University of Würzburg
Institute of Computer Science
Research Report Series

An approach for the identification of nonlinear, dynamic processes with Kalman-Filter-trained recurrent neural structures

Dipl. Inform. Frank Heister,
Dr. Rainer Müller

Report No. 193 April 1999

Research and Technology (FT2/EA)
Electronic Architecture and Networking
DaimlerChrysler AG
HPC T721
D-70546 Stuttgart, Germany
[frank.heister|rainer.m.mueller]@daimlerchrysler.com

An approach for the identification of nonlinear, dynamic processes with Kalman-Filter-trained recurrent neural structures

Dipl. Inform. Frank Heister,
Dr. Rainer Müller
Research and Technology (FT2/EA)
Electronic Architecture and Networking
DaimlerChrysler AG
HPC T721
D-70546 Stuttgart, Germany
[frank.heister|rainer.m.mueller]@daimlerchrysler.com

Abstract

In this article we demonstrate the identification of a nonlinear, dynamic process with recurrent neural structures. The employed network-structure is a Recurrent Multilayer Perceptron (RMLP), which combines feedforward– and recurrent architectures. We will show that RMLPs are capable of learning the temporal behavior and characteristic of an arbitrary, nonlinear, dynamic process. Apart from conventional gradient-based algorithms, a sophisticated statistical method has been considered for this challenging task — Global Extended Kalman Filtering (GEKF). This powerful algorithm yields neural structures with a significantly better performance, compared to conventional gradient-based approaches. The new element in this work is the application of the GEKF-Algorithm for recurrent neural structures, which are employed in the identification of nonlinear, dynamic processes. In order to supervise the quality of network-training, appropriate performance-indexes for neural identification are introduced. The distribution of the Moving Average Squared Error (MASE) is employed as an objective optimality-criterion, in order to survey the actual performance of recurrent neural structures during training.

1 Introduction

Since the early 1990s, a growing interest in the field of neural processing and adaptive algorithms could be observed, concerning their employment for the identification and control of technical processes. Mathematical systems theory, which has evolved into a scientific discipline with wide applicability over the past five decades, deals with the analysis and synthesis of mathematical models of dynamic processes. This theory provides

methods for the treatment of dynamic systems, using well-established techniques based on linear algebra, complex variable theory and the theory of ordinary linear differential equations. The drawback of conventional approaches is their potential complexity, in particular if a high level of refinement and accuracy is required. Hence, a satisfying solution, which describes a given process, does not necessarily need to exist; even for apparently simple systems.

In order to provide a feasible, supplementary approach toward reducing the complexity of analytical model– and controller-design, a new approach has been considered for this task — adaptive, recurrent structures with a sophisticated training-algorithm, the *Global Extended Kalman Filter* (GEKF).

An outline for a typical life-time-cycle of a controller-design is depicted in Fig. 1, cf. [1]. Designing a feedback-controller requires an initial process-model. If the model proves to be insufficient it has to be successively refined. However, if conventional methods are used, a refinement-step results in an increased complexity of the mathematical model, as mentioned above.

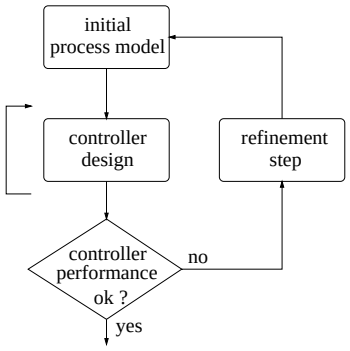

Figure 1: Controller design.

In order to overcome the complexity of conventional methods an alternative approach has to be chosen, which requires less background-knowledge about the process being modeled. Adaptive structures, such as recurrent neural networks, are capable of providing an implicit representation of the dynamics of a process rather than an exact mathematical solution. The major drawback of a neural approach is the problem of stability, which cannot be guaranteed from the outset. However, since an exact solution might be very expensive, sometimes even unfeasible, adaptive systems are expected to be a good compromise to obtain an initial process-model.

Identification of the temporal behavior of a system is equivalent with the characterization and modeling of relevant process-variables and their interaction. Ideally, identification produces an exact image of the real process-behavior. For real-world applications, this assumption turns out to be illusory, due to the relation between the desired degree of accuracy and the complexity of the process-model.

Mathematical methods for the treatment of nonlinear systems are introduced by [2] and [1]. However, the presented methods are always tied to a specific class of systems and suffer from the lack of general applicability. The desire for a general framework, suitable for the identification of arbitrary nonlinear, dynamic processes, motivates a new approach — Recurrent Multilayer Perceptrons with Extended Kalman Filter Training.

This class of recurrent neural networks combines the topologies of conventional Multilayer Perceptrons with those of general recurrent network-structures, eg. Hopfield Networks.

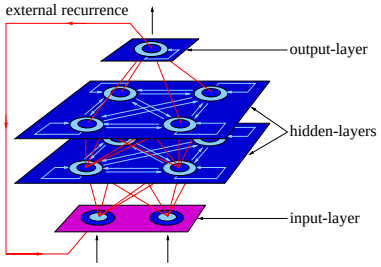

Figure 2: Recurrent Multilayer Perceptron (RMLP).

As depicted in Fig. 2, a RMLP consists of successive layers with no recurrent weight-connections between them. External recurrences between nodes in the output-layer and nodes in the input-layer of a RMLP are also admissible. Since there are distinct back– and forth-connections between two particular neurons, a particular layer could be regarded as an "extended Hopfield-Network".

1.1 Representation and characterization of processes

Characterizing a process means determining all static and dynamic factors effecting its behavior. The term *dynamic* refers to the temporal behavior of the process itself, as well

as to its parameters. However, dynamic systems can be divided into two major groups: *discrete-time systems* and *continuous-time systems*.

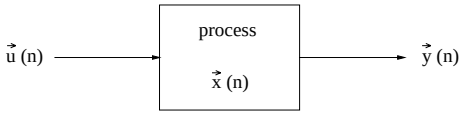

Figure 3: Characterization of a process.

In Fig. 3, a discrete-time system is depicted. The vector $\vec{x}(n)$ represents the *internal state* of the process. Depending on the dimensions of the vectors $\vec{u}(\cdot)$ and $\vec{y}(\cdot)$, the process is denoted a SISO-System (Single-Input-Single-Output), respectively a SIMO–, MISO– or a MIMO-System.

