ELSEVIER

Robotics and Autonomous Systems 22 (1997) 87–101

Robotics and Autonomous Systems

An integrated architecture for learning of reactive behaviors based on dynamic cell structures *

Jörg Bruske *, Ingo Ahrns, Gerald Sommer

Institut für Informatik und Praktische Mathematik, Christian-Albrechts-Universität Kiel, Preußerstraße 1-9, D-24105 Kiel, Germany

Received 1 May 1996; accepted 7 April 1997

Abstract

In this contribution we want to draw the readers attention to the advantages of *dynamic cell structures* (DCSs) (Bruske and Sommer, 1995) for learning reactive behaviors of autonomous robots. These include *incremental on-like learning, fast output calculation,* a flexible *integration of different learning rules* and a close connection to *fuzzy logic*. The latter allows for incorporation of prior knowledge and to interpret learning with DCSs as fuzzy *rule generation and adaptation.*

After successful applications of DCSs to tasks involving supervised learning, feedback error learning and incremental category learning, in this article we take *reinforcement learning* of reactive *collision avoidance* for an autonomous mobile robot as a further example to demonstrate the validity of our approach. More specifically, we employ a *REINFORCE* (Williams, 1992) algorithm in combination with an *adaptive heuristic critique (AHC)* (Sutton, 1988) to learn a *continuous valued sensory motor mapping* for obstacle avoidance with a TRC Labmate from delayed reinforcement. The sensory input consists of eight unprocessed sonar readings, the controller output is the continuous angular and forward velocity of the Labmate. The controller and the AHC are integrated within a single DCS network, and the resulting avoidance behavior of the robot can be analyzed as a set of fuzzy rules, each rule having an additional *certainty value.*

Keywords: Dynamic cell structures; RBF networks; Sugeno fuzzy control; Reactive control; Integrated architecture; Reinforcement learning; Mobile robot; Obstacle avoidance

1. Introduction

Reactive control rests on the hypothesis that the desired system behavior can be realized by a simple mapping between sensory data and motor actions, i.e. without involving a controller state. Since such a simple mapping from (sensory) input to (motor) output is the essence of most conventional feed forward artificial neural networks (ANNs) and fuzzy logic controllers, their application to learning reactive control is self-evident and has been tackled by numerous researchers in the fields of robotics, ANNs and fuzzy logic. Taking reactive collision avoidance as an example, this problem has been tackled using either supervised learning or reinforcement learning to train ANNs, e.g. [13,17], or to refine (neuro-) fuzzy controllers, e.g. [3]. Various fuzzy logic controllers without adaptability

* A preprint of this article is available as: Technical Report No. 9604, Institut für Informatik und Praktische Mathematik, Christian-Albrechts-Universität zu Kiel, 1996.

* Corresponding author. Tel.: +49 431 560480; fax: +49 431 560481; e-mail: jbr@informatik.uni-kiel.de.

PII S0921-8890(97)00032-8

have been suggested as well, e.g. [18]. As is evident from the continuing research effort in this field, reactive control and collision avoidance in particular are far from being closed chapters in robotics.

On behalf of learning reactive control, the system should possess the following properties:

- Fast on-line learning to adapt in real time.
- Incremental learning, i.e. the learner should not forget what he has learned so far but incrementally adapt the model complexity.
- One shot learning, i.e. just storing a new situation–action pair on demand.
- Integration of domain knowledge to avoid learning from scratch.
- Support of different and potentially complementary learning rules (unsupervised, reinforcement, feedback error and supervised learning) to exploit all available information and to escape from local minima (w.r.t. some performance measure).

All these topics are hotly debated as reflected in a variety of recent publications, e.g. [11,15,19]. In Section 2 we argue that dynamic cell structures (DCSs) [8] with additional provisions for on-line learning [5] meet these requirements and can thus be beneficially utilized for learning reactive control. Recent applications of DCSs involving supervised learning [5], feedback error learning [7], incremental category learning [20] and reinforcement learning (this article) underline our claim.

In Section 3 we introduce the integrated learning architecture which we employ to learn reactive collision avoidance from delayed reinforcement in Section 4. Different to e.g. [13] we associate a continuous sensory input with continuous motor actions, i.e. we learn an appropriate forward velocity and angular velocity, not only to turn either left or right with a fixed velocity. As input we use the unprocessed readings of eight noisy sonar sensors, which yield learning more difficult than e.g. the three laser readings used in [17]. We avoid over-simplified simulations by simulating the geometric, dynamic and sensory characteristics of our real TRC Labmate. Employing a REINFORCE [23] algorithm this application is interesting on its own since it avoids rather questionable heuristics and provides an additional certainty value for each control rule. We relate our architecture and experiments to previous work in Section 5 and close with a brief summary in Section 6.

2. Why choosing a DCS

In this section we first provide the reader with the foundations of DCS network. We then discuss some straightforward modifications for on-line learning and show that DCSs can be regarded as restricted Sugeno-type fuzzy controllers with additional learning and exploitation of the topology of the input space. Finally we show how different learning rules can be used to train DCSs. From this section it should become clear that DCSs meet the requirements for learning of reactive control as put forward in Section 1.

2.1. Dynamic cell structures (DCSs)

DCSs as introduced in [8] denote a class of RBF-based approximation schemes attempting to concurrently learn and utilize perfectly topology preserving feature maps (PTPMs.) DCSs are a subclass of Martinetz's topology representing networks (TRN) [14] defined to contain any network using competitive Hebbian learning for building PTPMs.

The architectural characteristics of a DCS network (see Fig. 1) are:

- one hidden layer of radial basis function units (possibly growing/shrinking);
- a dynamic lateral connection structure between these units; and
- a layer of (usually linear) output units.

Training algorithms for DCSs adapt the lateral connection structure towards a PTPM by employing a competitive Hebbian learning rule and activate and adapt RBF units only in the neighborhood of the current stimulus. Here, "neighborhood" relates to the simultaneously learned topology.

