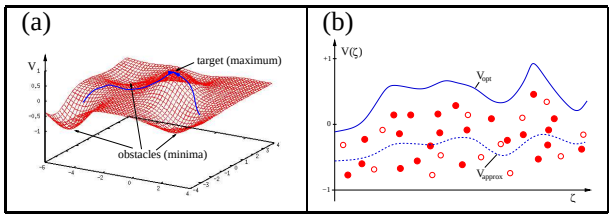

Figure 2: *(a): Navigating through state space using the gradient. The maximum of the value function corresponds to the target state. (b): The generalized function ($\mathcal{V}_{approx}$) is a lower bound for the optimal value function $\mathcal{V}_{opt}$ and has a shape similar to it.*

Reinforcement learning has been shown to reliably solve a number of robot control problems. A lot of work has been done on reinforcement learning in high dimensional state spaces, and on learning in continuous state spaces too. The development of behavioral models for external components has been surveyed as well. But to develop a reinforcement learning solution that reliably solves complex robot control problems in a highly dynamic, high dimensional, and non-discretized state space including the behavior of intelligent external components is still a big challenge.

Our contributions in this paper are (1) the non-incremental learning of a value function that approximates an upper bound of the *temporal distance* (the time needed) of any continuous state to the target state and (2) a neural projection of the successor state given any action in the current state. Using (1) and (2) we simply choose the action leading to the successor state getting the best value (Fig. 2(a)).

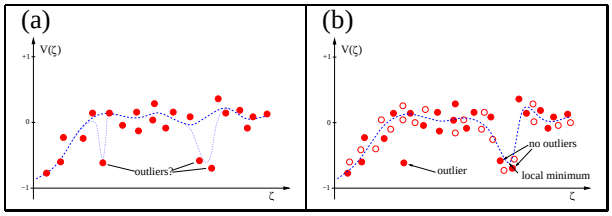

Figure 3: *Detecting outliers in state space: (a): Which measured values are outliers? (b): The dashed line is learned from training data (filled circles) using validation data (not filled circles). Outliers are identified.*

The remainder of the paper is organized as follows: In section 2 we explain our approximation method in detail and provide advice for practical application. Section 3 presents results of applying our method to two problems in robot control. An overview of related work is given in section 4. Finally in section 5 we close with conclusions and future work.

2 Approximating the Value Function

2.1 Overview

As mentioned in the introduction we want to learn a (temporal) distance metric in the state space $\mathcal{S}$. According to the syntax of a value function we call the optimal metric $\mathcal{V}_{opt}$ and define that $+1$ is the best and -1 the worst value. To have an upper bound for the time to reach the target state (or the target space) from a certain start state we only have to navigate there through state space and record the time needed. An upper bound for the time corresponds to a lower bound for $\mathcal{V}_{opt}$. As we navigate through states $\zeta(t)$ at time t until we arrive at the target space $\mathcal{S}_{target}$ we can assign values to these states:

$$\mathcal{V}_{ass}(\zeta(t)) = \begin{cases} +1 \;\; \textit{if} \;\; \zeta(t) \in \mathcal{S}_{target} \\ -1 \;\; \textit{if} \;\; \mathcal{S}_{target} \;\; \textit{cannot be reached} \\ \gamma * \mathcal{V}_{ass}(\zeta(t+1)) \;\; \textit{else} \end{cases} \tag{1}$$

If it is not possible to reach $\mathcal{S}_{target}$ from $\zeta(t)$ in any way we penalize $\zeta(t)$ with the value $\mathcal{V}_{ass}(\zeta(t)) = -1$ and its predecessor states (with $\gamma \in (0,1)$ being a discount factor). If we reach $\mathcal{S}_{target}$ at time t we assign $\mathcal{V}_{ass}(\zeta(t)) = 1$ to $\zeta(t)$ and the values of the predecessor states of $\zeta(t)$ are increased too. Thus we get patterns

$$\langle \zeta(i), \mathcal{V}_{ass}(\zeta(i)) \rangle \qquad \zeta(i) \in \mathcal{S}, \mathcal{V}_{ass}(\zeta(i)) \in [-1,+1] \tag{2}$$