Time-invariant, linear Systems

A *time-invariant, linear system* can be described in *discrete-time* [3] by the linear equations:

$$\begin{aligned} \vec{x}(n+1) &= A\vec{x}(n) + B\vec{u}(n) \\ \vec{y}(n) &= C\vec{x}(n) + D\vec{u}(n), \end{aligned} \tag{1}$$

in *continuous-time*[1] by the linear *differential equations*:

$$\begin{aligned} \dot{\vec{x}}(t) &= A\vec{x}(t) + B\vec{u}(t) \\ \vec{y}(t) &= C\vec{x}(t) + D\vec{u}(t), \end{aligned} \tag{2}$$

where A, B, C, D are system-matrices of appropriate dimensions.

Time-invariant, nonlinear systems

Any *time-invariant system* which cannot be expressed using (1) or (2) is defined to be *nonlinear*. Nonlinear systems are described in *discrete-time* by:

$$\begin{aligned} \vec{x}(n+1) &= f(\vec{x}(n), \vec{u}(n)) \\ \vec{y}(n) &= h(\vec{x}(n), \vec{u}(n)), \end{aligned} \tag{3}$$

[1] The variable t is used in order to indicate that continuous time is assumed.

in *continuous-time* by:

$$\begin{aligned} \dot{\vec{x}}(t) &= f(\vec{x}(t), \vec{u}(t)) \\ \vec{y}(t) &= h(\vec{x}(t), \vec{u}(t)), \end{aligned} \tag{4}$$

where $f(\cdot), h(\cdot)$ are nonlinear, vector-valued functions.

Time-variant, nonlinear systems

The systems described so far are time-invariant, i.e. their behavior does not change. However, in reality some effects, e.g. ageing processes, might cause the system-behavior to change with time. Hence, time must also be taken into account as a factor, effecting the system-behavior.

In *discrete-time* Eqn. 3 becomes:

$$\begin{aligned} \vec{x}(n+1) &= f(\vec{x}(n), \vec{u}(n), n) \\ \vec{y}(n) &= h(\vec{x}(n), \vec{u}(n), n). \end{aligned} \tag{5}$$

In *continuous-time* Eqn. 4 becomes:

$$\begin{aligned} \dot{\vec{x}}(t) &= f(\vec{x}(t), \vec{u}(t), t) \\ \vec{y}(t) &= h(\vec{x}(t), \vec{u}(t), t). \end{aligned} \tag{6}$$

1.2 Identification with neural networks

Modified feedforward-structures such as tapped delay lines (TDLs), have been previously applied for modeling nonlinear, dynamic systems by [4]. This technique allows only a limited number of information to be considered for obtaining a suitable process-model. Assuming the utilization of recurrent neural structures, an arrangement used for identification is depicted in Fig. 4. The resulting recurrent network implicitly represents the relevant process-parameters, which are obtained by considering only the input– and output-signal of a real system.

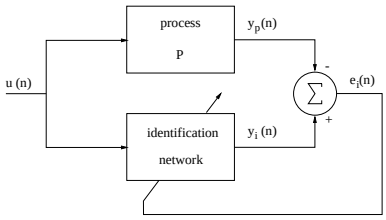

Figure 4: Identification with a neural network.

Since no thorough investigation of the intrinsics is necessary, the required amount of expert-knowledge is reduced considerably, compared to conventional approaches. After successful identification, the neural network is capable of imitating the real system, ie. to behave like the original process, being exposed to the same input-sequence $u(\cdot)$. Obtaining this *identification-network* is essential for further implementation of neural feedback-controllers.

2 Training recurrent neural structures

In this section, two training-algorithms for recurrent neural networks are presented — Real-Time-Recurrent-Learning (RTRL) and the Global-Extended-Kalman-Filter Algorithm (GEKF).

2.1 Real Time Recurrent Learning

RTRL is, as well as standard Backpropagation, a pure gradient-based algorithm for general structured neural networks, which was proposed by [5].

The output of a particular neuron is governed by the following equation:

$$o_j(n+1) = \sigma\left(\sum_{i=1}^{k}(w_{i,j}o_i(n)) + u_j(n+1)\right) \tag{7}$$

Similar to standard Backpropagation this yields for the adaption of a particular weight $w_{i,j}$:

$$\Delta w_{i,j} = -\eta \frac{\partial E(n)}{\partial w_{i,j}} = \eta \sum_{k=1}^{M} E_k(n) \frac{\partial o_k(n)}{\partial w_{i,j}}, \tag{8}$$

where M is the number of output-neurons and η denotes the learning rate.

Depending on the structure of the network being used, two possibilities of determining the partial derivatives $\frac{\partial o_k(n)}{\partial w_{i,j}}$ are presented, which are refered to as *static* and *dynamic* derivatives.

Static derivatives

The partial derivatives $\frac{\partial o_k(n)}{\partial w_{i,j}}$ presented in Eqn. 8, denoting *static* derivatives, are obtained by differentiating Eqn. 7 with respect to a particular weight $w_{i,j}$.

$$\frac{\partial o_k(n)}{\partial w_{i,j}} = \frac{\partial}{\partial w_{i,j}} \left(\sigma(net_k(n))\right) = \sigma'(net_k(n)) \left[\delta_{i,k} o_i(n-1) + \sum_m w_{m,k} \frac{\partial (o_m(n-1))}{\partial w_{i,j}}\right] \tag{9}$$

where $\delta_{i,k}$ is the Kronecker symbol:

$$\delta_{i,j} =: \begin{cases} 1 & , \quad i = j \\ 0 & , \quad i \neq j. \end{cases} \tag{10}$$

When employing static derivatives, no particular network structure is required, ie. static derivatives can be applied to networks with arbitrarily connected neurons. A less general method, which allows spatial– and temporal dependencies between particular neurons to be taken into account, are *dynamic derivatives*.

Dynamic derivatives

Using dynamic derivatives [6], requires a RMLP, depicted in Fig. 2. According to this definition, each layer is treated as a separate subnetwork. The output $y_{i,j}(n)$ of a neuron

j in layer i is a function of the output-vector $\vec{y}_{i-1}(n)$ of the preceding layer in time-step n, the output-vector $\vec{y}_i(n-1)$ of the actual layer in the preceding time-step and the weight-vector $\vec{w}_i$ of the i-th subnet.

