Identification of Mechatronic Systems with Dynamic Neural Networks using Prior Knowledge

C. Endisch, M. Brache, P. Endisch, D. Schröder and R. Kennel *

Abstract—**This paper introduces a new approach for the identification of mechatronic systems with dynamic neural networks. Based on a given continuous signal flow chart a discrete chart is constructed. This discrete chart is implemented in a general dynamic neural network (GDNN). Certain weights of the neural network model correspond with physical parameters of the system and can be trained by an optimization algorithm. Moreover, nonlinear parts of the signal flow chart can be identified by nonlinear subparts of the neural network. The parameters are trained with the Levenberg-Marquardt (LM) optimization algorithm. Therefore the Jacobian matrix is required. The Jacobian is calculated by real time recurrent learning (RTRL). The proposed identification method is tested with a nonlinear two mass system.**

Keywords: System identification, mechatronic system, dynamic neural network, GDNN, real time recurrent learning

1 Introduction

Neural Network models are usually applied if no or very little prior knowledge of the system is available. The identification approach in [2][3] starts with an oversized general dynamic neural network (GDNN) and removes unnecessary parts (in GDNNs the layers can have feedback connections with time delays, see Fig. 1). During the identification process the network architecture is reduced to find a model for the plant as simple as possible. The approach in this paper is completely different. The assumption is that the structure of the system is known in form of a continuous signal flow chart. The model is constructed in two steps. First the continuous signal flow chart is redrawn in order to insert integrator blocks in all backward paths. In the second step the integrator blocks are discretized by the Implicit or the Explicit Euler approximation.

A similar approach is suggested in [6][7]. The so called structured recurrent neural network makes use of a state observer for system identification. Contrary to the structured recurrent neural network in [6][7] the approach in this paper does neither need a state observer nor system-dependent derivative calculations are necessary. The derivative calculations are conducted for the GDNN-model in general [10][9][11] no matter what model is used for the identification process.

The next section presents the recurrent neural network used in this paper. Administration matrices are introduced for implementing a special model structure. Section 3 deals with the parameter optimization method used throughout this paper. Section 4 describes the mechatronic system to be identified. In section 5 the structured GDNN is constructed. The identification process is presented in section 6. Finally, in section 7 we summarize the results.

2 General Dynamic Neural Network

De Jesus described the GDNN-model in his doctoral thesis [10]. The sophisticated formulations and notations of the GDNN-modell allow an efficient computation of the Jacobian matrix using real time recurrent learning (RTRL) [4][9][11][15]. Therefore we follow these conventions suggested by De Jesus. The simulation equation for layer m is calculated by

$$\underline{n}^m(t) = \sum_{l \in L_m^f} \sum_{d \in DL^{m,l}} \underbrace{LW}^{m,l}(d) \cdot \underline{a}^l(t-d) + \sum_{l \in I_m} \sum_{d \in DI^{m,l}} \underbrace{IW}^{m,l}(d) \cdot \underline{p}^l(t-d) + \underline{b}^m, \quad (1)$$

where $\underline{n}^m(t)$ is the summation output of layer m, $\underline{p}^l(t)$ is the l-th input to the network, $\underbrace{IW}^{m,l}$ is the input weight matrix between input l and layer m, $\underbrace{LW}^{m,l}$ is the layer weight matrix between layer l and layer m, $\underline{b}^m$ is the bias vector of layer m, $DL^{m,l}$ is the set of all delays in the tapped delay line between layer l and layer m, $DI^{m,l}$ is the set of all input delays in the tapped delay line between input l and layer m, I_m is the set of indices of input vectors that connect to layer m and L_m^f is the set of indices of layers that directly connect forward to layer m. The output of layer m is

$$\underline{a}^m(t) = \underline{f}^m(\underline{n}^m(t)), \quad (2)$$

*Technische Universität München, Institute for Electrical Drive Systems and Power Electronics, 80333 München, Germany, Email: christian.endisch@tum.de

