EXTRAPOLATION WITH A SELF-ORGANISING LOCALLY INTERPOLATING MAP

Controlling nonlinear Processes with ambiguous inverse Behaviour

Helge Hülsen, Sergej Fatikow
Division of Microrobotics and Control Engineering, University of Oldenburg
Uhlhornsweg 84, 26111 Oldenburg, Germany

Keywords: Associative Networks, Self-Organisation, Self-Supervised Learning, Extrapolation.

Abstract: Besides their typical classification task, Self-Organizing Maps (SOM) can be used to approximate input-output relations. They provide an economic way of storing the essence of past data into input/output support vector pairs. In this paper the SOLIM algorithm (Self-Organising Locally Interpolating Map) is reviewed and an extrapolation method is introduced. This framework allows finding one inverse of a nonlinear many-to-one mapping by exploiting the inherent neighbourhood criteria of the SOM part. Simulations show that the performance of the mapping including the extrapolation is comparable to other algorithms.

1 INTRODUCTION

Several derivates of Self-Organizing Maps (SOM) have been successfully applied to the control of nonlinear, dynamic processes, which can also have ambiguous inverse system behaviour (Barreto 2003).

A very important enhancement of the classical SOM from Kohonen (overview in (Kohonen 2001)) has been the Local Linear Map (LLM) from Ritter et al. (Ritter 1992) that has been further developed and applied to various problems in robotics (Moshou 1997) and system dynamics modelling (Principe 1998)(Cho 2003). An LLM not only divides an input space into subspaces as a standard SOM does but assigns a local linear model to each subspace and thus performs a mapping to an output space. By descending on the error function of the local models better estimates for these models are found. Remarkable was the ability of the algorithm to learn a meaningful mapping for processes with an ambiguous inverse behaviour. This effect results from applying Kohonen's self-organising rule to update the local model that has been responsible for the output and all of its neighbours. The drawbacks of the LLM are its discontinuities in the mapping at the transitions between neighboured local models and the strong dependency of the learning performance on the Jacobians that define the linear models.

To solve the discontinuity problem Aupetit et al. developed a continuous version of the LLM, the Continuous Self-Organizing Map (CSOM) (Aupetit 1999)(Aupetit 2000), but still the mapping and the learning was depending on the Jacobians and, in addition, depending on interpolation parameters.

Walter followed another approach by avoiding a discretisation of the input space and sharing the influence of each model on the output (Walter 1997)(Walter 2000). The influences are found with help of an optimisation algorithm. On one hand this leads to a continuous mapping, which can be optimised with respect to different user-defined criteria to resolve ambiguities. On the other hand this optimisation can be a relatively high computational load for high network dimensions. Furthermore, under certain conditions, there are problems with extrapolation and the continuity of the mapping.

The Self-Organising Locally Interpolating Map (SOLIM) from the author (Hülsen 2004a)(Hülsen 2004b) interpolates between different models without any additional parameters. Learning is performed in a self-supervised structure, which can be interpreted as identification with exchanged input and output. The main drawback of this algorithm is the high computational effort for big networks, which is in great part due to the extrapolation principle. In addition, extrapolation was only performed in

Hülsen H. and Fatikow S. (2005).
EXTRAPOLATION WITH A SELF-ORGANISING LOCALLY INTERPOLATING MAP - Controlling nonlinear Processes with ambiguous inverse Behaviour.
In *Proceedings of the Second International Conference on Informatics in Control, Automation and Robotics - Signal Processing, Systems Modeling and Control*, pages 173-178
DOI: 10.5220/0001177801730178

a small area around the main mapping area. This paper will introduce a new extrapolation algorithm for the SOLIM.

The paper is organised as follows: In the next section the fundaments of the SOLIM algorithm will be explained. In section three the new extrapolation principle will be presented in detail, followed by simulations in section four. The paper ends with the conclusion.