which we can use for learning a lower bound for $\mathcal{V}_{opt}$. The idea is that having a sufficient number of those patterns (unlimited in theory but a reasonable number in practice) we will be able to approximate a function $\mathcal{V}_{approx}$ whose local extrema are located at the same points in state space as those of $\mathcal{V}_{opt}$ (see figure 2(b)). This approach not only guides us towards the target space but allows us to define states (or spaces) we definitely do not want to visit (the states assigned with -1). If we randomly explore the state space a point near the target space will more likely get a good value than a point near a state we do not want to visit.

As proposed by Bertsekas and Tsitsiklis [3] we use multi layer artificial neural networks for the approximation of our value function. For updating the networks' weights we use the RPROP algorithm [13]. Using a training and a validation set of patterns we are able to detect if a low measured value corresponds to a local minimum in state space or if the exploration was a detour (see figure 3).

The computational amount of learning is $O(n^d)$ where n is the amount for the sufficient exploration of one dimension and d is the number of dimensions of the state space. However, as the virtual size of the state space grows exponentially the amount needed for learning in that space does as well.

Assuming we have managed to learn the value function

$$\mathcal{V}_{approx} : \begin{cases} \mathcal{S} \to [-1,+1] \\ \zeta(i) \mapsto \mathcal{V}_{approx}(\zeta(i)) \end{cases} \tag{3}$$

we need to know which action $a(i)$ in a given state $\zeta(i)$ leads to the best successor state $\zeta(i+1)$. This is done in two steps: First, we heuristically generate a number of possible actions $\mathcal{A}_{poss}$[1]. Thereafter we predict the supposed successor state for each possible action and choose the action $a_{ch}(i)$ corresponding to the best-valued successor state. Thus our policy π is defined

$$\pi : \begin{cases} \mathcal{S} \to \mathcal{A} \\ \zeta(i) \mapsto a_{ch}(i) \end{cases} \tag{4}$$

where

$$a_{ch}(i) = \arg\max_{a(j) \in \mathcal{A}_{poss}} \mathcal{V}_{approx}(\mathcal{P}(\zeta(i), a(j))) \tag{5}$$

Herein

$$\mathcal{P} : \begin{cases} \mathcal{S} \times \mathcal{A} \to \mathcal{S} \\ \langle \zeta(i), a(i) \rangle \mapsto \zeta(i+1) \end{cases} \tag{6}$$

is a one-step projection learned from real robot data recorded during exploration. $\mathcal{P}$ is learned using neural networks once again.

2.2 Practical Application

In this section we address problems which may occur in the practical application of the approach described above. All problems mentioned here also occur when using other algorithms or are general problems in reinforcement learning.

What happens for example if we define a number of states with minimal value (-1) around the target state? This could correspond to a robot which is to navigate through

[1] This can be done by an equidistant covering of the action space. Interpolation between different actions is possible too.

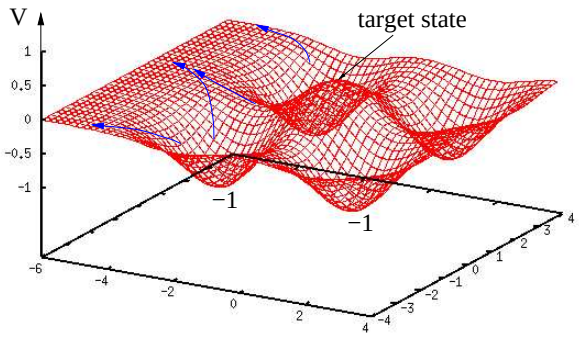

Figure 4: *The target state is surrounded by states with minimal value. Assuming the state space is uniformly explored it may not be possible to find a way to the target state unless the current state is near the target state.*