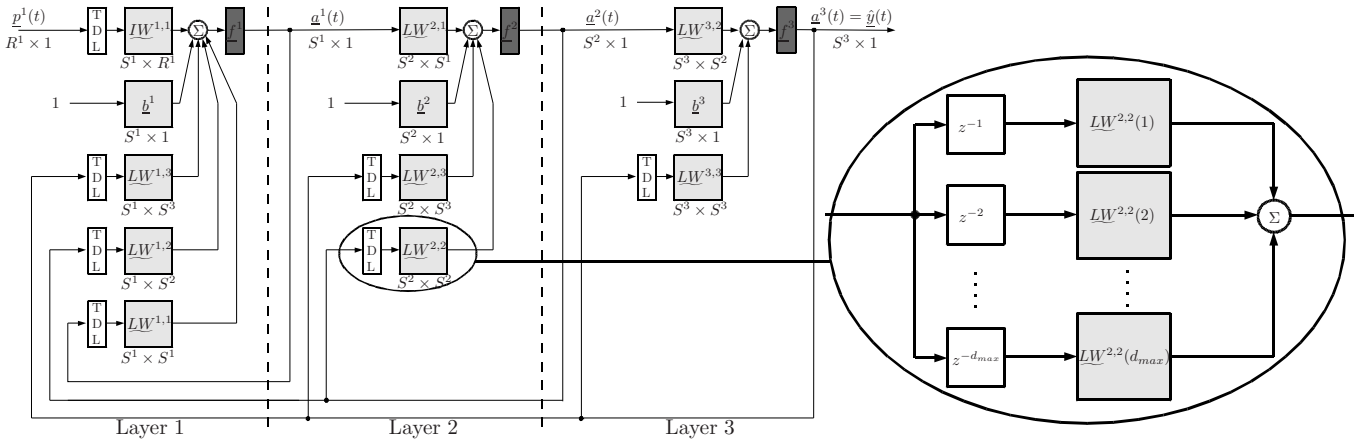

Figure 1: Example of a three-layer GDNN with feedback connections in all layers.

where $\underline{f}^m(\cdot)$ are either nonlinear *tanh*- or linear activation functions. At each point in time the equations (1) and (2) are iterated forward through the layers. Time is incremented from $t = 1$ to $t = Q$. (See [10] for a full description of the notation used here). Fig. 1 shows a tree-layer GDNN with feedback connections in all layers. In this example all feedback connections have complete tapped delay lines (from a first-order time delay element z^{-1} up to the maximum order time delay element $z^{-d_{max}}$). The output of a tapped delay line (TDL) is a vector containing delayed values of the TDL input. As seen in (1), also the network inputs have TDLs. In fig. 1 below the matrix-boxes and below the arrows the dimensions are shown. R^m and S^m respectively indicate the dimension of the input and the number of neurons in layer m. $\underline{\hat{y}}$ is the output of the GDNN.

In order to construct a flexible model-structure, it is necessary that only particular weights in the weight matrices do exist. This is realized by the introduction of administration matrices.

2.1 Administration Matrices

For each weight matrix there exists one weight administration matrix to mark which weights are used in the GDNN-model. The layer weight administration matrices $\underset{\sim}{\mathcal{AL}}^{m,l}(d)$ have the same dimensions as the layer weight matrices $\underset{\sim}{LW}^{m,l}(d)$, the input weight administration matrices $\underset{\sim}{\mathcal{AI}}^{m,l}(d)$ have the same dimensions as the input weight matrices $\underset{\sim}{IW}^{m,l}(d)$ and the bias weight administration vectors $\underline{\mathcal{Ab}}^m$ have the same dimensions as the bias weight vectors $\underline{b}^m$. The elements of the administration matrices can have the boolean values 0 or 1, indicating if a weight is valid or not. If e.g. the layer weight $lw^{m,l}_{k,i}(d) = [\underset{\sim}{LW}^{m,l}(d)]_{k,i}$ from neuron i of layer l to neuron k of layer m with a dth-order time-delay is valid, then $\left[\underset{\sim}{\mathcal{AL}}^{m,l}(d)\right]_{k,i} = \alpha l^{m,l}_{k,i}(d) = 1$. If the element in the administration matrice equals to zero, the corresponding weight has no influence on the GDNN. With these definitions the kth output of layer m can be computed by