2 SOLIM: SELF-ORGANIZING LOCALLY-INTERPOLATING MAP

The SOLIM is a framework that shall perform two tasks in a control context (Figure 1):

1. Map from a desired state of a process $\mathbf{g}_d$ to actuation parameters $\mathbf{p}_a$ to reach that state.
2. Use that pair of measured state $\mathbf{g}_m$ and actuation parameters $\mathbf{p}_a$ to adapt the mapping.

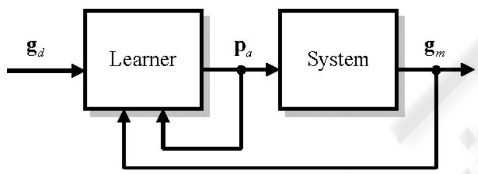

Figure 1: Self-supervised learning (Barreto 2004)

2.1 Mapping

As in a standard SOM (Hagan 1996), the base of SOLIM is a grid c of N neurons with a certain topology (Figure 2). Each neuron i is connected to an input support vector $\mathbf{g}_i \in G$ and an output support vector $\mathbf{p}_i \in P$. To perform the mapping $G \to P$ an influence weight f_i with respect to the input vector $\mathbf{g}_d$ is calculated for each neuron. The output vector $\mathbf{p}_a$ is then the linear combination of all output support vectors

$$\mathbf{p}_a = \sum_i \tilde{f}_i \cdot \mathbf{p}_i , \tag{1.1}$$

where the weight f_i is scaled down to not exceed 1

$$\tilde{f}_i = \begin{cases} \frac{f_i}{\sum_k f_k} & \text{if } \sum_k f_k > 1 \\ f_i & \text{otherwise} \end{cases} . \tag{1.2}$$

The influence weights f_i are measures of how close the input vector $\mathbf{g}_d$ is to the corresponding input support vectors $\mathbf{g}_i$. The influence weight is 1 for $\mathbf{g}_d = \mathbf{g}_i$ and decreases to 0 at the limits of the influence range of neuron i (Figure 3).

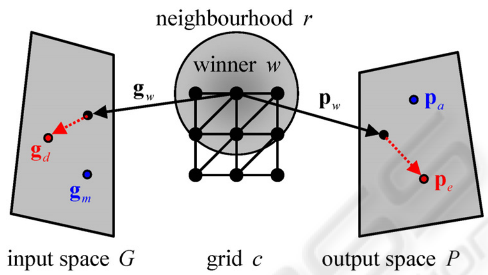

Figure 2: Mapping and learning with self-organizing maps (adopted from (Ritter 1992))

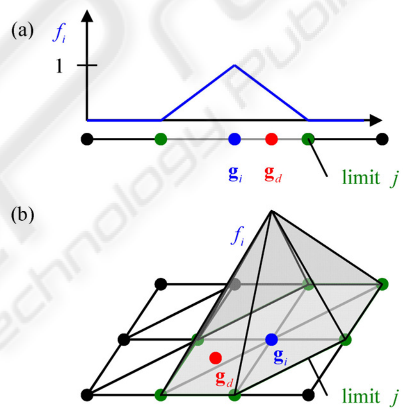

Figure 3: Influence f_i of neuron i for (a) a 1D grid in a 1D input space and (b) a 2D grid in a 2D input space

The limit j of an influence range is defined solely by a selection of input support vectors $\mathbf{g}_j$. For the case of a 1D grid there are two limits for each input support vector (Figure 3(a)). When the grid is placed in a 1D input space the limits are points, in a 2D input space the limits are lines and in a 3D input space the limits are planes. Now considering a 2D grid there are six limits for each input support vector (Figure 3(b)). When the grid is placed in a 2D input space the limits are lines and in a 3D input space the limits are planes again. In input spaces with higher dimensions the limits will be hyperplanes.