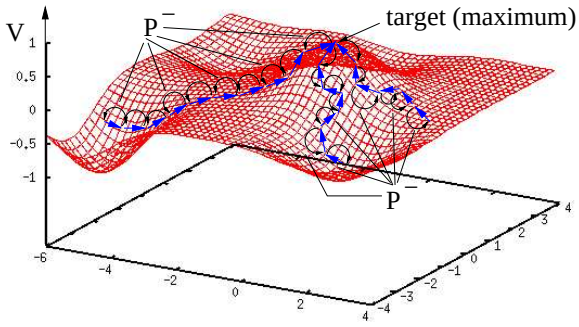

Figure 5: *Using the inverse projection $\mathcal{P}^-$ the state space can be explored starting at the target state.*

a narrow passage. The resulting value function may have local maxima and the robot may not find a way to its target avoiding the states with negative values (see figure 4). This problem is similar to problems occurring in path planning using potential fields [2]. In addition to numerous variations of the potential field method that overcome the problem of local maxima we propose a special solution for that problem in our context: In addition to $\mathcal{V}_{approx}$ we learn a second value function $\mathcal{V}^+_{approx}$ using only patterns $\langle \zeta(i), \mathcal{V}(\zeta(i)) \rangle$ with $\mathcal{V}(\zeta(i)) > 0$. That means we regard only those experiments of our exploration that reached the target. Considering $\mathcal{V}^+_{approx}$ we can detect local maxima because $\mathcal{V}^+_{approx}$ is a convex function ($\mathcal{V}^+_{approx}$ is $\mathcal{V}_{approx}$ without repellent states).

We can imagine another disadvantageous scenario: What happens if we never reach the target state during exploration? That means $\mathcal{V}_{approx}(\zeta(i))$ will be less than zero for all states $\zeta(i)$ (we cannot assign positive values to states). To overcome this problem we can learn the inverse projection

$$\mathcal{P}^- : \begin{cases} \mathcal{S} \times \mathcal{A} \to \mathcal{S} \\ \langle \zeta(i), a(i-1) \rangle \mapsto \zeta(i-1) \end{cases} \tag{7}$$

mapping a state and the last chosen action to the predecessor state. Applying this mapping recursively from the target state we can explore the state space starting at the target state (see figure 5). In this case we can assign positive values to states.

Furthermore there is the question if the state space was covered sufficiently during exploration. As proposed before for the discrete case [10] we can explore regions of interest more in detail.

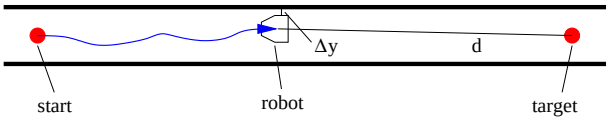

Figure 6: *The corridor following task: A robot must be navigated from a start to target state in a corridor using minimal time.*

3 Experimental Results

In order to show that our approach can reliably be used for several robot control tasks we apply it to two concrete problems. Due to efficiency all exploration is done in simulation (setting up hundreds of exploration runs in a real world environment is next to impossible). In our case we use the M-ROSE [5] simulation environment which has been shown to be an accurate means for simulating the dynamical behavior of Pioneer robots.

While the first chosen task is supposed to be a simple problem and convex in all dimensions the second task is a complex non-convex problem.

3.1 The Corridor-Following Task

The corridor-following task involves learning to steer a robot down a corridor towards a target point and has been used before by Smart and Kaelbling [14]. The robot uses a laser range-finder to localize itself on the corridor. The state space

$$\mathcal{S} = \langle d_{target}, \Delta\varphi, \Delta y \rangle \tag{8}$$

consists of the distance to the target state (d_{target}), the orientation of the robot relative to the corridor ($\Delta\varphi$), and the minimal distance of the robot to the wall (Δy). Figure 6 shows a possible scenario. The objective of the navigation policy π is to minimize the amount of time for reaching the target state by choosing an appropriate rotational velocity (in degree per second) while the translational velocity of the robot is fixed.