$$\begin{aligned} n^m_k(t) = &\sum_{l\in L^f_m}\sum_{d\in DL^{m,l}}\left(\sum_{i=1}^{S^l} lw^{m,l}_{k,i}(d)\cdot\alpha l^{m,l}_{k,i}(d)\cdot a^l_i(t-d)\right)\\ &+\sum_{l\in I_m}\sum_{d\in DI^{m,l}}\left(\sum_{i=1}^{R^l} iw^{m,l}_{k,i}(d)\cdot\alpha i^{m,l}_{k,i}(d)\cdot p^l_i(t-d)\right)\\ &+b^m_k\cdot\alpha b^m_k,\\ a^m_k(t) = &f^m_k(n^m_k(t)), \end{aligned} \tag{3}$$

where S^l is the number of neurons in layer l and R^l is the dimension of the lth input. By setting certain entries of the administration matrices to one a certain GDNN-structure is generated. As this model uses structural knowledge from the system, it is called Structured Dynamic Neural Network (SDNN).

2.2 Implementation

For the simulations throughout this paper the graphical programming language *Simulink (Matlab)* is used. SDNN and the optimization algorithm are implemented as S-function in C++.

3 Parameter Optimization

First of all a quantitative measure of the network performance has to be defined. In the following we use the squared error

$$\begin{aligned} E(\underline{w}_k) &= \frac{1}{2}\cdot\sum_{q=1}^{Q}(\underline{y}_q-\underline{\hat{y}}_q(\underline{w}_k))^T\cdot(\underline{y}_q-\underline{\hat{y}}_q(\underline{w}_k))\\ &= \frac{1}{2}\cdot\sum_{q=1}^{Q}\underline{e}^T_q(\underline{w}_k)\cdot\underline{e}_q(\underline{w}_k), \end{aligned} \tag{4}$$

where q denotes one pattern in the training set, $\underline{y}_q$ and $\underline{\hat{y}}_q(\underline{w}_k)$ are the desired target and the actual model output of the q-th pattern respectively. The vector $\underline{w}_k$ is composed of all weights in the SDNN. The cost function $E(\underline{w}_k)$ is small if the training process performs well and large if it performs poorly. The cost function forms an error surface in a $(N+1)$-dimensional space, where N is equal to the number of weights in the SDNN. In the next step this space has to be searched in order to reduce the cost function.

3.1 Levenberg-Marquardt Algorithm

All Newton methods are based on the second-order Taylor series expansion about the old weight vector $\underline{w}_k$:

$$\begin{aligned} E(\underline{w}_{k+1}) &= E(\underline{w}_k + \Delta\underline{w}_k) \\ &= E(\underline{w}_k) + \underline{g}_k^T \cdot \Delta\underline{w}_k + \frac{1}{2} \cdot \Delta\underline{w}_k^T \cdot \underset{\sim}{H}_k \cdot \Delta\underline{w}_k. \end{aligned} \tag{5}$$

If a minimum on the error surface is found, the gradient of the expansion (5) with respect to $\Delta\underline{w}_k$ is zero:

$$\nabla E(\underline{w}_{k+1}) = \underline{g}_k + \underset{\sim}{H}_k \cdot \Delta\underline{w}_k = 0. \tag{6}$$

Solving (6) for $\Delta\underline{w}_k$ results in the Newton method

$$\begin{aligned} \Delta\underline{w}_k &= -\underset{\sim}{H}_k^{-1} \cdot \underline{g}_k^T, \\ \underline{w}_{k+1} &= \underline{w}_k - \underset{\sim}{H}_k^{-1} \cdot \underline{g}_k. \end{aligned} \tag{7}$$