The output of neuron j in subnet i is governed by:

$$y_{i,j}(n) = F\left(\vec{y}_{i-1}(n), \vec{y}_i(n-1), \vec{w}_i\right), \tag{11}$$

where

$$F\left(\vec{y}_{i-1}(n), \vec{y}_i(n-1), \vec{w}_i\right) = \sigma\left(\sum_{p=1}^{N_{i-1}} w_{p,j}^{f,i} y_{i-1,p}(n) + \sum_{p=1}^{N_i} w_{p,j}^{r,i} y_{i,p}(n-1)\right) \tag{12}$$

and

$\vec{y}_{i-1}(n)$	is the output vector of subnet $i-1$ at time-step n,
$\vec{y}_i(n-1)$	the output vector of subnet i at time-step $n-1$,
$\vec{w}_i$	the weight vector of subnet i,
$y_{i,j}$	the activation of neuron j in layer i,
$w_{k,j}^{r,i}$	the recurrent weight from neuron k to neuron j in layer i and
$w_{k,j}^{f,i}$	the feedforward weight from neuron k in layer $i-1$ to neuron j in layer i.

Similar to Eqn. 8, the rule for adapting a particular weight $w_{k,j}^{x,g}$ is:

$$\Delta w_{k,j}^{x,g} = -\eta \frac{\partial E(n)}{\partial w_{k,j}^{x,g}} = \eta \sum_{p=1}^{N_l} E_p(n) \frac{\overline{\partial} y_{l,p}(n)}{\overline{\partial} w_{k,j}^{x,g}}, \tag{13}$$

where N_l is the number of neurons in layer l (the output layer of the RMLP) and

$$x = \begin{cases} r, \text{ for recurrent weights} \\ f, \text{ for feedforward weights.} \end{cases}$$

Compared to static derivatives, the overlined notation in Eqn. 13 points out the dynamic character of this type of derivative, which is obtained by differentiating Eqn. 12 with

respect to a particular weight $w_{k,j}^{x,g}$ and application of the general chainrule. The symbol $\delta_{g,i}$ in the last term of Eqn. 14 denotes the Kronecker symbol.

$$\begin{aligned}\frac{\overline{\partial} y_{i,j}(n)}{\overline{\partial} w_{k,j}^{x,g}} &= \frac{\partial \left(F(\vec{y}_{i-1}(n), \vec{y}_i(n-1), \vec{w}_i)\right)}{\partial w_{k,j}^{x,g}} = \\ &\sum_{p=1}^{N_{i-1}} \frac{\partial y_{i,j}(n)}{\partial y_{i-1,p}(n)} \frac{\overline{\partial} y_{i-1,p}(n)}{\overline{\partial} w_{k,j}^{x,g}} + \\ &\sum_{p=1}^{N_i} \frac{\partial y_{i,j}(n)}{\partial y_{i,p}(n-1)} \frac{\overline{\partial} y_{i,p}(n-1)}{\overline{\partial} w_{k,j}^{x,g}} + \\ &\frac{\partial y_{i,j}(n)}{\partial w_{k,j}^{x,g}} \delta_{g,i}. \end{aligned} \tag{14}$$

The partial derivatives $\frac{\partial y_{i,j}(n)}{\partial y_{i-1,p}(n)}$, $\frac{\partial y_{i,j}(n)}{\partial y_{i,p}(n-1)}$ and $\delta_{g,i}\frac{\partial y_{i,j}(n)}{\partial w_{k,j}^{x,g}}$ in Eqn. 14 are found to be:

$$\frac{\partial y_{i,j}(n)}{\partial y_{i-1,p}(n)} = \sigma'(net_{i,j}(n)) w_{p,j}^{f,i} \ , \tag{15}$$

$$\frac{\partial y_{i,j}(n)}{\partial y_{i,p}(n-1)} = \sigma'(net_{i,j}(n)) w_{p,j}^{r,i} \quad and \tag{16}$$

$$\delta_{g,i}\frac{\partial y_{i,j}(n)}{\partial w_{k,j}^{x,g}} = \begin{cases} 0 & , \text{if } g \neq i \\ \sigma'(net_{i,j}(n)) y_{i,k}(n) & , \text{if } g = i. \end{cases} \tag{17}$$

2.2 Global Extended Kalman Filter

For the parameters of *linear* systems with white input– and observation-noise the *Kalman Filter*, proposed by [7], is known to be an optimum estimator. The *Global Extended Kalman Filter*, which employs a linearization around the current point of estimate, can be used to determine the parameters of *nonlinear* systems, in our case the weights of recurrent neural networks.

Conventional applications of the Global Extended Kalman Filter, eg. in radar-tracking devices, are used for the *direct estimation* of actual system parameters. Regarding a recurrent neural network as a dynamic system and its *weights as the parameters to be estimated*, [6] showed the applicability of the GEKF as a training-algorithm for such structures. Compared to conventional, gradient-based training-algorithms, the weights are successively estimated, based on the input-vector $\vec{u}(n)$ in time-step n and the deviation of the network from the desired output. Estimating the weight-vector $\vec{\hat{w}}(n)$ of the neural network is equivalent with the problem of determining the minimum of the expectation value of the mean squared error between the actual weight-vector $\vec{w}(n)$ and its estimation $\vec{\hat{w}}(n)$:

$$E\left[\left(\vec{w}(n)-\vec{\hat{w}}(n)\right)^T\cdot S\cdot\left(\vec{w}(n)-\vec{\hat{w}}(n)\right)\right] \tag{18}$$

The GEKF is proved, for linear systems, to find the minimum of Eqn. 18 by calculating $\vec{\hat{w}}(n)$ from previous estimates. The term *global* refers to the introduction of a covariance-matrix $P(n)$, which describes the dependencies of *all* weights with each other, based on previous estimations and inputs. Due to the remarkable length of the derivation, only the Global-Extended-Kalman-Filter equations are given below. A detailed mathematical background is presented extensively in [8] and [9].