In our case the distance between start and target state is 8 meters while the width of the corridor is 0.8 meters and the diameter of the robot 0.4 meters. During exploration we chose a random rotational velocity $V_{rot} \in \mathrm{N}(0, 30)$ (N is the Gaussian distribution). We change the velocity after a random time. Thus the action space $\mathcal{A}$ is $[-180, +180]$. $\mathcal{V}_{ass}(\zeta(i))$ is set to -1 if the robot hits the wall (terminal state), to $+1$ if it reaches its target (terminal state), and to $\gamma * \mathcal{V}_{ass}(\zeta(i+1))$ else (non-terminal state).

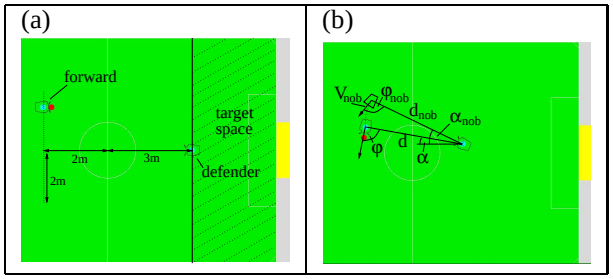

Figure 7: *The dribble task. (a): the initial setting of the task. The forward has to dribble around the defender to reach the target space. (b): Some features of the state space.*

In Application we choose

$$V_{rot} \in \{-180, -170, \ldots, 0, \ldots, +170, +180\} \quad (9)$$

and evaluate all 37 possible successor states at a frequency of 10Hz. In practice we achieve a success rate of 100% in reaching the target while in the ten runs of exploration the success rate was only 20%. The optimal time to reach the target (for constantly setting $V_{rot} = 0$) is 10.9 seconds. The time needed by our policy is 10.9 seconds as well. If we add a Gaussian noise on our chosen action ($V_{rot} = V_{rot} + \mathrm{N}(0, 60)$) the time need is 11.1 seconds only. Due to acceleration the robot never reaches a high rotational velocity.

All the above is based on the assumption that the state space is full observable. That means there are no further features outside of $\mathcal{S}$ which can influence the performance of π.

3.2 The Dribble Task

In human soccer players can quickly change their direction and, not seldom, try to deke opponent players by moving into one direction for a short time just to make the opponent move in that direction too. Exploiting the delay of the opponents perception as well as its physical inertia they quickly change their direction. We try to learn such a behavior for robot soccer using our method presented in section 2.

One can see, this task obviously is much more complex than the corridor-following task:

- A reasonable state space will have about *three times as many dimensions*.
- The value function is *not convex* any more.
- Any successor state depends not only on the behavior of the controlled robot but on the *behavior of the opponent* too.
- Parts of the state space may not be observable (*hidden state*).

Let us first introduce the scenario of the dribble task: There is one robot (the *forward*) that has a ball in its guiding device. The second robot (called *defender*) tries to get the ball from the forward using a fixed and deterministic policy. At the beginning the defender is placed at $(3m, 0m)$ and the forward (with ball) at $(-2m, ym)$ with $y \in [-2, +2]$ and $(0m, 0m)$ being the center of a robot soccer field. The forwards target space is defined by all points *behind* the defender in the sense of the x-coordinate.

The state space consists of the features d (the distance between forward and defender), α (the angle between forward and defender), φ (the forwards orientation relative to α), V_{tr} and V_{rot} (the forwards translational and rotational velocities), d_{nob} (the distance between forward and defender observed by the defender), α_{nob} (α observed by the defender), φ_{nob} (φ observed by the defender), V_{nob} (V_{tr} observed by the defender), and $V_{def,nob}$ (the maximal velocity of the defender). Figure 7 depicts the task and some components of the state space.

To deal with the hidden state problem we split the state space $\mathcal{S}$ into an observable part $\mathcal{S}_{obs}$ and a not observable part $\mathcal{S}_{nob}$:

$$\mathcal{S} = \mathcal{S}_{obs} \times \mathcal{S}_{nob} \quad (10)$$

The observable part includes

$$\mathcal{S}_{obs} = \langle d, \alpha, \varphi, V_{tr}, V_{rot} \rangle \quad (11)$$

while the not observable part consists of

$$\mathcal{S}_{nob} = \langle d_{nob}, \alpha_{nob}, \varphi_{nob}, V_{nob}, V_{def,nob} \rangle \quad (12)$$

Fortunately $\mathcal{S}_{nob}$ is not really unobservable: As written above deking is based on delays in perception and motion. Further the perception of the defender definitely depends on some former position of the forward. Since we know the defender will compute a point where it can intercept the ball and will simply go there we need to construct a model of its perception and motion abilities. This can be done by online estimating delay parameters and computing the defenders maximal velocity. Since this would be another chapter (obtaining a model for hidden state is not the matter of this paper) we so far give reasonable values of $\mathcal{S}_{nob}$ to the forward robot in the following experiments.