The vector $-\underset{\sim}{H}_k^{-1} \cdot \underline{g}_k^T$ is known as the Newton direction, which is a descent direction, if the Hessian matrix $\underset{\sim}{H}_k$ is positive definite. The LM approach approximates the Hessian matrix by [5]

$$\underset{\sim}{H}_k \approx \underset{\sim}{J}^T(\underline{w}_k) \cdot \underset{\sim}{J}(\underline{w}_k) \tag{8}$$

and it can be shown that

$$\underline{g}_k = \underset{\sim}{J}^T(\underline{w}_k) \cdot \underline{e}(\underline{w}_k), \tag{9}$$

where $\underset{\sim}{J}(\underline{w}_k)$ is the Jacobian matrix

$$\underset{\sim}{J}(\underline{w}_k) = \begin{bmatrix} \frac{\partial e_1(\underline{w}_k)}{\partial w_1} & \frac{\partial e_1(\underline{w}_k)}{\partial w_2} & \cdots & \frac{\partial e_1(\underline{w}_k)}{\partial w_N} \\ \frac{\partial e_2(\underline{w}_k)}{\partial w_1} & \frac{\partial e_2(\underline{w}_k)}{\partial w_2} & \cdots & \frac{\partial e_2(\underline{w}_k)}{\partial w_N} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial e_Q(\underline{w}_k)}{\partial w_1} & \frac{\partial e_Q(\underline{w}_k)}{\partial w_2} & \cdots & \frac{\partial e_Q(\underline{w}_k)}{\partial w_N} \end{bmatrix} \tag{10}$$

which includes first derivatives only. N is the number of all weights in the neural network and Q is the number of evaluated time steps. With (7), (8) and (9) the LM method can be expressed with the scaling factor μ_k

$$\underline{w}_{k+1} = \underline{w}_k - \left[\underset{\sim}{J}^T(\underline{w}_k) \cdot \underset{\sim}{J}(\underline{w}_k) + \mu_k \cdot \underset{\sim}{I}\right]^{-1} \cdot \underset{\sim}{J}^T(\underline{w}_k) \cdot \underline{e}(\underline{w}_k), \tag{11}$$

where $\underset{\sim}{I}$ is the identity matrix. As the LM algorithm is the best optimization method for small and moderate networks (up to a few hundred weights), this algorithm is used for all simulations in this paper.

LM optimization is usually carried out offline. In this paper we use a sliding time window that includes the information of the last Q time steps. With the last Q errors the Jacobian matrix $\underset{\sim}{J}(\underline{w}_k)$ from equation (10) is calculated quasi-online. In every time step the oldest training pattern drops out of the time window and a new one (from the current time step) is added — just like a first in first out (FIFO) buffer. If the time window is large enough, it can be assumed that the information content of the training data is constant. With this simple method we are able to implement the LM algorithm quasi-online. For the simulations in this paper the window size is set to $Q = 25000$ using a sampling time of 1ms.

3.2 Jacobian Calculations

To create the Jacobian matrix, the derivatives of the errors have to be computed, see (10). The GDNN has feedback elements and internal delays, so that the Jacobian cannot be calculated by the standard backpropagation algorithm. There are two general approaches to calculate the Jacobian matrix for dynamic systems: By backpropagation through time (BPTT) [14] or by real time recurrent learning (RTRL) [15]. For Jacobian calculations the RTRL algorithm is more efficient than the BPTT algorithm [11]. According to this the RTRL algorithm is used in this paper. The interested reader is referred to [8]-[11],[2] for further details.

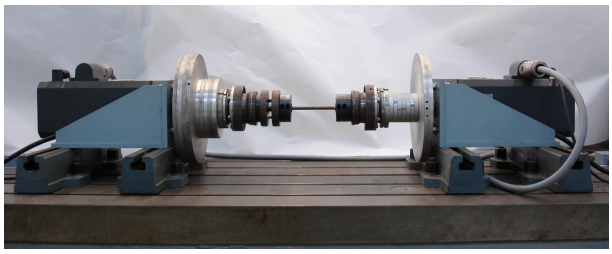

Figure 2: Laboratory setup of the TMS.