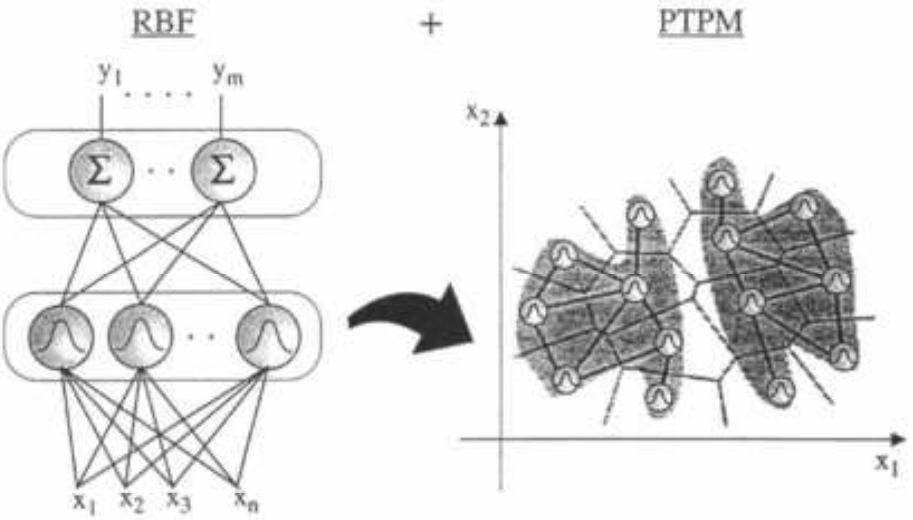

Fig. 1. DCSs are RBF networks (left) plus a lateral connection structure between the RBF units attempting to build a PTPM by competitive Hebbian learning (right, closely modified Fig. 5(d) from [14]).

We use a normalized RBF approximation scheme and hence the output of the DCSs calculates as [1]

$$y(\boldsymbol{x}) = \frac{\sum_{i \in \mathrm{Nh}(\mathrm{bmu}_x)} o^i h_i(\|\boldsymbol{x} - \boldsymbol{c}^i\|)}{\sum_{i \in \mathrm{Nh}(\mathrm{bmu}_x)} h_i(\|\boldsymbol{x} - \boldsymbol{c}^i\|)}, \tag{1}$$

where $h_i(\|\boldsymbol{x} - \boldsymbol{c}^i\|)$ denotes a radial basis function with center $\boldsymbol{c}^i$. The vectors o^i can be thought of as output weight vectors attached to each RBF unit. The functions $h_i : \mathbb{R}^+ \to \mathbb{R}^+$ are strictly monotonically decreasing with $h_i(0) = 1.0$ and $h_i(\infty) = 0$, which we have realized by Gaussians.

The neighborhood $\mathrm{Nh}(j)$ of neural unit j is defined as the unit itself together with its direct topological neighbors

$$\mathrm{Nh}(j) = \{i | C_{ji} \neq 0,\ 1 \leq i \leq N\} \cup \{j\}, \tag{2}$$

and the best matching unit, bmu_x is given by

$$\|\boldsymbol{c}^{\mathrm{bmu}} - \boldsymbol{x}\| \leq \|\boldsymbol{c}^i - \boldsymbol{x}\|, \quad (1 \leq i \leq N). \tag{3}$$

The adjacency matrix $\boldsymbol{C}$ represents the lateral connection structure between the RBF units and is adapted by a competitive Hebbian learning rule like

$$\Delta C_{ij} = \begin{cases} 1, & h_i \cdot h_j \geq h_k \cdot h_l \ \ \forall (1 \leq k, l \leq N), \\ 0, & \text{otherwise}, \end{cases} \tag{4}$$

giving rise to a PTPM (see [14]).

Compared to RBF networks, learning and *exploiting* the topology of the input space as in DCSs not only significantly speeds up computation of the output value (because only the bmu and its topological neighbors need to be

[1] In the following we denote vectors by $\boldsymbol{x}$, components of a vector by $\boldsymbol{x}_i$ and enumerations of vectors by $\boldsymbol{x}^i$.

evaluated and adapted), but is also like to improve approximation quality. This is because in the usual RBF scheme units whose centers have small Euclidean distance to the stimulus but large intra-manifold distance to the point of projection of the stimulus onto the manifold can spoil the approximation.

In order to adapt the output vector $\boldsymbol{o}^i$ we usually employ gradient descent on an error function $E(\boldsymbol{y})$. As in RBF networks, the centers $\boldsymbol{c}^i$ can be adapted by gradient descent as well but may alternatively be trained by a Kohonen-type learning rule (utilizing the lateral connection structure) for adaptive vector quantization, leading to a hybrid training scheme.

Incremental growth of the DCS network is accomplished by inserting prototypes $(\boldsymbol{x}, \boldsymbol{y})$ with $\boldsymbol{c}^i = \boldsymbol{x}$ and $\boldsymbol{o}^i = \boldsymbol{y}$ (one shot learning) [2] or inserting neutral units in regions of the input space where the approximation performance of the DCSs is unsatisfactory (hence increasing the resolution in this region). The latter can be realized by attaching an additional error variable (resource value [9]) to each neural unit which monitors the performance of the network when this unit is involved in output calculation. Such an error driven insertion of new units has the advantage that the distribution of neural units attempts to minimize the error (and not merely reflects the input probability density, as it would be the case with a Kohonen-type rule). Moreover, error driven insertion is known to cope with problems of local minima since it takes into account the global error distribution. See also [9] for more on incremental growth of DCSs like networks utilizing resource values.

2.2. On-line learning with dynamic cell structures

DCSs lend themselves for on-line learning but some modifications are necessary to take into account specific problems, e.g. the non-stationary input probability density in on-line learning. The first provision is a mixed insertion strategy which inserts new neural units not only between units with high error variables (*error driven insertion*) but also in regions not sufficiently covered by the network, i.e. when the activation of the bmu is too small. This additional *distance driven insertion* strategy helps the network with unfolding problems and can be regarded as a coarse scale approximation to be refined by the error driven insertion strategy.

Second, the Hebbian learning rule (4) must be modified to allow "forgetting" of no longer existing neighborhood relations (lateral connections). This can easily be achieved by extending (4) with a decay term. However, care has to be taken to avoid "dead neurons", i.e. units which are disconnected from the remaining and are not further utilized. Our solution is a non-symmetric Hebbian learning rule as proposed in [5], where only lateral connections emerging from the bmu can be discarded.

Finally, if using a Kohonen-type learning rule for center adaptation, one has to take care that the centers do not eventually collapse in a region of high data density. This problem can be circumvented by either freezing of the learning constant (as is usually done) or error modulation of the learning constant utilizing the error variables of the neural units. An example for such a modulation can be found in [5] as well, and this was the method of choice for our experiments. Here, the learning parameter for center adaptation becomes the smaller the more equal the local error variables.

2.3. Fuzzy logic for pre-structuring and interpreting DCSs

In this section we will show that apart from representing an easy-to-implement, fast incremental on-line learning scheme DCSs are closely related to Sugeno-type fuzzy control. More particular, we will show that viewed from the perspective of fuzzy control, normalized DCSs represent restricted Sugeno-type fuzzy controllers with the additional feature that only the best matching rules are evaluated and adapted.

[2] Strictly speaking, setting $\boldsymbol{o}^i = \boldsymbol{y}$ is only valid for negligible overlap of the basis functions. It is, however, straightforwarding to solve (1) for $\boldsymbol{o}^i$ in case of non-negligible overlap as well.

evaluated and adapted), but is also like to improve approximation quality. This is because in the usual RBF scheme units whose centers have small Euclidean distance to the stimulus but large intra-manifold distance to the point of projection of the stimulus onto the manifold can spoil the approximation.