Analogously to the split of $\mathcal{S}$ we split our projection $\mathcal{P}$ as well:

$$\mathcal{P} = \mathcal{M} \circ (\mathcal{P}_{obs} \times \mathcal{P}_{nob}) \quad (13)$$

where $\mathcal{P}_{obs}$ is a projection for the change in state of the observable part of $\mathcal{S}$ (only depending on the robots action because the defenders behavior is deterministic), $\mathcal{P}_{nob}$ a projection for the change in state of the unobservable part, and $\mathcal{M}$ a merging function:

$$\mathcal{P}_{obs} : \begin{cases} \mathcal{S}_{obs} \times \mathcal{A} \to \mathcal{S}_{obs} \\ \langle \zeta_{obs}(t), a(t) \rangle \mapsto \zeta_{obs}(t+1)_{succ} \end{cases} \quad (14)$$

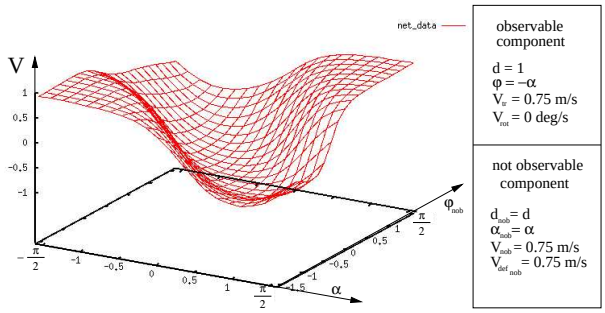

Figure 8: *Generalization of the trained neural network. The graphic shows the output of the network computing the value function with fixed input in all but two dimensions.*

$$\mathcal{P}_{nob} : \begin{cases} \mathcal{S}_{nob} \rightarrow I\!R^2 \\ \zeta_{nob}(t) \mapsto \langle \Delta x, \Delta y \rangle \end{cases} \tag{15}$$

$$\mathcal{M} : \begin{cases} \mathcal{S}_{obs} \times I\!R^2 \rightarrow \mathcal{S}_{obs} \\ \langle \zeta_{obs}(t)_{succ}, \langle \Delta x, \Delta y \rangle \rangle \mapsto \zeta_{obs}(t+1) \end{cases} \tag{16}$$

$\mathcal{P}_{obs}$ computes the movement of the forward in the observable part of the state space ($\mathcal{S}_{obs}$) dependent on the chosen action $a(t)$. $\mathcal{P}_{nob}$ maps to $I\!R^2$ because it computes the movement in position of the defender depending on the unobservable features of the state space $\mathcal{S}_{nob}$. The merging function $\mathcal{M}$ maps the result of $\mathcal{P}_{obs}$ and the change in position of the defender to a successor state in $\mathcal{S}_{obs}$.

The action space $\mathcal{A}$ is $[-180, +180]$ once again where $\mathcal{A} = V_{rot}$ (an action is a rotational velocity). $\mathcal{V}_{ass}(\zeta(i))$ is set to -1 if the defender touches the forward or the forward exceeds a certain time limit, to $+1$ if the forward reaches the state space, and to $\gamma * \mathcal{V}_{ass}(\zeta(i+1))$ else.

During exploration the robot performs random actions each for a random time. Some initial hand-coded policy is used also. In this policy at some point the robot suddenly changes its direction. We perform a set of 500 explorations with an average duration of about 6 seconds. The states are measured at a frequency of 10Hz. Altogether that leads to 30000 patterns obtained in less than 2 hours of simulation. The success rate in the exploration phase is less than 10%.