4 Two-Mass-System

The considered plant (shown in Fig. 2) is a nonlinear two-mass flexible servo system (TMS), which is a common example for an electrical drive connected to a work machine via flexible shaft. Fig. 3 displays the signal flow

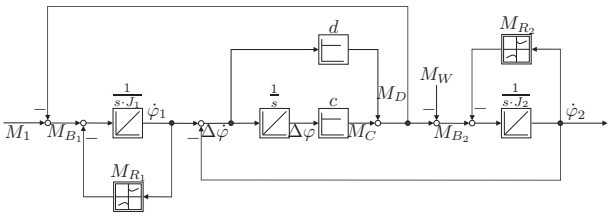

Figure 3: Signal flow chart of the TMS with friction.

chart of the TMS, where the spring constant c and the damping d model the shaft between the two machines [13]. $\dot{\varphi}_1$ and $\dot{\varphi}_2$ denote the rotation speed of the main engine and the work machine respectively. The torque of inertia of the machines are depicted by J_1 and J_2. The motor torque is M_1 and the torque at the work machine is M_2. M_{B1} and M_{B2} are the acceleration torques of the main engine and the work machine respectively. The difference of the rotation speeds is denoted by $\Delta\dot{\varphi}$ and $\Delta\varphi$ is the difference of the angles. The torque of the spring and the damping are M_C and M_D. The friction torques of the engine and the working machine are M_{R1} and M_{R2} respectively. The objective in this paper is to identify the linear TMS-parameters and the characteristics of the two friction torques.

5 Structured Dynamic Neural Networks

To construct a structured network with the help of GDNNs it is necessary to redraw the signal flow chart from fig. 3 because the implementation of algebraic loops is not feasible [1]. All feedback connections must contain at least one time delay, otherwise the signals cannot be propagated through the network correctly. This goal is accomplished by inserting integrator blocks in the feedback loops. Fig. 4 displays the redrawn version of fig. 3. By using the Euler approximation it is possible to dis-

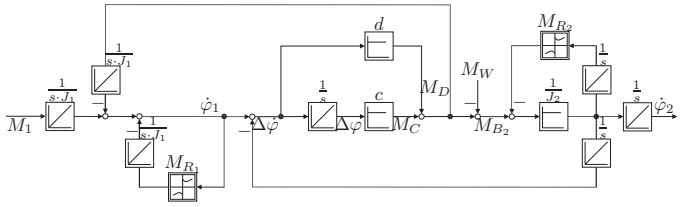

Figure 4: Redrawn signal flow chart of the TMS from fig. 3.

cretize the redrawn signal flow chart. The Implicit Euler approximation $y(t) = y(t-1) + x(t) \cdot T$ replaces all integrator blocks in the forward paths and the Explicit Euler approximation $y(t) = y(t-1) + x(t-1) \cdot T$ replaces all integrator blocks in the feedback paths. This approach ensures that all feedback connections contain the necessary time delays. The resulting discrete signal flow chart, which can be implemented as a SDNN, is displayed in fig. 5. z^{-1} denotes a first order time delay and T is the sampling time. All other denotations are the same as in fig. 3. In total the SDNN consists of 16 layers. The summing junctions depict the neurons of the network. Every summing junction is marked with a number, which denotes the layer of the neuron and its position within the layer. For instance, 15.1 marks the first neuron of the 15th layer. The friction of the engine and the work machine can be modeled by an optional number of neurons in the 5th layer and in the 11th layer respectively. These are the only neurons with tanh-transfer functions. All other neurons have linear transfer functions. The connections in fig. 5 which do neither belong to a linear parameter (depicted as box) nor to a friction-subpart are initialized with 1 or -1. The optimization algorithm is able to tune the parameters corresponding to the spring constant c, the damping d and the torque of inertia J_2 and the friction weights of the work machine. As it is not possible to identify the two torques of inertia as well as the two characteristic curves of the TMS simultaneously, the engine parameters are determined in a first no-load-identification which is conducted in idle running, see section 6.2.