In order to adapt the output vector $\boldsymbol{o}^i$ we usually employ gradient descent on an error function $E(\boldsymbol{y})$. As in RBF networks, the centers $\boldsymbol{c}^i$ can be adapted by gradient descent as well but may alternatively be trained by a Kohonen-type learning rule (utilizing the lateral connection structure) for adaptive vector quantization, leading to a hybrid training scheme.

Incremental growth of the DCS network is accomplished by inserting prototypes $(\boldsymbol{x}, \boldsymbol{y})$ with $\boldsymbol{c}^i = \boldsymbol{x}$ and $\boldsymbol{o}^i = \boldsymbol{y}$ (one shot learning) [2] or inserting neutral units in regions of the input space where the approximation performance of the DCSs is unsatisfactory (hence increasing the resolution in this region). The latter can be realized by attaching an additional error variable (resource value [9]) to each neural unit which monitors the performance of the network when this unit is involved in output calculation. Such an error driven insertion of new units has the advantage that the distribution of neural units attempts to minimize the error (and not merely reflects the input probability density, as it would be the case with a Kohonen-type rule). Moreover, error driven insertion is known to cope with problems of local minima since it takes into account the global error distribution. See also [9] for more on incremental growth of DCSs like networks utilizing resource values.

2.2. On-line learning with dynamic cell structures

DCSs lend themselves for on-line learning but some modifications are necessary to take into account specific problems, e.g. the non-stationary input probability density in on-line learning. The first provision is a mixed insertion strategy which inserts new neural units not only between units with high error variables (*error driven insertion*) but also in regions not sufficiently covered by the network, i.e. when the activation of the bmu is too small. This additional *distance driven insertion* strategy helps the network with unfolding problems and can be regarded as a coarse scale approximation to be refined by the error driven insertion strategy.

Second, the Hebbian learning rule (4) must be modified to allow "forgetting" of no longer existing neighborhood relations (lateral connections). This can easily be achieved by extending (4) with a decay term. However, care has to be taken to avoid "dead neurons", i.e. units which are disconnected from the remaining and are not further utilized. Our solution is a non-symmetric Hebbian learning rule as proposed in [5], where only lateral connections emerging from the bmu can be discarded.

Finally, if using a Kohonen-type learning rule for center adaptation, one has to take care that the centers do not eventually collapse in a region of high data density. This problem can be circumvented by either freezing of the learning constant (as is usually done) or error modulation of the learning constant utilizing the error variables of the neural units. An example for such a modulation can be found in [5] as well, and this was the method of choice for our experiments. Here, the learning parameter for center adaptation becomes the smaller the more equal the local error variables.

2.3. Fuzzy logic for pre-structuring and interpreting DCSs

In this section we will show that apart from representing an easy-to-implement, fast incremental on-line learning scheme DCSs are closely related to Sugeno-type fuzzy control. More particular, we will show that viewed from the perspective of fuzzy control, normalized DCSs represent restricted Sugeno-type fuzzy controllers with the additional feature that only the best matching rules are evaluated and adapted.

[2] Strictly speaking, setting $\boldsymbol{o}^i = \boldsymbol{y}$ is only valid for negligible overlap of the basis functions. It is, however, straightforwarding to solve (1) for $\boldsymbol{o}^i$ in case of non-negligible overlap as well.

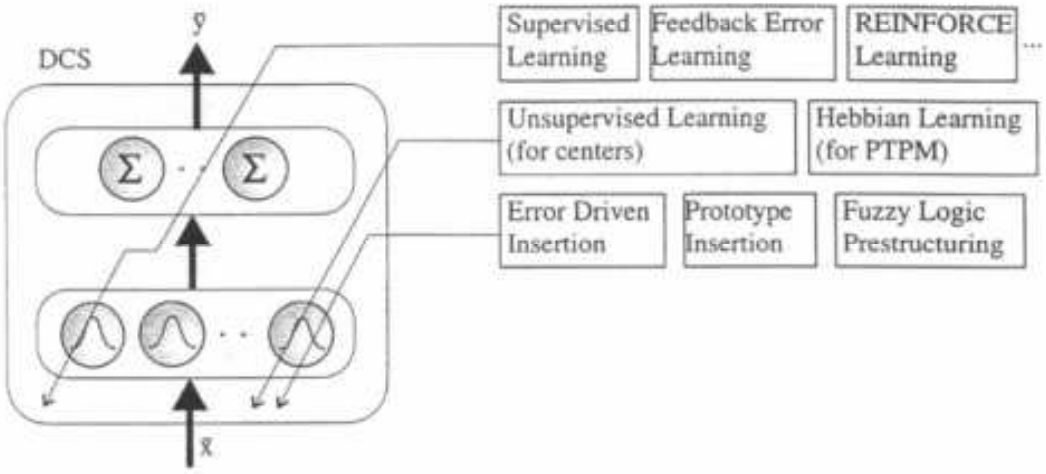

Fig. 2. Combining different learning paradigms and error sources for training DCSs.

by competitive Hebbian learning. The resource values (error variables) attached to the units can be utilized for incremental growing of the network (error driven insertion) and error modulation of unsupervised learning laws. Note that the error function contributing to the resource values does not need to be differentiable. Finally, the already discussed insertion of prototypes and prestructuring with fuzzy logic can help in learning reactive control.

Examples of DCSs utilizing different learning rules and error sources include Ref. [7], employing feedback error learning and the fixation error as a resource for learning accurate and fast saccade control of a binocular head, Ref. [5] employing supervised learning and the approximation error as a resource for learning pole balancing, Ref. [6] employing Q-learning and the TD error as the resource for learning discrete control policies and Ref. [20] using unsupervised learning and the quantization error as a resource for incremental category learning of KHEPERA robots.

Following the recent discussion in learning reactive control, combining and utilizing different learning paradigms seems to be of utmost importance and the key to practical success. Each learning paradigm has its weakness (e.g. pure reinforcement learning is usually too slow) and they all suffer from local minima. In combination, however, difficult problems can be solved in reasonable time (e.g. [15], combining reinforcement with self-supervised learning for goal driven obstacle avoidance).

3. REINFORCEment learning with DCS

We will now introduce a learning controller based on DCSs for learning continuous valued sensory–motor mappings from delayed reinforcement. DCSs integrate an adaptive heuristic critique (AHC) as well as the actual controller within a single network. Referring to Section 2.4, the controller learns from a gradient based REINFORCE algorithm as well as an error modulated Kohonen rule. It utilizes error driven as well as distance driven insertion for incremental growing and can be prestructured with fuzzy logic rules.