The Global-Extended-Kalman-Filter Equations are:

$$\begin{aligned}
P(n+1) &= P(n)-K(n)\cdot H^T(n)\cdot P(n)+Q(n)\\
K(n) &= P(n)\cdot H(n)\cdot\left[(\eta(n)\cdot S(n))^{-1}+H^T(n)\cdot P(n)\cdot H(n)\right]^{-1}\\
\vec{\hat{w}}(n+1) &= \vec{\hat{w}}(n)+K(n)\cdot\left(\vec{d}(n)-h(\vec{\hat{w}}(n),\vec{u}(n))\right),
\end{aligned}$$

where

$$\vec{d}(r) = \begin{pmatrix} d_1(br) \\ d_2(r) \\ . \\ . \\ . \\ d_m(r) \end{pmatrix}$$

is the desired output-vector of the neural network in time-step r.

$$h\left(\vec{\widehat{w}}(r), \vec{u}(r)\right) = \begin{pmatrix} h_1\left(\vec{\widehat{w}}(r), \vec{u}(r)\right) \\ h_2\left(\vec{\widehat{w}}(r), \vec{u}(r)\right) \\ . \\ . \\ . \\ h_m\left(\vec{\widehat{w}}(r), \vec{u}(r)\right) \end{pmatrix}$$

describes the output-vector of the neural network, depending on the estimation of the weight-vector $\vec{\widehat{w}}(r)$ and the input-vector $\vec{u}(r)$ in time-step r. $h(\cdot)$ represents the observation-function of the neural network.

$$\vec{\widehat{w}}(r) = \begin{pmatrix} \widehat{w}_1(r) \\ \widehat{w}_2(r) \\ . \\ . \\ . \\ \widehat{w}_n(r) \end{pmatrix}$$

is the estimation of the weight-vector in time-step r, comprising all weights of the neural network.

$$H(r) = \begin{pmatrix} \frac{\partial y_1(r)}{\partial w_1(r)} & \frac{\partial y_2(r)}{\partial w_1(r)} & ... & \frac{\partial y_m(r)}{\partial w_1(r)} \\ . & . & & . \\ . & . & & . \\ . & . & & . \\ \frac{\partial y_1(r)}{\partial w_n(r)} & \frac{\partial y_2(r)}{\partial w_n(r)} & ... & \frac{\partial y_m(r)}{\partial w_n(r)} \end{pmatrix}$$

depicts the Jacobi matrix, linearizing the nonlinear system around the point of estimate in time-step r.

$$K(r) = \begin{pmatrix} k_{1,1}(r) & k_{1,2}(r) & ... & k_{1,m}(r) \\ . & . & & . \\ . & . & & . \\ . & . & & . \\ k_{n,1}(r) & k_{n,2}(r) & ... & k_{n,m}(r) \end{pmatrix}$$

is the Kalman-gain matrix, used for updating the covariance-matrix $P(r)$.

$$P(r) = \begin{pmatrix} p_{1,1}(r) & p_{1,2}(r) & ... & p_{1,n}(r) \\ . & . & & . \\ . & . & & . \\ . & . & & . \\ p_{n,1}(r) & p_{n,2}(r) & ... & p_{n,n}(r) \end{pmatrix}$$

represents the covariance-matrix. The elements of $P(r)$ describe the dependencies of all weights with each other, based on previous estimation– and input-data.

$$Q(r) = \begin{pmatrix} q_{1,1}(r) & 0 & ... & 0 \\ 0 & q_{2,2}(r) & ... & \cdot \\ \cdot & \cdot & \cdot & 0 \\ 0 & ... & 0 & p_{n,n}(r) \end{pmatrix}$$

is the noise-matrix $Q(r)$, introducing artificial noise to prevent the estimation process from getting stuck in local minima. $q_{i,i} \in \left[10^{-6}, 10^{-2}\right], i = 1, ..., n$

$$S(r) = \begin{pmatrix} s_{1,1}(r) & s_{1,2}(r) & ... & s_{1,m}(r) \\ s_{1,2}(r) & s_{2,2}(r) & ... & s_{2,m}(r) \\ \cdot & \cdot & & \cdot \\ s_{1,m}(r) & s_{2,m}(r) & ... & s_{m,m}(r) \end{pmatrix}$$

is a user-defined, positive definite, symmetric matrix. $S(r)$ defines, in conjunction with $\eta(\cdot)$, the learning-rate.

3 Performance indexes for neural identification

3.1 Error-Functions

In order to supervise network-training, modifications of the standard error-functions are used as objective optimality-criterions. Since real-time algorithms are employed, appropriate error-functions are required. The error-functions, which are used throughout this work are the *Maximum Squared Error* (MSE) and the *Moving Average Squared Error* (MASE). These functions are defined as follows:

- *The Average Squared Error*

$$E_{ASE}(n) = \frac{1}{2} \sum_{k=1}^{M} (t_k(n) - o_k(n))^2 \tag{19}$$

  is the sum of the squared errors of all output neurons in time-step n.

- *The Maximum Squared Error*:

$$E_{MSE}(n) = \max_k \quad (t_k(n) - o_k(n))^2, \tag{20}$$

  is the maximum of the squared errors of all output neurons in time-step n with $k = 1, ..., M$.

- *The Moving Average Squared Error*

$$E_{MASE}(n) = \sum_{k=n-N+1}^{n} E_{ASE}(k) = \frac{1}{2} \sum_{k=n-N+1}^{n} \sum_{l=1}^{M} (t_l(k) - o_l(k))^2 \tag{21}$$

is the sum over all average squared errors $E_{ASE}(n)$ within the considered time-horizon, where

$M \in \mathbb{N}$ is the number of output-neurons and

$N \in \mathbb{N}$ the size of the time-horizon.

3.2 Residue

In control technology, in order to gain information about the performance of a controller, a number of caracteristic quantities are determined. The remaining, integral system-deviation, or *residue* [10], is a quantity, which can be transfered to the problem of system-identification with recurrent neural networks. It is defined in the following way:

The residue $E_{res}(n)$ of a control system with the desired output $t(n)$ and the actual output $o(n)$ is defined as:

$$E_{res}(n_0 + \tau) := \sum_{k=n_0}^{n_0+\tau} E(k), \quad with \quad E(k) := o(k) - t(k). \tag{22}$$

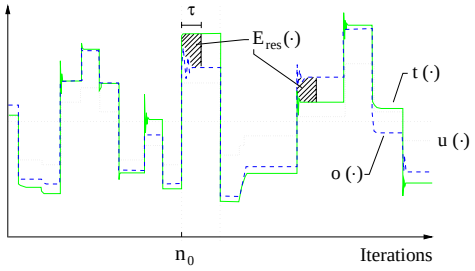

Figure 5: Residue

Figure 5 shows the response $t(n)$ and the approximation $o(n)$ of a system, being exposed to an input-signal of random steps. The residue $E_{res}(\cdot)$ is determined over the interval $[n_0, n_0 + \tau]$, where n_0 marks the beginning of an input-step. The constant τ depends on the settling-time of the neural network and specifies the number of time-steps to be considered for the residue.