6 Identification

6.1 Excitation Signal

The system is excited by an APRBS-signal (Amplitude Modulated Pseudo Random Binary Sequence [12]) combined with a bias produced by a relay. The APRBS-signal has an amplitude range between -7 and 7Nm and an amplitude interval between 10 and 250ms. The relay output switches to -4Nm if the rotation speed of the TMS is greater than $10\frac{\text{rad}}{\text{s}}$ and it switches to 4Nm if the rotation speed is smaller than $-10\frac{\text{rad}}{\text{s}}$. The suggested excitation signal ensures that the TMS, wich is globally integrating, remains in a well defined range for which the SDNN is able to learn the friction. Moreover, the output of the relay is multiplied by 0.2 for a rotation speed in the range of -2 to $2\frac{\text{rad}}{\text{s}}$. Thus, the SDNN receives more information about the friction in the region of the very important zero crossing. The resulting output of the TMS can be regarded in upper panel of fig. 8. This excitation signal is used in all the following identification processes.

6.2 Engine Parameters

For a successful TMS-identification it is necessary to identify the engine parameters in idle mode first. The obtained values are used as fix parameters in the identification of the whole TMS in chapter 6.3. The upper left drawing of fig. 6 displays the signal flow chart of the nonlinear engine, where M_1 is the torque, $\dot{\varphi}_1$ is the rotation speed, and J_1 denotes the torque of inertia. In order to be able to discretize the signal flow chart, we insert the integrator block in the backward path. The resulting signal flow chart is shown in the upper right drawing of fig. 6.

As explained above, for discretizing the signal flow chart the Implicit Euler approximation has to replace the integrator in the forward path, whereas the Explicit Euler approximation replaces the integrator in the backward path. The sampling time T is incorporated in the gain corresponding to the torque of inertia J_1. The lower drawing of fig. 6 displays the resulting discrete signal flow chart, which can be implemented as a SDNN in which all the summing junctions are regarded as neurons. The neurons

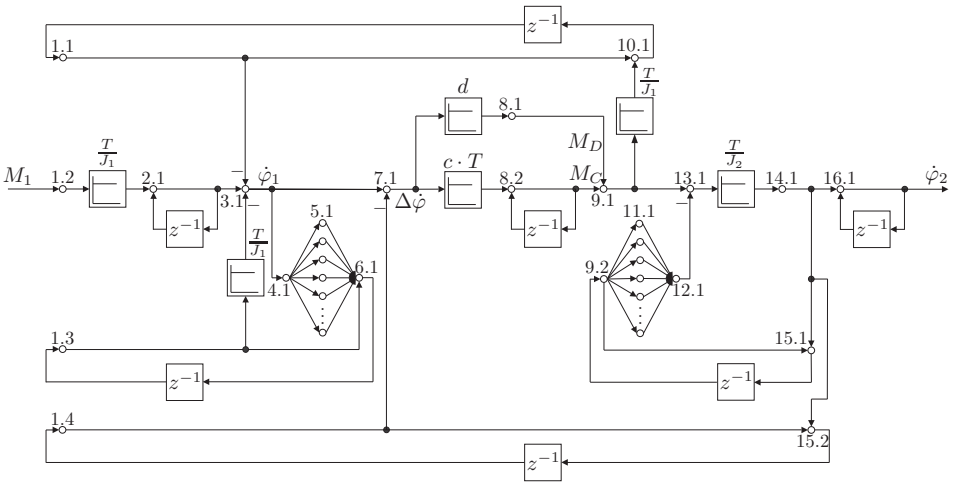

Figure 5: Structured recurrent network of a nonlinear TMS.

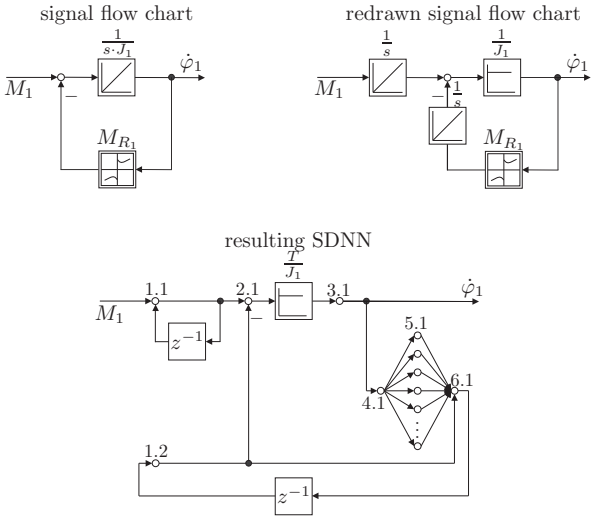

Figure 6: Signal flow chart of the engine (upper left side), redrawn signal flow chart of the engine (upper right side) and resulting SDNN (lower drawing).

in layer 5 model the friction of the engine.