Fig. 3 depicts the controller architecture. Besides DCS we utilize an additional *associative search element* (ASE).

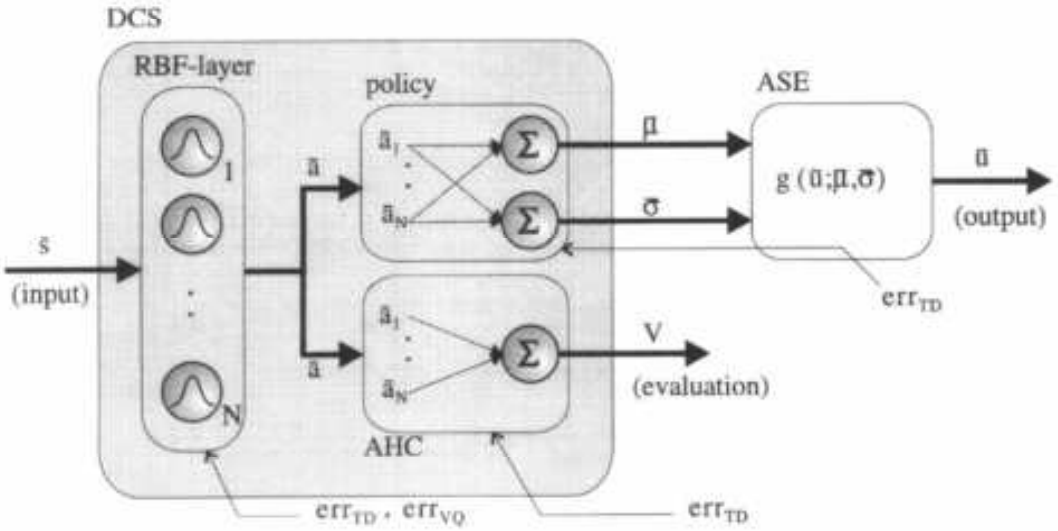

Fig. 3. DCS-based controller architecture for real valued reinforcement learning.

3.1. Controller output calculation

The calculation of the control vector $\boldsymbol{u}$ given the sensory input $\boldsymbol{s}$ proceeds as follows:

First, the input vector $\boldsymbol{s}$ is transformed to an activity vector $\boldsymbol{a}$ representing the normalized activations of the RBF units with centers $\boldsymbol{c}_i$ and uniform width d:

$$a_i = \frac{\mathrm{rbf}_i(s)}{\sum_{j\in \mathrm{Nh}(\mathrm{bmu}_s)} \mathrm{rbf}_j(s)} \quad \text{with } \mathrm{rbf}_i(s) = \exp\left(\frac{-(s-c^i)^2}{d^2}\right) \quad \forall i \in \mathrm{Nh}(\mathrm{bmu}_s). \tag{8}$$

Second, a prototypical action vector $\boldsymbol{\mu}$, the certainty vector $\boldsymbol{\sigma}$ and the predicted cumulative reinforcement V are calculated by a weighted sum of contributing output vectors attached to the RBF units:

$$\mu = \sum_{j\in \mathrm{Nh}(\mathrm{bmu}_s)} a_j \mu^j, \qquad \sigma = \sum_{j\in \mathrm{Nh}(\mathrm{bmu}_s)} a_j \sigma^j, \qquad V = \sum_{j\in \mathrm{Nh}(\mathrm{bmu}_s)} a_j V_j. \tag{9}$$

Finally the ASE draws the actual action vector $\boldsymbol{u}$ according to the probability density function $g(\boldsymbol{u};\boldsymbol{\mu},\boldsymbol{\sigma})$, the components of g being Gaussians:

$$\mathrm{Prob}(u_i) = g_i(u;\mu,\sigma) = \frac{1}{\sqrt{2\pi\sigma_i^2}} \exp\left(-\frac{(u_i-\mu_i)^2}{2\sigma_i^2}\right). \tag{10}$$

3.2. Controller adaptation

On-line adaptation is performed w.r.t. the contribution prototypical vectors $\boldsymbol{\mu}^j$, certainty vectors $\boldsymbol{\sigma}^j$ and evaluation values V_j attached to the RBF units, as well as to the centers $\boldsymbol{c}^j$ of the RBF units.

The evaluation values V_j are updated using a TD(1) rule [21]:

$$\Delta V_j = \beta \, \mathrm{err}_{\mathrm{TD}}(t) \sum_{\tau=0}^{t} \gamma^{t-\tau} \frac{\partial}{\partial V_j} V(s_\tau) \quad \text{with} \quad \frac{\partial}{\partial V_j} V(s_t) = \boldsymbol{a}_j(t), \tag{11}$$

where $\mathrm{err}_{\mathrm{TD}}(t) = r(t) - b(t)$ denotes the current temporal difference error with $r(t)$ the reinforcement signal, and $b(t) = V(s_{t-1}) - \gamma V(s_t)$ is the adaptive baseline. Of course, for computing the sum in (11) it is not necessary to store all previous activations. Instead, we only need to store the last value of this sum and recursively updated it.

Prototypical action vectors $\boldsymbol{\mu}^j$ and certainty vectors $\boldsymbol{\sigma}^j$ are adapted using a REINFORCE gradient descent:

$$\Delta \boldsymbol{\mu}_i^j = \beta_\mu (r - b) \frac{\partial}{\partial \boldsymbol{\mu}_i^j} \ln(g_i(\boldsymbol{u}; \boldsymbol{\mu}, \boldsymbol{\sigma})) \quad \text{and} \quad \Delta \boldsymbol{\sigma}_i^j = \beta_\sigma (r - b) \frac{\partial}{\partial \boldsymbol{\sigma}_i^j} \ln(g_i(\boldsymbol{u}; \boldsymbol{\mu}, \boldsymbol{\sigma})). \tag{12}$$

With $g_i(\boldsymbol{u}; \boldsymbol{\mu}, \boldsymbol{\sigma})$ being normal distributions we obtain the characteristic eligibilities

$$\frac{\partial}{\partial \boldsymbol{\mu}_i^j} \ln(g_i(\boldsymbol{u}_i; \boldsymbol{\mu}, \boldsymbol{\sigma})) = \frac{(\boldsymbol{u}_i - \boldsymbol{\mu}_i)}{(\boldsymbol{\sigma}_i)^2} \boldsymbol{a}_j \quad \text{and} \quad \frac{\partial}{\partial \boldsymbol{\sigma}_i^j} \ln(g_i(\boldsymbol{u}; \boldsymbol{\mu}, \boldsymbol{\sigma})) = \frac{(\boldsymbol{u}_i - \boldsymbol{\mu}_i)^2 - (\boldsymbol{\sigma}_i)^2}{(\boldsymbol{\sigma}_i)^3} \boldsymbol{a}_j \tag{13}$$

and hence

$$\Delta \boldsymbol{\mu}_i^j = \beta_\mu (r - b) \frac{(\boldsymbol{u}_i - \boldsymbol{\mu}_i)}{(\boldsymbol{\sigma}_i)^2} \boldsymbol{a}_j \quad \text{and} \quad \Delta \boldsymbol{\sigma}_i^j = \beta_\sigma (r - b) \frac{(\boldsymbol{u}_i - \boldsymbol{\mu}_i)^2 - (\boldsymbol{\sigma}_i)^2}{(\boldsymbol{\sigma}_i)^3} \boldsymbol{a}_j. \tag{14}$$