3.3 Statistics

In order to obtain objective means for comparing the experimental results in Section 3, basic methods of statistics are chosen. Considering the residue defined above, its mean value μ can be calculated from the distribution of its relative frequency.

Let X be a discrete random variable and $\{x_0, x_1, ..., x_{m-1}\}, m \in \mathbb{N}$ an equidistant partition of the interval $[x_{min}, x_{max}]$. The distribution of the relative frequency is described by: $P(X = x)$, with $x := x_k$ if $x_k \leq E_{res}(n) < x_{k+1}$.

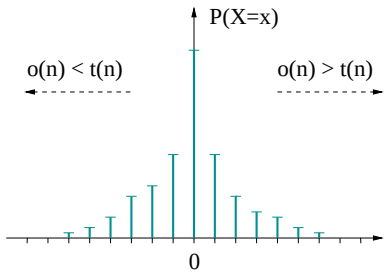

Figure 6: Distibution.

Figure 6(a) depicts the distribution of the relative frequency of the residue, which has been defined in Eqn. 22.

Let X be a discrete random variable and $x_1, x_2, ...$ the realizations of X. The expected value μ of the random variable X is defined as:

$$E[X] = \sum_i x_i P(X = x_i). \tag{23}$$

4 Identification of a nonlinear, dynamic process

4.1 Problem-Statement

The nonlinear, dynamic process [11], which is described by the second order state-space equations:

$$\begin{aligned} x_1(n+1) &= 1.145\ x_1(n) - 0.549\ x_2(n)\ + 0.584\ u(n), \\ x_2(n+1) &= \frac{x_1(n)}{1\ +\ 0.01\ (x_2(n))^2} + 0.330\ u(n), \\ y(n) &= 4\ tanh\left(\frac{x_1(n)}{4}\right), \end{aligned} \tag{24}$$

is considered for identification with a RMLP. In Eqn. 24, the variables $x_1(n)$ and $x_2(n)$ are the internal states in time-step n, $y(n)$ the observation or output and $u(n)$ the input to the process. The goal is to adjust the weights of a RMLP, according to Fig. 4, to obtain a neural network with the same temporal behavior.

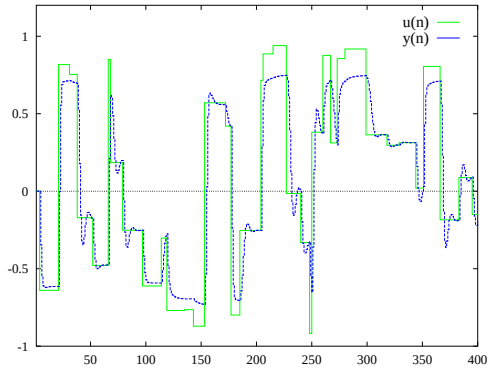

Figure 7: Process-behavior.

Figure 7 depicts the behavior of the exemplary process described by Equations 24. The dynamics of this process are oscillatory with unity static gain around zero. The denominator in the second state-equation, along with the $tanh(\cdot)$ in the output-equation, increases the damping and decreases the static gain in large amplitudes, cf. [11]. In this example, the input-sequence to the deterministic process consists of steps of random amplitude and a random duration of 1 to 20 time-steps. During training the neural network is provided with patterns, depicted in Fig. 7, including the input-signal $u(n)$ and the desired output $y(n)$. In time-step n, the identification-error $e_i(n)$ is determined from

the response $\hat{y}(n)$ of the network and the desired output $y(n)$. This error is minimized with a training-algorithm, by successively adjusting the weights of the recurrent neural network.

4.2 Net-Structure and Parameters

The Recurrent Multilayer Perceptron, depicted in Fig. 8, is employed for identification of the nonlinear, dynamic process described in Eqn. 24.

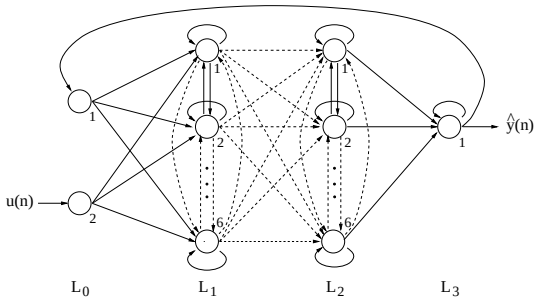

Figure 8: Identification-network

- *Network-Structure*

  The employed network consists of one input layer L_0 with a pseudo input-node, two hidden layers L_1, L_2 and an output-layer L_3. The output of the neural network is fed back, through an external recurrent weight-connection, into a pseudo input-node in layer L_0. According to Fig. 4, the neural network is provided only with the input-signal $u(n)$ and the desired output $y(n)$, which is used with the network-response $\hat{y}(n)$ for calculating the approximation error $e_i(n)$.

- *Learning Rate*

  The learning rate is set to $\eta = 0.04$.

- *Initialization of Weights*

  The values of the weight-connections are uniformly distributed over the interval $[-0.5, 0.5]$.

- *Training-Sequence*

  During training, the neural network is exposed to the sequence, consisting of 5000 training-patterns. The input-signal consists of steps of random amplitude from $[-1, 1]$ and random duration between 50 and 100 iterations.

- *Training-Algorithm*

  Identification is performed, using the Global Extended Kalman Filter with dynamic derivatives.

4.3 Results

In Figure 9, the output $\hat{y}(n)$ of the neural network, during the first epoch of training, is plotted along with the desired output $y(n)$, which illustrates the extremely rapid accommodation of the neural network.

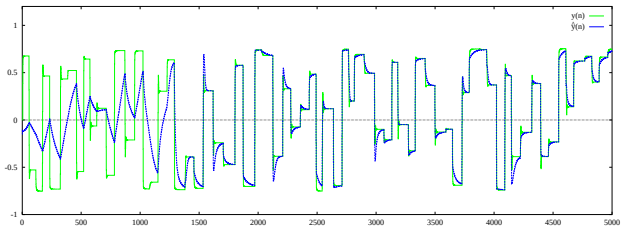

Figure 9: Behavior during training.

After approximately 1000 patterns (0.2 training-epochs!) have been presented to a randomly initialized network, its output $\hat{y}(n)$ begins follow the desired trajectory $y(n)$.