A limit j can generally be defined by a position vector $\mathbf{g}_{ij}$ and a plane normal $\mathbf{n}_{ij}$, whose calculation can be found in (Hülsen 2004b). The influence weight f_{ij} with respect to a limit j is calculated with help of the relative distance d_{ij} of the input vector $\mathbf{g}_d$ from $\mathbf{g}_i$ in the direction to the limit plane

$$d_{ij} = \frac{\left|\mathbf{n}_{ij} \cdot (\mathbf{g}_d - \mathbf{g}_i)\right|}{\left|\mathbf{n}_{ij} \cdot (\mathbf{g}_{ij} - \mathbf{g}_i)\right|}. \tag{1.3}$$

A blending function $f_{ij} = b(d_{ij})$ sets the influence to 0 for $d_{ij} > 1$, to 1 for $d_{ij} < 0$ and defines a transition for $0 < d_{ij} < 1$, i.e. the change of the influence from one neuron to another.
The influences to the limits j are then combined to the influence of the neuron i on the output vector

$$f_i = \min_j (f_{ij}). \tag{1.4}$$

Finally, it should be noted that the limits are defined in a way that for a 1D grid two neurons are responsible for the output vector, for a 2D grid three neurons are responsible, for a 3D grid four neurons are responsible, and so on (see also (Hülsen 2004b)).

2.2 Learning

Learning is performed in the input space G as well as in the output space P with help of the Kohonen learning rule (Kohonen 2001). Following the rule not only the support vector of the "winner-neuron" w is updated but to a certain extent $h(w,i,r)$ also the support vectors belonging to a neighbourhood r in the grid c (see Figure 2). In case of learning in the input space the winner w is the neuron with the highest influence with respect to the input vector $\mathbf{g}_d$, which in turn serves as an attractor for $\mathbf{g}_w$ and its neighbours

$$\mathbf{g}_i^{(\nu)} = \mathbf{g}_i^{(\nu-1)} + \varepsilon_g \cdot h(w,i,r_g) \cdot \left(\mathbf{g}_d^{(\nu)} - \mathbf{g}_i^{(\nu-1)}\right). \tag{1.5}$$

ε_g is a learning constant that, as the neighbourhood radius r_g, is decreased with time. In case of learning in the output space the winner w is the neuron with the highest influence with respect to the measured process output $\mathbf{g}_m$. Since the process input $\mathbf{p}_a$ belongs to $\mathbf{g}_m$, it can be used to find an estimate $\mathbf{p}_e$ for $\mathbf{p}_w$ by solving (1.1) for $\mathbf{p}_{i=w}$

$$\mathbf{p}_e = \frac{1}{\tilde{f}_w}\left(\mathbf{p}_a - \sum_{i \neq w} \tilde{f}_i \cdot \mathbf{p}_i\right). \tag{1.6}$$

$\mathbf{p}_e$ then serves as attractor for $\mathbf{p}_w$ and its neighbours during the Kohonen update rule

$$\mathbf{p}_i^{(\nu)} = \mathbf{p}_i^{(\nu-1)} + \varepsilon_p \cdot h(w,i,r_p) \cdot \left(\mathbf{p}_e^{(\nu)} - \mathbf{p}_i^{(\nu-1)}\right). \tag{1.7}$$

The Kohonen learning rule is topology conserving, which means that support vectors of neurons that are neighbours in the grid c become neighbours in the input space and output space. This property is an inherent criterion to resolve ambiguities in the inverse system behaviour $G \to P$, since neighbours in the input space map to neighbours in the output space.

3 EXTRAPOLATION WITH SOLIM

The main contribution of this paper is to show that extrapolation is possible within the context of the SOLIM-algorithm. Like the SOLIM-interpolation the extrapolation algorithm only needs to know the support vectors in the input and output space to perform a reasonably accurate extrapolation. The learning algorithm can be adapted to this enhanced mapping in a straight-forward manner.