For a detailed discussion of REINFORCE algorithms we refer the reader to [23] by Williams. Utilizing a REINFORCE algorithm for reinforcement learning offers a sound theoretical basis, because in case of immediate reinforcement and constant baseline the update rules (12) are guaranteed to perform gradient descent on the expected reinforcement, see [23]. Unfortunately, no guarantee exists for our case of delayed reinforcement and adaptive baseline. Yet to us it seems reasonable to prefer algorithms, whose benevolent behavior has been proved at least in simple cases, to the ad hoc designed learning rules frequently encountered in this domain. Interestingly, Eq. (14) have a very intuitive interpretation. The prototypical actions $\boldsymbol{\mu}_i^j$ are moved in the direction of the current actions $\boldsymbol{u}_i$ if the temporal difference error $\mathrm{err}_{\mathrm{TD}}$ indicates an improvement (positive value) and in the opposite direction otherwise. Similarly, the $\boldsymbol{\sigma}_i^j$ change such that the current outputs $\boldsymbol{u}_i$ become more likely if $\mathrm{err}_{\mathrm{TD}}$ is positive (improvement) and more unlikely if it is negative. As stated by Williams and confirmed in our experiments the $\boldsymbol{\sigma}_i^j$ narrow down both in theory and practice if the reinforcement reaches its local maximum. They are, however, not guaranteed to converge to zero.

The $\boldsymbol{\sigma}_i^j$ can be used both in prestructuring and in the analysis of the DCS-based controller. If we are very certain about a prototypical action for a given situation we will set its $\boldsymbol{\sigma}_i^j$ value close to zero thus preventing the controller from exploring alternatives. Since non-decreasing $\boldsymbol{\sigma}_i^j$ indicate convergence to a local maximum, non-decreasing values at consecutive time steps indicate convergence to the corresponding prototypical action. This is our motivation for calling the $\boldsymbol{\sigma}_i^j$ certainty values.

Finally, the centers $\boldsymbol{c}^j$ of the bmu and its topological neighbors are updated according to an error modulated Kohonen rule as described in [5]:

$$\Delta \boldsymbol{c}^j = \varepsilon_{\text{modulated}} \, \mathrm{err}_{\mathrm{VQ}} \quad \text{with } \mathrm{err}_{\mathrm{VQ}} = \boldsymbol{s} - \boldsymbol{c}^j. \tag{15}$$

The error we use for modulation is the squared TD-error, $\mathrm{err}_{\mathrm{TD}}^2$, which serves as the local error variable (resource) for our DCS network. The lateral connection structure of the DCS is adapted towards a PTPM, see [5] for details.

A new neural unit (rule) is inserted whenever the distance to the current best matching unit is too large (distance driven insertion) or the resource value of a unit exceed a threshold (error driven insertion). At most $N_{\max}$ units

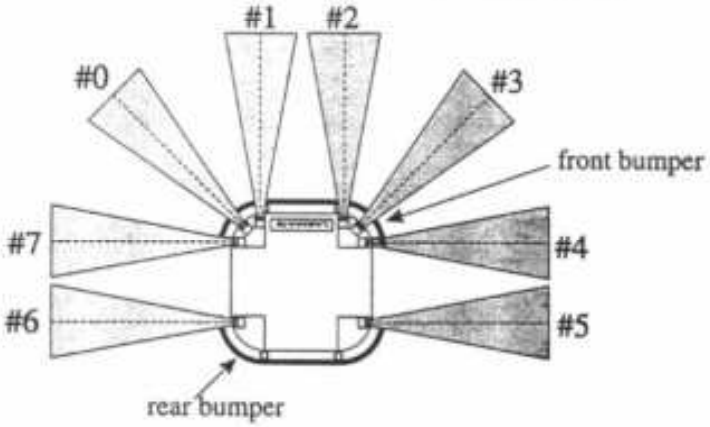

Fig. 4. Diagram of our TRC Labmate, sonar configuration and numbering.

are inserted. Using the squared TD-error as a resource value and thereby as a criterion for rule insertion serves to disambiguate regions of the input space where similar motor actions result in different rewards.

4. Experiments

As a testbed for our DCS-based REINFORCEment learning controller we have chosen reactive collision avoidance with our TRC Labmate. Our variant of this task involves learning a continuous valued sensory–motor mapping from unprocessed readings of eight sonar sensors to both forward and turn velocity. Since we want to test the performance of the controller independent of prior knowledge and other learning rules, we learn from scratch. This, however, would be highly impracticable for the real robot since pure real valued reinforcement learning from delayed rewards usually takes the robot too long time and too many collisions. We therefore performed our experiments in a simulator. For practical use, reinforcement learning must be augmented by incorporation of prior knowledge and/or additional self-supervised learning.

Simulations for testing the REINFORCEment learner are adequate if two requirements are met. First, the simulated learning problem must have similar characteristics and the same complexity as the real problem. We are confident to meet this requirement since we carefully modeled the dynamics and the sensory characteristics of the Labmate. Our simulator has been tested versus the real robot with a set of hand coded behaviors for indoor navigation [1], and they compare very well. The second requirement is that the reinforcement learning actually can be combined with prior knowledge and other learning rules as intended. As laid down in Section 2, our architecture does not pose a theoretical limitation, and the work of Millan [15] points a way of how it can be done in practice. [3]

From a reinforcement learning scheme intended to be used in combination with prior knowledge and other learning schemes we expect that:

(a) it actually improves performance and fine tunes the parameters and
(b) it does not show sudden breakdowns in performance, i.e. behaves stable.

[3] While Millan [15] has demonstrated the benefit of additional learning mechanisms, the claimed benefit of integrating (prior) domain knowledge for reinforcement learning still remains to be demonstrated. It is part of our ongoing work on using reinforcement learning for the real (physical) Labmate.

Learning stable control from scratch includes these issues but, moreover, demonstrates the learner to be able to generate new knowledge (rules) from the reinforcement signal.