In Fig. 10, the Moving Average Squared Error and the Maximum Squared Error are plotted against the number of iterations and training-epochs. After 25 epochs of training, the network reaches a state of over-training and the MASE increases rapidly. The MSE of a small number of training-patterns remains almost zero, whereas the MSE for the majority of patterns diverges to large values, which can be observed in Fig.10.

Figure 11 and 12 depict the behavior of the neural network during recall, after 80000 iterations or 16 training-epochs have been performed. In Fig.11, the input-signal consists of steps with random amplitude and duration of 1 to 20 iterations. Although, the network

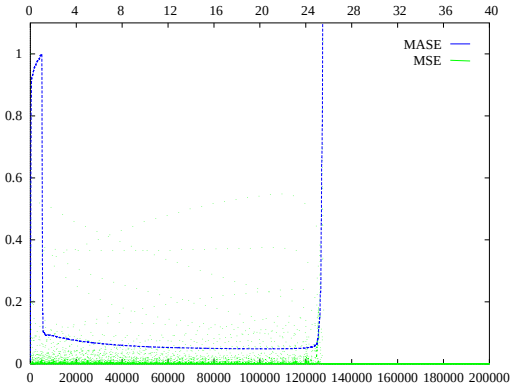

Figure 10: MASE and MSE during training.

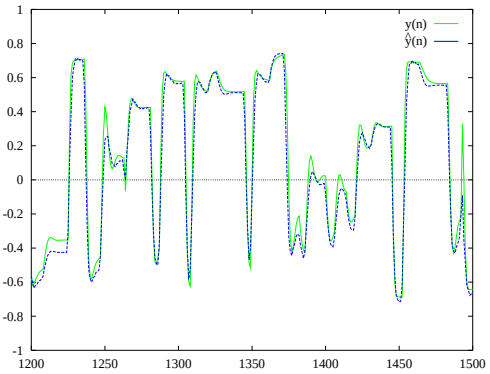

Figure 11: Recall after 16 training-epochs. Input-signal: steps 1 to 20 iterations.

did not explicitly learn how to respond to this kind of input-signal, its output follows the desired trajectory.

In turn, the neural network represents an identification of the original nonlinear, dynamic process. In Fig. 12, steps of random amplitude and duration of 50 to 100 iterations, are utilized as input-signal. In both cases, the input to the network, is not identical with the training-sequence.

The results, which have been presented in this section, show the principal applicability of recurrent neural networks for the identification of nonlinear, dynamic processes. In order to investigate effects of different parameters on the quality of identification, parameter-studies are carried out in the sequel.

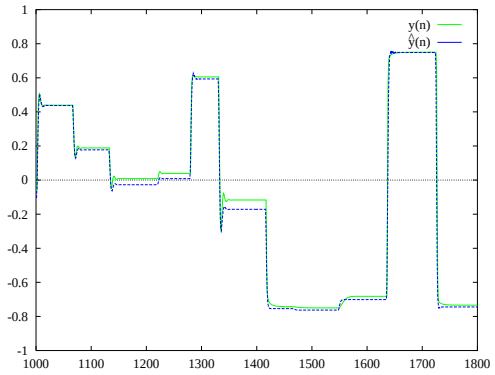

Figure 12: Recall after 16 training-epochs. Input-signal: steps 50 to 100 iterations.

4.4 Parameter Study

Utilization of neural structures requires various network-parameters to be chosen properly. Since no reliable methods for the determination of reasonable parameters from scratch is available, the effects of particular parameters are investigated in this section. In order to provide reasonable clues for successfull identification, the distributions of the *residues* during an actual recall phase (Section 3.2) are compared for various parameters.

Training-Algorithms

Based on the parameters in Sec. 4.2, the learning-algorithms Real Time Recurrent Learning (RTRL) and Global Extended Kalman Filter (GEKF) are employed for identification. RTRL is combined with *static*– and the GEKF with *dynamic* derivatives, yielding two training-methods to be investigated.

Figure 13 depicts the Moving Average Squared Error of both algorithms. The upper x-Axis displays the number of epochs, the lower x-Axis the number of performed iterations. Comparing both algorithms shows clearly better convergence results for the GEKF network. While the MASE of the network, trained with RTRL, reaches a stable value, the MASE of the network, trained with the GEKF, is conspicuous for its drop-off after approximately 125000 iterations. In the sequel, recalls will be performed with both networks, when training has been stopped around the drop-off point of the MASE.

In Fig. 14 and 15, the distributions of the residue, which are obtained during actual recall,

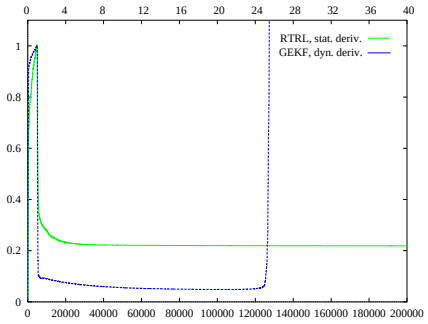

Figure 13: MASE during training.

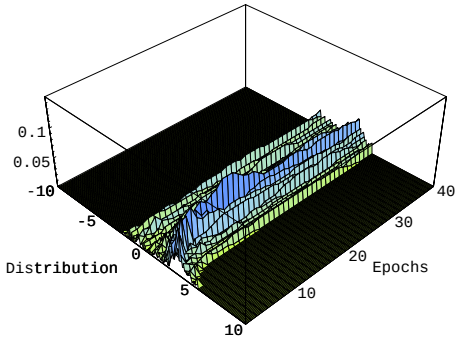

Figure 14: Distributions (RTRL, static deriv.).

are plotted against the number of training-epochs. As expected from the trajectory of the MASE for the GEKF in Fig. 13, the drop-off point, after 25 training-epochs, also appears in Fig. 15. Comparing the distributions of the residue of both algorithms, the employment of the GEKF-Algorithm shows a clearly better quality of identification. The GEKF-network produces smaller errors, which are distributed around zero more frequently, compared to the RTRL-trained network.

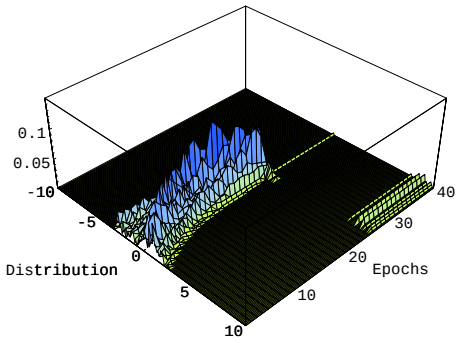

Figure 15: Distributions (EKF, dynamic deriv.).