3.1 Mapping

The mapping for regions outside the grid of input support vectors is performed by adding an extrapolation component $\mathbf{p}_{xi}$ to each support vector $\mathbf{p}_i$ prior to interpolating between them (Figure 4)

$$\mathbf{p}_a = \sum_i \tilde{f}_i \cdot (\mathbf{p}_i + \mathbf{p}_{xi}). \tag{1.8}$$

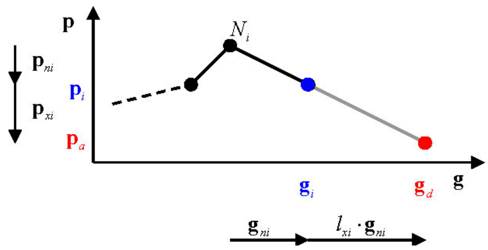

Figure 4: Extrapolation component $\mathbf{p}_{xi}$ corresponds to distance $l_{xi} \cdot \mathbf{g}_{ni}$ of input vector from grid (1D grid)

The extrapolation component is computed as

$$\mathbf{p}_{xi} = l_{xi} \cdot \mathbf{p}_{ni}, \tag{1.9}$$

where l_{xi} is a weight that defines the distance of the input vector $\mathbf{g}_d$ from the input support vector $\mathbf{g}_i$ in relation to $\mathbf{g}_{ni}$ (Figure 4, Figure 5)

$$l_{xi} = \frac{(\mathbf{g}_d - \mathbf{g}_i)\mathbf{g}_{ni}}{\mathbf{g}_{ni}^2}. \tag{1.10}$$

$\mathbf{g}_{ni}$ is the mean difference vector between the input support vector $\mathbf{g}_i$ and all input support vectors in its limiting neighbourhood N_i

$$\mathbf{g}_{ni} = \sum_{k \in N_i} (\mathbf{g}_i - \mathbf{g}_k). \tag{1.11}$$

Analogously, $\mathbf{p}_{ni}$ is the mean difference vector between the output support vector $\mathbf{p}_i$ and all output support vectors in its limiting neighbourhood N_i

$$\mathbf{p}_{ni} = \sum_{k \in N_i} (\mathbf{p}_i - \mathbf{p}_k). \tag{1.12}$$

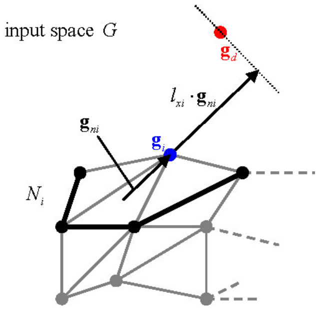

Figure 5: Calculation of $l_{xi} \cdot \mathbf{g}_{ni}$ in a 2D grid

The main idea behind the interpolation as well as the extrapolation mapping is that the relation between input vector and input support vector grid is similar to the relation between output vector and output support vector grid (Table 1). The weights $\tilde{f}_i$ and l_{xi} can therefore be applied in the input space in the same way as in the output space. In addition, since the calculation of the weights only depends on the dimension of the grid c the mapping can be applied in both directions.

Table 1: Similarity of mapping in input and output space

input space	output space
$\mathbf{g}_d \approx \sum_i \tilde{f}_i \cdot (\mathbf{g}_i + \mathbf{g}_{xi})$	$\mathbf{p}_a = \sum_i \tilde{f}_i \cdot (\mathbf{p}_i + \mathbf{p}_{xi})$
$\mathbf{g}_{xi} = l_{xi} \cdot \mathbf{g}_{ni}$	$\mathbf{p}_{xi} = l_{xi} \cdot \mathbf{p}_{ni}$
$\mathbf{g}_{ni} = \sum_{k \in N_i} (\mathbf{g}_i - \mathbf{g}_k)$	$\mathbf{p}_{ni} = \sum_{k \in N_i} (\mathbf{p}_i - \mathbf{p}_k)$

3.2 Learning

The learning algorithm remains unchanged except that the calculation of the estimation $\mathbf{p}_e$ for the support vector $\mathbf{p}_w$ that is most responsible for the system output $\mathbf{g}_m$ (compare (1.6)) must take the extrapolation components into account. Therefore (1.8) must be solved for $\mathbf{p}_{i=w}$, using (1.9) and (1.12)

$$\mathbf{p}_e = \frac{1}{1 + l_{xw} \cdot k} \left(\begin{array}{l} \frac{1}{\tilde{f}_w} \left(\mathbf{p}_a - \sum_{i \neq w} \tilde{f}_i \cdot (\mathbf{p}_i + \mathbf{p}_{xi}) \right) \\ + l_{xw} \cdot \sum_{k \in N_w} \mathbf{p}_k \end{array} \right). \tag{1.13}$$

It can be seen that when the extrapolation weight for $\mathbf{p}_w$ is $l_{xw} = 0$ (1.13) becomes similar to (1.6), because no extrapolation is performed.