4.1. The robot

We work with a TRC Labmate (see Fig. 4) equipped with eight sonar sensors and two tactile sensors (front and rear bumpers). The Labmate has a differential drive with two active wheels on a central axis allowing for turning on a point. We operate the Labmate in jog mode by controlling the turn rate and the forwarding velocity. The Labmate never leaves jog mode except in case of bumper contact or too small sensor readings, in which case it stops and triggers an emergency and orientation behavior. The sonars are polaroids with cones of approximately 20°.

Our simulator attempts to simulate the dynamics of the Labmate, has the same software interface as the motor control library and simulates the sonars as 20° cones with a variable amount of noise on the measurements. We also did simulations with idealized beams of 0° cone width.

4.2. Experimental setup

For training reactive collision avoidance the simulated Labmate is placed in a training environment as depicted in Fig. 5 (left). The Labmate is then allowed to drive around until either the distance to an obstacle drops below 20 cm or 200 time steps have elapsed, ending a trial. In the former case, the robot's orientation behavior is triggered that causes the Labmate to rotate until the front sensors indicate free space. In the latter case the Labmate is stopped and rotated for a random angle to prevent it from staying on the same collision free trajectory all the time and thus producing deceivingly good avoidance results. Alternatively, we could have randomly translated the robot or changed the environment from time to time, as others do. The advantage of our method is that it could be applied to the real Labmate as well. A disadvantage is that by randomly turning the Labmate we frequently force the Labmate into situations it would not have reached relying on the controller and can only escape by triggering the orientation behavior. This leads to a negative bias of avoidance learning performance and is an additional challenge to the learning algorithm.

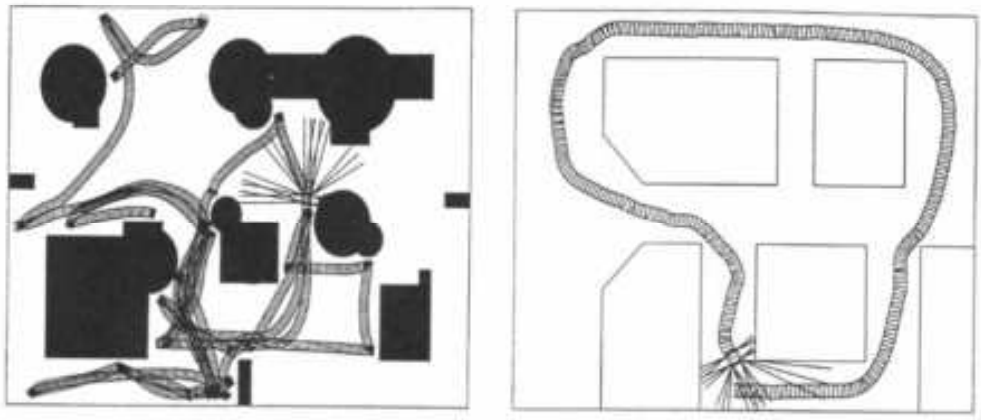

Fig. 5. Start of training, in training environment (left); end of training, in the test environment (right).

4.2.1. Initial knowledge

Since we want to test the performance of the controller independently of incorporated prior knowledge, the controller started with only one fuzzy rule:

$$\text{if } (x_1 \approx 5000) \wedge \cdots \wedge (x_8 \approx 5000) \text{ then } (\mu_v \approx v_0) \wedge (\mu_\omega \approx 0) \wedge (\sigma_v \approx \sigma_v^0) \wedge (\sigma_\omega \approx \sigma_\omega^0), \tag{16}$$

stating that if all sensor readings are about 5 m the Labmate should drive forward (zero angular velocity μ_ω) with velocity v_0. The certainty values for forward velocity and angular velocity (σ_v, σ_ω) were set to small initial values (σ_v^0, σ_ω^0).

4.2.2. Reinforcement function

As immediate reinforcement $r(t)$ we used the difference between evaluations of two succeeding situations, $r(t) = \Psi(s_t) - \Psi(s_{t-1})$, with $\Psi : \mathbb{R}^8 \to [0, 1]$ the evaluation function of a sensory situation. In addition, the Labmate was given a high negative reinforcement signal, $r_{\text{final}} = -100$, if it had approached an obstacle within less than 20 cm and thus triggered the orientation behavior.

More specifically, we use a convex combination with weighting factor c_i of non-linear transforms $\phi(s_i)$ of the individual components of s to construct Ψ:

$$\Psi(s) = \frac{\sum_{i=0}^{7} c_i \phi(s_i)}{\sum_{i=0}^{7} c_i}. \tag{17}$$

In our experiments we only used the four front sensors for the evaluation of a state and thus set $c_0 = \cdots = c_3 = 1$ and the remaining weighting factors to zero (compare Fig. 4).

For the non-linear transformation $\phi : \mathbb{R} \to [0, 1]$ we have

$$\phi(x) = \sqrt{\frac{x - x_{\min}}{x_{\max} - x_{\min}}}, \tag{18}$$

where $x_{\min} = 20$ cm is the minimal distance measured by our sonars and $x_{\max} = 2$ m is their (adjustable) time out distance.

We have chosen this kind of non-linear transformation because for $\sum_{i=0}^{3} s_i = \text{const}$ it yields the maximum value for $\Psi(s) = \sum_{i=0}^{3} \phi(s_i)$ when all components s_i are equal (as can be easily verified by a Lagrangian multiplier). This is desirable in collision avoidance tasks, where the vehicle should stay far from all obstacles and should not receive decreasing reinforcement for departing from one object on the cost of approaching another.

4.3. Experimental results

For a typical run Fig. 5 (left) shows the Labmate at the beginning of training in the training environment. Fig. 5 (right) shows collision free navigation of the Labmate in a test environment after the training phase. End of training is indicated by the averaged TD-error approaching a minimum, the averaged reinforcement approaching its maximum, and of course, avoidance of collision. Plots for both the TD-error and the reinforcement (averaged over 100 trials) are depicted in Figs. 6(a) and (b). In our experiments training took between 1000 and 10 000 trials, taking (on average) a longer time when "more realistic" sonar sensors (characteristic beam width of 20° and 5% noise) were simulated than idealized sensors (0° beam width, no noise). However, the difference between these two types of simulated sensors is rather small (see Fig. 6), indicating that the REINFORCE algorithm is robust w.r.t. small amounts of noise. This is not surprising, since it attempts to maximize the *expectation* of the reinforcement for each state. At most $N_{\max} = 100$ neural units (rules) have been utilized.