Neurons

In this section, the effect of the number of hidden neurons on the quality of identification is investigated. Based on the network-structure introduced in Sec. 4.2, the hidden layers L_1 and L_2 consist of 2×2, 4×4 or 6×6 neurons.

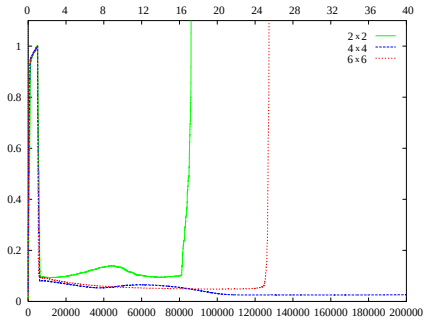

Figure 16: MASE during training.

In Fig. 16, the MASE is plotted against the number of training-epochs for different sizes of hidden layers. Comparing the MASEs of the 2×2 and 6×6 networks, with the MASE of the 4×4-network, no drop-off point is observed for the latter. This leads to the assumption of the existence of a constellation for the hidden layer, yielding stability during training.

Figure 17 shows a poor identification, performed by the according 2×2-network, since the distributions of the residue are flat and highly variant. However, if a 4×4-network is employed for identification, a stable distribution is reached after approximately 15

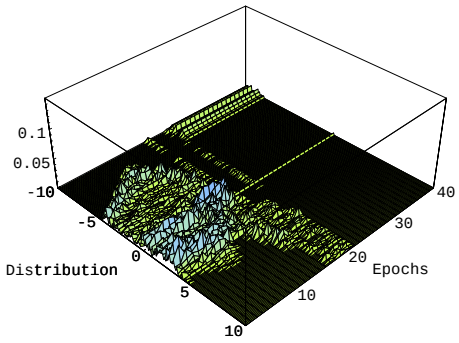

Figure 17: Distributions (2x2-network).

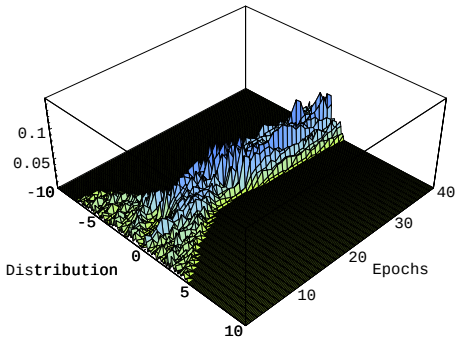

Figure 18: Distributions (4x4-network).

epochs, showing no further point of instability. The best quality of identification is obtained by a 6×6-network after 16 training-epochs, although the network reaches a state of instability after 25 epochs. The obtained results lead also to the assumption of an optimal structure for hidden layers, yielding stability during network-training. However, an optimal network-size depends on the accuracy-of-approximation versus computational-complexity trade-off and has to be chosen specifically for each problem. Furthermore, correlations with other parameters, eg. the learning-rate η, have to be considered also in the network-design.

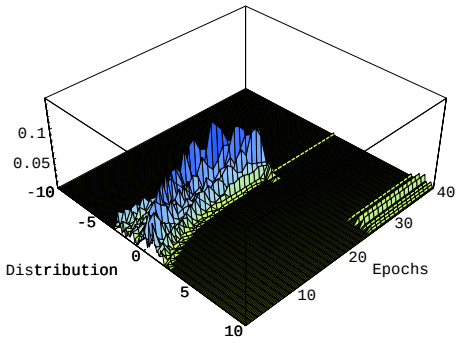

Figure 19: Distributions (6x6-network).

Layers

Refering to the parameters from Sec. 4.2, the influence of the number of hidden layers on the quality of identification is investigated for one– and two hidden layers. Identification is performed with a RMLP, comprising 6 neurons in its hidden layer(s).

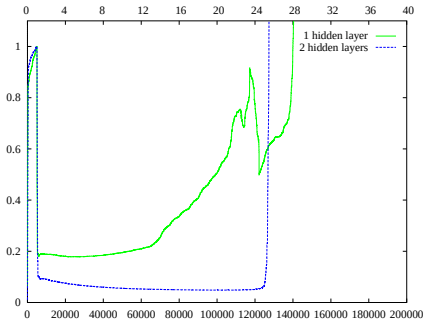

Figure 20: MASE during training.

In Fig. 20, the MASEs of both networks are depicted. After approximately 27 training-epochs, a drop-off point can be noticed, similar to the previous section. In Fig. 21 and 22, the distributions are presented, depicting the behavior of the considered networks during actual recalls. The neural network, incorporating only one hidden layer in its structure, yields a poor overall performance, compared to the two-hidden-layer network.

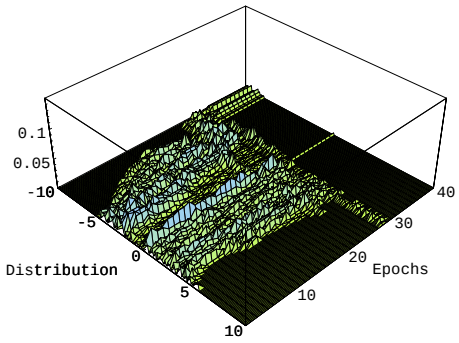

Figure 21: Distributions (1 hidden layer).

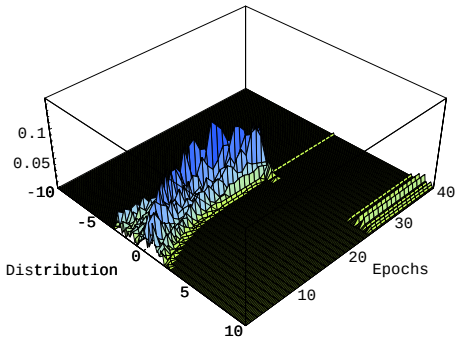

Figure 22: Distributions (2 hidden layer).

Learning Rate

In this section, the influence of the learning-rate on the performance of the resulting identification-network is investigated, based on the parameter-set from Sec. 4.2. Identification is performed with the learning-rate η set to 0.4, 0.04 and 0.004.