4 SIMULATIONS

The performance of the extrapolation algorithm can be evaluated by mapping with support vectors that represent a 2D Gaussian bell and by learning the inverse of a well-known function.

4.1 Mapping

For a performance measurement of the mapping 5x5 input support vectors $\mathbf{g}_i$ will be placed at the positions $g_{i,1}, g_{i,2} \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$. The corresponding output support vectors $\mathbf{p}_i$ will be placed at the corresponding positions of a 2D Gaussian bell with $\mu = 0.5$ and $\sigma = 0.1$ (Figure 6)

$$\mathbf{p}_i = \exp\left(-\frac{1}{2} \cdot \left(\left(\frac{g_{i,1} - \mu}{\sigma} \right)^2 + \left(\frac{g_{i,2} - \mu}{\sigma} \right)^2 \right) \right). \tag{1.14}$$

For this support vector constellation the SOLIM mapping in the range $g_{d,1}, g_{d,2} \in [0..1]$ is shown in Figure 7. The RMS-error of 0.043 is comparable to other algorithms as stated in Table 2.

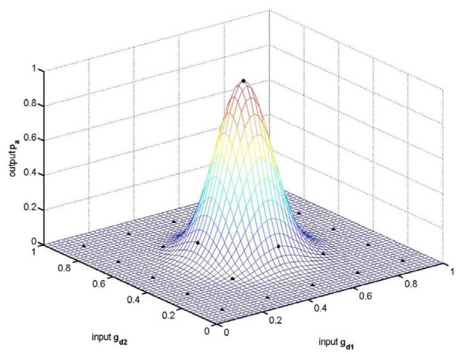

Figure 6: 2D Gaussian bell in the range $g_{d,1}, g_{d,2} \in [0..1]$

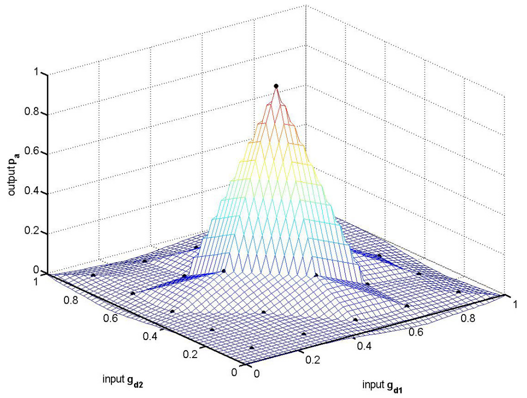

Figure 7: SOLIM-approximation (5x5 neurons) of 2D Gaussian bell in the range $g_{d,1}, g_{d,2} \in [0..1]$

Table 2: RMS-error for approximation of 2D Gaussian bell ($\mu = 0.5$, $\sigma = 0.1$) with 25 support vectors in the range $g_{d,1}, g_{d,2} \in [0..1]$. Values from (Göppert 1997).

Algorithm	RMS-error
local PSOM	0.049
RBF	0.041
CRI-SOM	0.016
SOLIM	0.043

To test the performance of the extrapolation functions the same mapping is shown in Figure 8 for a broader input range $g_{d,1}, g_{d,2} \in [-0.5..1.5]$. The extrapolation is reasonable but the error is relatively large (RMS-error = 0.063) because the falling slope at the borders is continued although the Gaussian bell approaches the zero-plane. This error can be decreased significantly by using more neurons that better represent the slopes, e.g. by placing 7x7 neurons in the same area as shown in Figure 9 (RMS-error = 0.025).