From our experiments we conclude that the controller is indeed able to learn collision avoidance from reinforcement and, in spite of remaining plasticity, behaves stable, i.e. does not show sudden breakdowns in avoidance performance. This stable behavior of real valued reinforcement learning is in accordance with experiments of

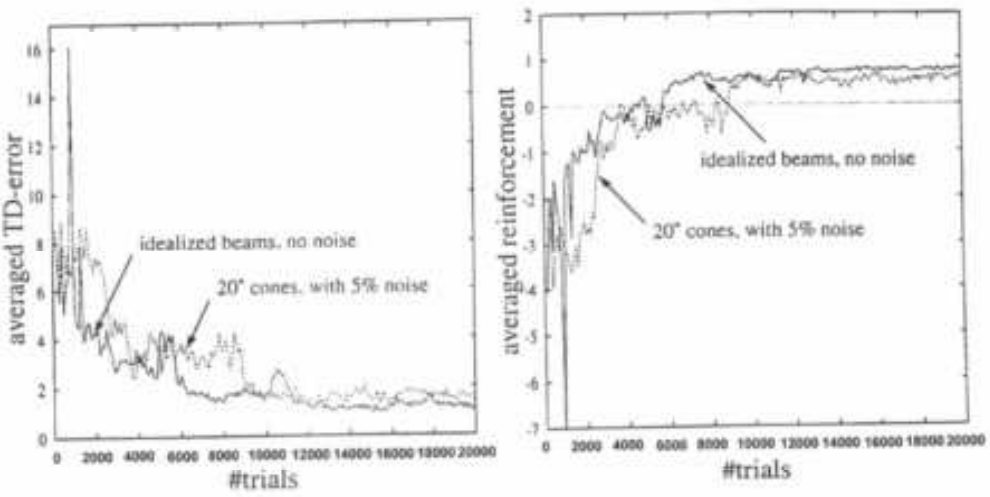

Fig. 6. Training: TD-error (averaged over 100 trials) versus no. of trials (left). Reinforcement (averaged over 100 trials) versus no. of trials (right).

Millan [15], using a similar reinforcement learning scheme. The experiments also underpin that learning from real reinforcement is a very slow process, although tuning of learning parameters may help to slightly improve performance.

We did not engage in a detailed investigation of the effects of different parameter values since very small changes in parameters can lead to very different exploration paths and very different experiences, yielding TD-error and reinforcement curves which can hardly be compared.[4] However, in most of the experiments we performed the simulated Labmate finally managed to learn a collision avoiding navigation policy, although the time it needed to succeed as well as the resulting policy largely differed.[5] Yet there were few cases where early in the training phase the robot got trapped in some region of the training environment and too frequent successive collisions (and the associated strong negative reinforcement) lead to divergence instead of convergence of both the policy (prototypical action and certainty values) and the AHC (prototypical evaluation values).

5. Related work

Due to the vast amount of related literature in the fields of artificial neural networks, fuzzy logic, reinforcement learning and obstacle avoidance, we will restrict ourselves to the most recent and most closely related publications.

Concerning the characteristics of our DCS networks, DCS are closely related to Fritzke's growing cell structures [9] and Martinetz topology representing networks (TRN) [14]. In fact, DCSs can be regarded as merging growing cell structures with TRN, see [8] for a detailed discussion.

The integrated architecture we suggest bears similarities to architectures proposed by Berns [11] and Millan [15]:

[4] Since single runs take very different courses even for the same parameters, the number of runs needed to calculate some kind of mean performance index for these parameters would be prohibitively large. Even if we attempted to do so, the variance would still be very high, thus making a comparison on the basis of mean performance indices (averaged over many runs for the same parameters) rather questionable.

[5] Parameter values and additional simulational details are documented in [1].

Berns employs an RBF-based architecture for adaptive control of a walking machine. He uses a similar algorithm for reinforcement learning of prototypical actions and shows how to combine RBF-based reinforcement learners within a hierarchical network. The latter idea may be applied to our architecture as well. Berns does not attempt to adapt centers of the RBF units, to exploit the TD-error for adaptive vector quantization or to build and utilize PTPMs. In addition, we emphasized the close connection to fuzzy logic (for incorporation of prior knowledge) and the flexibility of our architecture to benefit from different learning paradigms.

Millan's architecture for reinforcement learning of goal directed obstacle avoidance with a simulated Nomad 200 mobile robot is based on Alpaydin's GAL architecture [2]. GAL rests on locally tuned units as well but by introducing additional "category units" it differs from RBF networks and fuzzy logic controllers. Nor does GAL learn or utilize topology preserving feature maps. His reinforcement learning rule is based on TD(0) temporal differencing and a framework introduced by Gullapalli [10]. The latter differs from our REINFORCE algorithm in the calculation of the vector of standard deviations σ used for the generation of a stochastic output value by Eq. (10). Since in this framework σ does no longer depend on the contributing neural units they can no longer provide a certainty value. Millan's main innovation is his additional "planning component" for self-supervised learning which he demonstrates to speed up pure reinforcement learning. His work underlines the importance of combining different learning paradigms and hence the importance of an architecture capable of such integration.

Berenji's GARIC architecture [4] is perhaps the most well-known example of a hybrid neuro fuzzy controller (HNFC) trained by reinforcement learning. GARIC adapts the parameters of an action selection network (ASN), which encodes a fuzzy controller, by means of an action evaluation network (AEN) and a stochastic action modifier (SAM). While conceptually similar to GARIC, our control architecture for reinforcement learning integrates the ASN and the AEN within a single DCS network used for adaptive vector quantization of the input space, learning of the required sensory–motor mapping and evaluation of the control policy (see Fig. 1). Contrary to GARIC our controller is able to generate new control rules in addition to adapting parameters of existing ones. Utilizing a REINFORCE algorithm we avoid the rather questionable heuristics used in GARIC and, moreover, we are able to specify and adapt a certainty value for each control rule.

The work of Kröse et al. [13] is remarkable in that they were among the first to use adaptive perceptualization (adaptive quantization of the input space) with a growing/shrinking network of locally tuned units for reinforcement learning. Yet they use a discrete representation of the input space (Voronoi tessellation) and their action space is discrete as well (only two actions, turning either left or right), which makes their work hardly comparable to ours.

Obstacle avoidance using sonar sensors and fuzzy logic without adaptivity has been tackled by e.g. Reignier [18]. In [19] he elaborates on generating new fuzzy rules using supervised learning, much of which can be directly transferred to the generation of new neural units in our architecture.

With regard to collision avoidance our approach is finally related to potential field approaches, [12], generating vector fields for obstacle avoiding navigation. The main difference is that the potential functions involved in generating the potential field need to be determined a priori whereas we attempt to learn and adapt the required sensory–motor mapping. Moreover, vector fields represented by universal function approximators like neural networks can be far more general than vector fields obtained as gradients of a potential field (always having zero rotation).