In Fig. 23, the according MASEs are plotted against the number of training-epochs. Figure 23 clearly depicts the effect of the learning-rate η on the convergence of the employed RMLP. Decreasing the learning-rate η postpones the drop-off point of the MASE and yields better identification results. The neural network, with $\eta = 0.004$, shows the best results and no point of instability is reached during training. As expected from the trajectory of the MASE, a learning-rate of 0.4 does not yield a network, successfully identifying the process-behavior. However, the best identification-network is achieved by employing a lerning-rate $\eta = 0.004$.

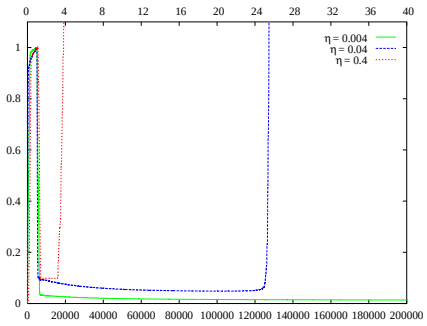

Figure 23: MASE during training.

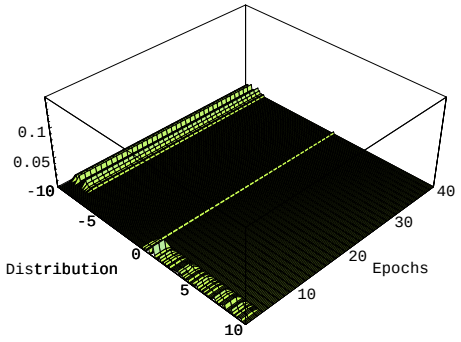

Figure 24: Distributions ($\eta = 0.4$).

The results of this section show a strong correlation between the learing-rate η and the stability during training. The drop-off point of the MASE is postponed or even completely suppressed, when applying small learning-rates. Refering to Fig. 26, one can also perceive η to have an essential influence on the quality of identification.

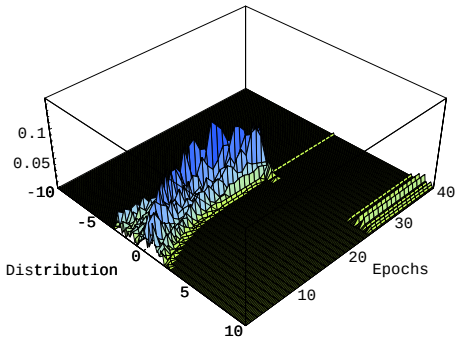

Figure 25: Distributions ($\eta = 0.04$).

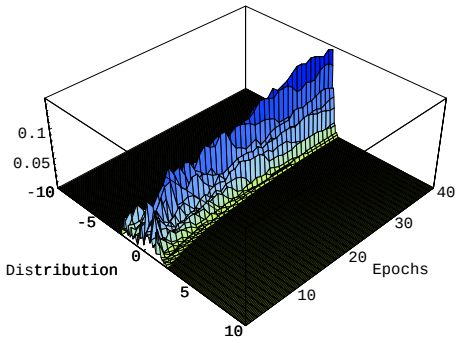

Figure 26: Distributions ($\eta = 0.004$).

5 Conclusion

The main objective of this work was the investigation of recurrent neural structures for identification of nonlinear, dynamic processes. This work was also intended to provide a general framework for a supplementary approach toward this problem, in order to overcome the complexity of analytical methods. Apart from conventional gradient–based algorithms for the training of neural networks, a new method has been considered with remarkable success — Global-Extended-Kalman-Filter training. During this work, a library has been implemented, which provides numerous functions for the efficient simulation and training of arbitrary recurrent neural structures.

In Section 4, a nonlinear, dynamic process could be successfully identified (Section 4) by employing a Recurrent Multilayer Perceptron and Global-Extended-Kalman-Filter training. Paramerter-studies have been carried out, to determine the influence of var-

ious parameters on the performance of resulting identification-networks. It turned out that recurrent neural structures are suitable for identifying nonlinear, dynamic processes. The obtained results showed that Global-Extended-Kalman-Filter training is a sophisticated algorithm, yielding neural networks with significantly better performance than conventional gradient-based algorithms.

The conclusion that can be drawn from the results of this work is that recurrent neural networks are capable to perform identification and control of nonlinear, dynamic processes. Furthermore, recurrent neural structures provide a general framework for the system engineer, which allows a process to be modeled and controlled without detailed knowledge about its intrinsic structure. However, employing recurrent neural network without prior feasibility-study, with respect to the process to beidentified, might yield unsatisfying results.

The application of recurrent neural networks should be understood as a supplement, not a substitute, of conventional mathematical systems theory. It turned out that recurrent structures are a promising approach toward the reduction of complexity during system modeling and controller-design.

References

[1] J. Raisch, *Mehrgrössenregelung im Frequenzbereich*. Oldenbourg–Verlag, 1994.

[2] O. Föllinger, *Nichtlineare Regelungen II*. Oldenbourg–Verlag, 1991.

[3] H. Nijmeijer and A. van der Schaft, *Nonlinear Dynamical Control Systems*. Springer–Verlag, 1990.

[4] M. C. Mozer, *Time Series Prediction: Forecating the Future and Understanding the Past*, pp. 243–264. Addison–Wesley, 1993.

[5] R. J. Williams and D. Zipser, "A learning algorithm for continually running fully recurrent neural networks," *Neural Computation*, vol. 1, pp. pp. 270–280, 1989.

[6] G. V. Puskorius and L. A. Feldkamp, "Neurocontrol of Nonlinear Dynamical Systems with Kalman Filter Trained Recurrent Networks," *IEEE Transactions on Neural Networks*, vol. Vol. 5, no. No. 2, pp. pp. 279–297, 1994.

[7] R. Kalman, "A New Approach to Linear Filtering and Prediction Problems," *J. Basic Eng., Trans. ASME*, vol. Vol. 82, no. No. 1, pp. pp. 35–45, 1960.

[8] C. K. Chui and G. Chen, *Kalman Filtering with Real-Time Applications*. Springer–Verlag, 1987.

[9] D. Catlin, *Estimation, control and the disrecte Kalman Filter*. Springer–Verlag, 1989.

[10] F. Gausch, A. Hofer, and K. Schlachen, *Digitale Regelkreise*. Oldenbourg–Verlag, 1993.

[11] I. Rivals, L. Personnaz, and G. Dreyfus, "Black–box Modelling with State–Space Neural Networks," tech. rep., NACT 1 Workshop on Neural Control, 1995.