For the identification process the engine is excited with the signal explained in section 6.1. The quasi-online calculated cost function $E(\underline{w}_k)$ (4) with $Q = 25000$ is minimized by the LM optimization algorithm (11). The sampling time is set to $T = 1$ms and the identification process starts after 5 seconds, seen in the upper panel of fig. 7. The SDNN-model of fig. 6 has three neurons in the fifth layer with six weights to model the friction. These weights are initialized randomly between -0.5 and 0.5. Table 1 shows two different initial values for the torque of inertia J_1 and the corresponding results. The calculated results are mean values between the last 25000 optimization steps. Fig. 7 displays the characteristic curve of the friction at $t = 100$ seconds identified by the SDNN. We observe that the network is able to model the jump due

Table 1: Initial values and final values of the torque of inertia of the engine J_1.

	J_1 [kgm^2]	E (mean value)
initial value	0.6	
result (mean value)	0.1912	$5.629 \cdot 10^{-2}$
initial value	0.01	
result (mean value)	0.1912	$4.387 \cdot 10^{-2}$

to the static friction with just three neurons. The following identification of the whole TMS uses this friction curve from fig. 7 and the result $J_1 = 0.1912$ from table 1 for the torque of inertia.

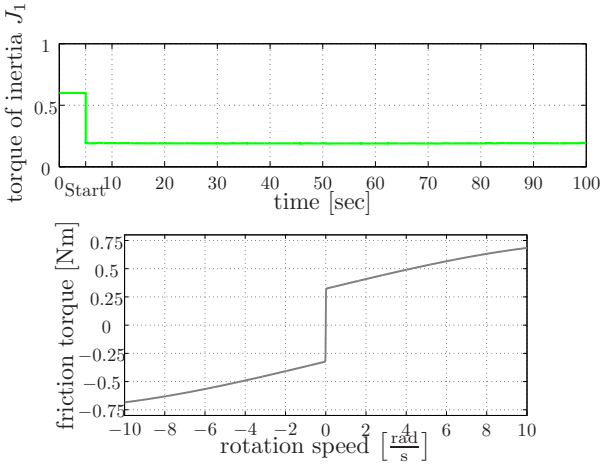

Figure 7: Identification of the engine parameters — Torque of inertia and friction curve identified by the nonlinear subpart in the 5th layer.

6.3 TMS Parameters

To identify the parameters of the whole TMS we excite the plant with the torque signal described in section 6.1 and use the SDNN-model constructed in chapter 5. The

Table 2: Initial values and final values for the identification of the TMS.

	J_2 [kgm^2]	d [$\frac{\text{Nms}}{\text{rad}}$]	c [$\frac{\text{Nm}}{\text{rad}}$]	Error (mean value)
initial value	0.7	0.4	100	
result (mean value)	0.3838	0.2406	477.2	$3.792 \cdot 10^{-2}$
initial value	0.1	0.7	800	
result (mean value)	0.3881	0.0919	477.6	$6.400 \cdot 10^{-2}$

torque of inertia of the engine J_1 and its friction are initialized according to the results of section 6.2. These weights remain unchanged during the whole identification process, whereas the weights corresponding to the torque of inertia of the work machine J_2, the spring constant c, the damping d and the work machine friction are trained. The work machine friction is modeled by three neurons with tanh functions in the 11th layer, see fig. 5. The six weights of this nonlinear subpart are initialized randomly between -0.5 and 0.5. The upper panel of fig.