4.2 Learning

To validate the learning algorithm one inverse of the function

$$g = 10 \cdot (\sin(p) + 1/3 \cdot \sin(3p)) \qquad (1.15)$$

shall be learned. In Figure 10 the result after 956 learning steps can be seen, where the RMS-error is 0.1% of the input vector range $g_d \in [-10..10]$. The development of the error can be found in Figure 11. It can be seen that the SOLIM algorithm with the presented extrapolation can find one inverse to a many-to-one, non-linear function.

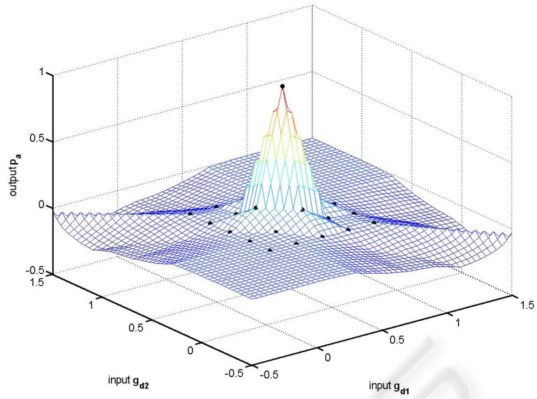

Figure 8: SOLIM-approximation (5x5 neurons) of 2D Gaussian bell in the range $g_{d,1}, g_{d,2} \in [-0.5..1.5]$

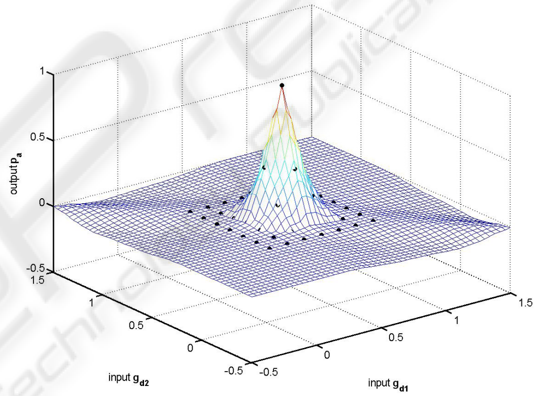

Figure 9: SOLIM-approximation (7x7 neurons) of 2D Gaussian bell in the range $g_{d,1}, g_{d,2} \in [-0.5..1.5]$

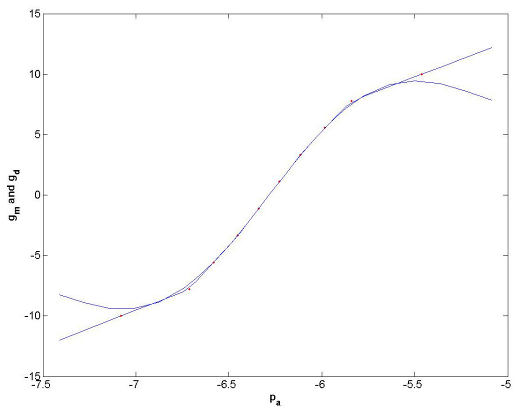

Figure 10: Linear map with 10 neurons fits to one inverse of (1.15) after 956 iteration steps. RMS-error: 0.1% of $\mathbf{g}_d$ -range

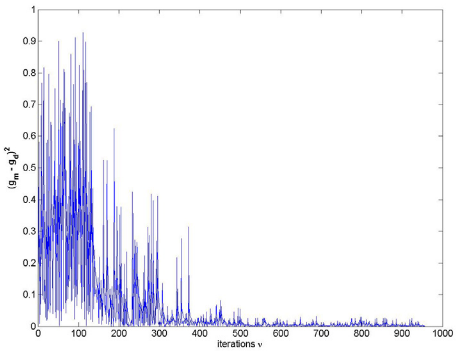

Figure 11: Development of RMS-error

5 CONCLUSION

It can be concluded that the presented algorithm has the following properties:

- The extrapolation results from the interpolation between the slopes at the borders, which is calculated from the border support vectors and their neighbours.
- The interpolation as well as the extrapolation part only needs the input and output support vectors to perform a mapping. No interpolation factors and no local linear model matrices are required. The price is that only 0-order continuity is ensured, which mostly is sufficient.
- The mapping performance is comparable to other algorithms (PSOM, RBF, ...).
- One mapping for ambiguous inverse system behaviour can be found within a sufficient number of iteration steps. Still a comparison to other algorithms is missing since there is no commonly accepted benchmark-system that can be easily set up.

The following problems have not been solved yet:

- The topology is still fixed and must be known a-priori. There exist algorithms that dynamically build topologies and neighbourhood relationships, depending on the input data "structure".
- There are still learning parameters that must be tuned before each experiment and that partially vary depending on time. For online training these parameters must be varied automatically.
- The presented algorithm shall be tested in a real application. One good demonstration is to learn non-linear, time-variant and many-to-one motion characteristics of microrobots (Hülsen 2004a).

REFERENCES

Aupetit, M., Couturier, P., and Massotte, P. (1999). A continuous self-organizing map using spline technique for function approximation. In *Proc. Artificial Intelligence and Control Systems (AICS'99)*, Cork, Ireland.

Aupetit, M., Couturier, P., and Massotte, P. (2000). Function approximation with continuous self-organizing maps using neighboring influence interpolation. In *Proc. Neural Computation (NC'2000)*, Berlin, Germany.

de A. Barreto, G., Araújo, A. F. R., and Ritter, H. J. (2003). Self-organizing feature maps for modeling and control of robotic manipulators. *Journal of Intelligent and Robotic Systems*, 36(4):407-450.

de A. Barreto, G. and Araújo, A. F. R. (2004). Identification and control of dynamical systems using the self-organizing map. *IEEE Transactions on Neural Networks*, 15(5):1244-1259.

Cho, J., Principe, J. C., and Motter, M. A. (2003). A local linear modeling paradigm with a modified counter-propagation network. In *Proc. Int. Joint Conf. on Neural Networks*, pages 34-38, Portland, OR, U.S.A.

Hagan, M. T., Demuth, H. B., and Beale, M. (1996). *Neural Network Design*. PWS Publishing Co., Boston, MA, U.S.A.

Hülsen, H., Trüper, T., and Fatikow, S. (2004a). Control system for the automatic handling of biological cells with mobile microrobots. In *Proc. American Control Conference (ACC'04)*, pages 3986-3991, Boston, MA, U.S.A.

Hülsen, H. (2004b). Design of a fuzzy-logic-based bidirectional mapping for kohonen networks. In *Proc. Int. Symposium on Intelligent Control (ISIC'04)*, pages 425-430, Taipei, Taiwan.

Kohonen, T. (2001). *Self-Organizing Maps*. Springer, Berlin, Germany, 3. edition.

Moshou, D. and Ramon, H. (1997). Extended self-organizing maps with local linear mappings for function approximation and system identification. In *Proc. Workshop on Self-Organizing Maps (WSOM'97)*, pages 181-186, Helsinki, Finland.

Principe, J. C., Wang, L., and Motter, M. A. (1998). Local dynamic modelling with self-organizing maps and applications to nonlinear system identification and control. *Proceedings of the IEEE*, 86(11):2240-2258.

Ritter, H., Martinetz, T., and Schulten, K. (1992). *Neural Computation and Self-Organizing Maps: An Introduction*. Addison-Wesley, Reading, M.A., U.S.A.

Walter, J., Nölker, C., and Ritter, H. (2000). The PSOM algorithm and applications. In *Proc. of Int. Symp. on Neual Computation (NC'2000)*, pages 758-764, Berlin, Germany.

Walter, J. (1997). *Rapid Learning in Robotics*. Cuvillier Verlag, Göttingen. http://www.techfak.uni-bielefeld.de/ walter/.