In application the action V_{rot} is chosen from

$$V_{rot} \in \{-120, 0, 120\} \tag{17}$$

which is a real coarse selection. Nevertheless the success rate in the application phase was 100% based on 250 runs and the experimental setup described above. An animation (gif,870KB) of a typical dribble scene can be viewed at *http://www9.in.tum.de/people/buck/dribble_task1.gif*. Figure 8 shows two dimensions of our trained value function with the other dimensions set by fixed rules. One can see the smooth surface as an indication for generalization.

The experiments described above show that our approach is able to deal with non-trivial robot control problems.

4 Related Work

There is a number of applications for reinforcement learning in robot control. However, most of them use variations of Q-learning [19] being one of the most popular algorithms in reinforcement learning [16]. Q-learning incrementally learns a state-action value function Q from experience. $Q(\zeta, a)$ computes how good it is to perform action a in state ζ. According to

$$Q(\zeta_t, a_t) \leftarrow Q(\zeta_t, a_t) + \alpha(r_{t+1} + \gamma \max_{a'} Q(\zeta_{t+1}, a') - Q(\zeta_t, a_t)) \tag{18}$$

Q is updated incrementally where $\gamma \in [0, 1)$ is a decrease factor and r_{t+1} is the reward received for performing action a_t in state ζ_t. All this takes place in a discrete state space. To apply Q-learning for tasks in continuous domains like robot control the state space can be discretized [8]. But in general operating in a discrete state space brings some problems: Using a coarse discretization the control output is not smooth and using a fine discretization the number of states becomes huge, especially in high dimensional spaces [6]. Furthermore state spaces in robot control usually include some non-linearities (resulting from angles, acceleration, etc.) which are hard to discretize reasonably. It is extremely difficult to determine the neighborhood of a state accessible by any action in a high dimensional state space including non-linearities.

Nevertheless some successful work has been done in robot reinforcement learning in discrete state spaces: Asada et al. [1] has transformed camera images into discrete states and applied Q-learning in that space. Stone and Sutton [15] have applied the SARSA(λ) algorithm to multiple agents (3 vs.2) in the RoboCup soccerserver simulation environment. Low-level skills and 2 vs.1-behavior have been learned in the same environment [12].

In order to both limit the number of states and smooth the control output Moore and Atkeson [10] propose to increase the resolution of state space in interesting regions while decreasing it in less interesting regions.

The standard approach for reinforcement learning in continuous state space is to use the gradient of the value function to choose an action [20]. But only replacing the discrete value function by a continuous approximation has been shown to fail [4]. Thrun and Schwartz [18] find the *overestimation phenomenon* to be the main reason for that. Overestimation results from noise which is likely in real world applications and since Q-learning is an incremental algorithm there is the danger of accumulation of such errors. The most recent works to overcome these

problems include the HEDGER algorithm [14], barycentric interpolators [11], and a type of instance based reinforcement learning [7].

Another problem reported in the literature is the discontinuity problem [17]: What happens if the optimal value function is non-continuous? Nearly all common function approximators will fail here. But in practice this case seldom appears.

Moreover traditional reinforcement learning assumes that all factors relevant for the computation of the value function are observable. But in practice this constraint does not hold. Due to the unobservable parts of the state space this problem is called *hidden state*. Especially in the dribble task in section 3.2 there is a lot of hidden state, namely the perception of the defending robot. Beside others McCallum [9] has developed solutions to overcome this problem.

5 Conclusions and Future Work

The work described in this paper is a non-incremental algorithm for reinforcement learning in multi-dimensional continuous state space. The main contributions are an appropriate method for the safe estimation of a value function, a neural one-step projection, and instructions how to overcome problems that may occur in practice.

In contrast to previous work our method does not suffer from possible overestimation resulting from incremental Q-learning. Our value function is a lower bound of the optimal value function. Experiments have shown that our method is a suitable means for robot control. However several improvements remain for the future: One can imagine to perform a second or even a third exploration phase based on some knowledge of initial policies. Further those additional explorations could concentrate on interesting regions of the state space since the theoretical amount for exploration increases exponential with the number of dimensions. But this remains an inherent problem of multi-dimensional state spaces.