6. Summary

We have presented an integrated architecture based on DCS for learning reactive behaviors. This architecture was shown to fulfill all the requirements put forward in Section 1. Due to utilizing locally tuned units and hence the resemblance to RBF networks the architecture is capable of incremental as well as one-shot learning. Different from RBF networks DCS learn perfectly topology preserving feature maps which are exploited for rapid and accurate output calculation and on-line adaptation. Bearing close resemblance to a restricted class of Sugeno-type fuzzy controllers the incorporation of prior knowledge in form of fuzzy rules is straightforward, this avoiding learning

from scratch. Not relying on only one but instead exploiting different learning paradigms seems to be a key to practical success in learning reactive behaviors and is a red thread through the recent literature on this subject. Our DCS-based architecture is particularly well suited to integrate different learning paradigms and to learn from different error signals.

We have tested our controller architecture on the task of learning reactive collision avoidance from delayed reinforcement for a simulated TRC Labmate. Conditions were unusually hard using unprocessed readings from eight sonar sensors as input and trying to learn a continuous forward and angular velocity. Our experiments confirm that even in the absence of prior domain knowledge the controller is able to learn collision avoidance from reinforcement and, furthermore, indicate that the controller in spite of remaining plasticity behaves stable. At the heart of reinforcement learning we employed a REINFORCE algorithm, which besides yielding a sound basis for reinforcement learning adds to interpretability by giving rise to certainty values. We utilized REINFORCE gradient descent, temporal differencing, Kohonen-type learning, Hebbian learning and TD-error controlled growth of the network.

Learning from reinforcement is usually very slow as underlined by our experiments. Hence the necessity for augmentative mechanisms to support pure reinforcement learning when turning to the physical robot. These are additionally learning mechanisms (e.g. some self-supervised or supervised learning) and, of course, incorporation of prior knowledge. Our architecture allows for both.

References

[1] L. Ahrns, Ultraschallbasiert Navigation und adaptive Hindernisvermeidung eines autonomen mobilen Roboters, Master Thesis, Inst. f. Inf. u. Prakt. Math., CAU zu Kiel, 1996.
[2] E. Alpaydin, GAL: Networks that grow when they learn and shrink when they forget, Technical Report 91-032, International Computer Science Institute, Berkeley, CA, 1991.
[3] H.R. Beom and H.S. Cho, A sensor-based navigation for a mobile robot using fuzzy logic and reinforcement learning, *IEEE Transactions on Systems, Man and Cybernetics* 25 (3) (1995) 464–477.
[4] H.R. Berenji and P. Kheda, Learning and tuning fuzzy logic controllers through reinforcements, *IEEE Transactions on Neural Networks* 3 (5) (1992) 724–740.
[5] J. Bruske, I. Ahrns and G. Sommer, On-line learning with dynamic cell structures, *ICANN'95* (1995) 141–146.
[6] J. Bruske, I. Ahrns and G. Sommer, Practicing Q-learning, *Proceedings of ESANN'96* (1996) 25–30.
[7] J. Bruske, M. Hansen, L. Riehn and G. Sommer, Adaptive saccade control of a binocular head with dynamic cell structures, *ICANN'96* (1996) 215–220.
[8] J. Bruske and G. Sommer, Dynamic cell structure learns perfectly topology preserving map, *Neural Computation* 7 (4) (1995) 845–865.
[9] B. Fritzke, Growing cell structures – a self-organizing network for unsupervised and supervised learning, *Neural Networks* 7 (9) (1995) 1441–1460.
[10] V. Gullapalli, A stochastic reinforcement learning algorithm for learning real-valued functions, *Neural Networks* 3 (1990) 671–692.
[11] W. Ilg and K. Berns, A learning architecture based on reinforcement learning for adaptive control of the walking machine LAURON, *Robotics and Autonomous Systems* 15 (1995) 321–324.
[12] O. Khatib, Real-time obstacle avoidance for manipulators and mobile robotics international, *Journal of Robotics Research* 5 (1) (1986) 90–98.
[13] B. Kroese and J. Dam, Adaptive state space quantization for reinforcement learning of collision-free navigation, *Proc. Int. Conf. on Int. Robots and Systems* (1992) 1327–1333.
[14] T. Martinetz and K. Schulten, Topology representing networks, *Neural Networks* 7 (1994) 505–522.
[15] J.R. Millan, Reinforcement leading of goal-directed obstacle avoiding reaction strategies in an autonomous mobile robot, *Robotics and Autonomous Systems* 15 (1995) 275–299.
[16] D. Nauck, F. Klawonn and R. Kruse, *Neuronale Netze und Fuzzy-Systeme* (Vieweg, Braunschweig, 1994).
[17] T. Prescott and J. Mayhew, Obstacle Avoidance through reinforcement learning, *NIPS* (1992) 523–530.
[18] P. Reignier, Fuzzy logic techniques for mobile robot obstacle avoidance, *Robotics and Autonomous Systems* 12 (1994) 143–153.
[19] P. Reignier, Supervised incremental learning of fuzzy rules, *Robotics and Autonomous Systems* 16 (1995) 57–71.
[20] C. Scheit, Incremental category learning in a real world artifact using growing dynamic cell structures, *Proceedings of ESANN'96* (1996) 117–122.
[21] R.S. Sutton, Learning to predict by the methods of temporal differences, *Machine Learning* (1) (1988) 9–44.

[22] C. Torass, Robot adaptivity, *Robotics and Autonomous Systems* 15 (1995) 11–23.
[23] R.J. Williams, Simple statistical gradient-following algorithms for connectionist reinforcement learning, *Machine Learning* (8) (1992) 229–256.

Jörg Bruske studied Computer Science at the University of Kaiserslautern where he graduated in 1993. In 1993 he joined the Cognitive Systems Group at the Computer Science Department of the University of Kiel as a research assistant. He is currently working towards his Ph.D. Thesis concerning theory and application of dynamic cell structures. His research interests include: machine learning with special emphasis on artificial neural networks; autonomous mobile robots; computer vision and pattern recognition.

Ingo Ahrns studied Computer Science at the University of Kiel where he graduated in 1996. In his awarded diploma thesis he extended the behavior based mobile robot architecture of M. Mataric using fuzzy logic, dynamic cell structures and reinforcement learning. Recently he started working towards his Ph.D. Thesis concerning depth estimation in robotics applications at the Daimler Benz Research Center at Ulm. His main research interests are artificial neural networks and autonomous mobile robots with emphasis on computer vision.

Dr. Gerald Sommer studied Physics at the University of Jena where he received his Dr. rer. nat. degree in 1975. In 1988 he obtained his habilitation from the Technical University of Ilmenau. He headed the medical image analysis group of the MEDIS institute of the GSE Neuherberg from 1991 to 1993. In 1993 he took the position of University Professor in Computer Science at the Computer Science Department of the University of Kiel and founded the Cognitive Systems Group. His general research activities are related to computer vision and robotics, with emphasis on early visual processing.