References

[1] M. Asada, S. Noda, S. Tawaratsumida, and K. Hosoda: *Purposive Behavior Acquisition for a Real Robot by Vision-Based Reinforcement Learning*. Machine Learning Journal, 23:279-303, 1996.

[2] 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.

[3] D. Bertsekas and J. Tsitsiklis: *Neuro-Dynamic Programming*. Athena Scientific, Belmont, MA, 1996.

[4] J. Boyan and A. Moore: *Generalization in Reinforcement Learning: Safely approximating the value function*. In Tesauro, G., D. S. Touretzky, and T. K. Leen (eds.), Advances in Neural Information Processing Systems 7 (NIPS). MIT Press, 1995.

[5] S. Buck, M. Beetz, and T. Schmitt: *M-ROSE: A Multi Robot Simulation Environment for Learning Cooperative Behavior*. In H. Asama, T. Arai, T. Fukuda, and T. Hasegawa (eds.): Distributed Autonomous Robotic Systems 5, Springer Verlag, 2002.

[6] K. Doya: *Reinforcement Learning In Continuous Time and Space*. Neural Computation, 12, 219-245, 2000.

[7] J. Forbes and D. Andre: *Real-time reinforcement learning in continuous domains*. AAAI Spring Symposium on Real-Time Autonomous Systems. 2000.

[8] Z. Kalmar, C. Szepesvari, and A. Lorincz: *Module Based Reinforcement Learning for a Real Robot*. Proceedings of the 6th European Workshop on Learning Robots, Lecture Notes in AI. 1998.

[9] A. McCallum: *Reinforcement Learning with Selective Perception and Hidden State*. PhD thesis, 1995.

[10] A. Moore and C. Atkeson: *The Parti-game Algorithm for Variable Resolution Reinforcement Learning in Multidimensional State-spaces*. Machine Learning Journal, 21(3):199-233, 1995.

[11] R. Munos and A. Moore: *Barycentric Interpolators for Continuous Space & Time Reinforcement Learning*. Advances in Neural Information Processing Systems 11, May 1999.

[12] M. Riedmiller and A. Merke: *Karlsruhe Brainstormers - a reinforcement learning approach to robotic soccer II*. In 5th International Workshop on RoboCup, Lecture Notes in Artificial Intelligence, 2001, Springer Verlag.

[13] M. Riedmiller and H. Braun: *A direct adaptive method for faster backpropagation learning: the Rprop algorithm*, Proceedings of the ICNN, San Francisco, 1993.

[14] W.D. Smart and L.P. Kaelbling: *Practical Reinforcement Learning in Continuous Spaces*. In Proceedings of the Seventeenth International Conference on Machine Learning, pp. 903-910, 2000.

[15] P. Stone and R. Sutton: *Scaling Reinforcement Learning toward RoboCup Soccer*. Eighteenth International Conference on Machine Learning, 2001.

[16] R.S. Sutton and A.G. Barto: *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

[17] M. Takeda, T. Nakamura, and T. Ogasawara: *Continuous Valued Q-learning Method Able to Incrementally Refine State Space*. Proceedings of the IEEE International Conference on Intelligent Robots and Systems, 2001.

[18] S. Thrun and A. Schwartz: *Issues in Using Function Approximation for Reinforcement Learning*. In M. Mozer, P. Smolensky, D. Touretzky, J. Elman, and A. Weigend, editors, Proceedings of the Connectionist Models Summer School, pp. 255-263, Hillsdale, NJ, 1993.

[19] C.J. Watkins and P. Dayan: *Q-learning*. Machine Learning Journal, 8:279-292, 1992.

[20] P. Werbos: *A menu of designs for reinforcement learning over time*. In Neural Networks for Control, W.T. Miller, R.S. Sutton, and P.J. Werbos, editors, pp 67-95, MIT Press, MA, USA, 1990.