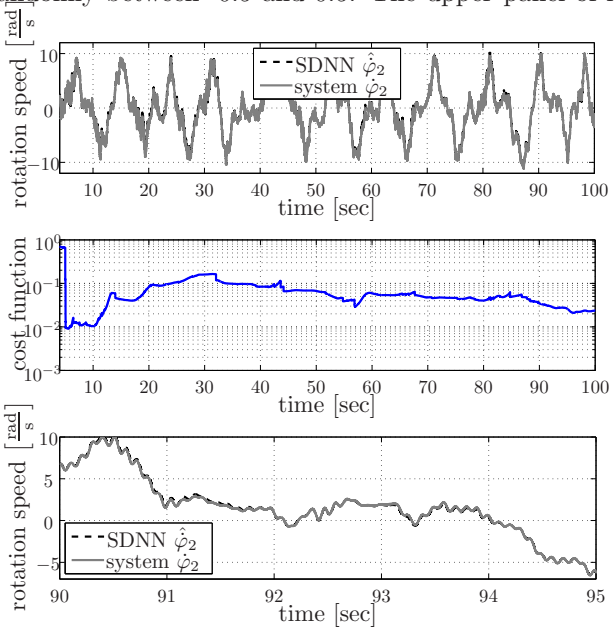

Figure 8: Output signals of the SDNN modell $\hat{\dot{\varphi}}_2$ and the real TMS $\dot{\varphi}_2$ with resulting cost function.

8 displays the outputs of the TMS and the SDNN-model during the identification process for the first set of initial values of table 2. The lower panel of this figure shows only five seconds for a detailed view. The identification process starts after 5 seconds. The sampling time is set to $T = 1$ms. The quasi-online calculated cost function $E(\underline{w}_k)$ (4) with $Q = 25000$ is minimized by the LM optimization algorithm (11) and is depicted in the middle panel of fig. 8. Due to the quasi-online approach the cost function value increases until $t = 25$ sec., until the training data window is completely filled up. The results in table 2 are mean values of the last 25000 optimization steps. Fig. 9 displays the developing of the damping, the spring constant and the torque of inertia during the identification process for the first initialization of table 2. Fig. 10 shows the characteristic curve of the work machine friction identified by the SDNN after 100 seconds. In addition to that table 2 shows the results of a

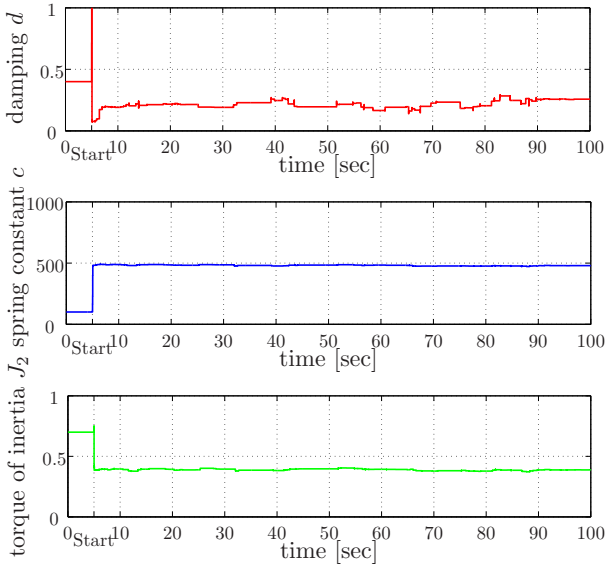

Figure 9: Linear parameter signals during the identification.

second identification run with another initialization. The resulting torque of inertia and spring constant are almost equal. Only the damping shows different final values. The higher mean error (compared to the first identification) implies that the second optimization process ended in a local minimum.

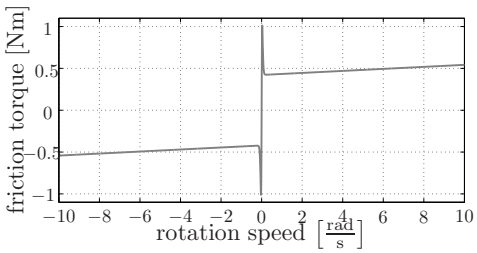

Figure 10: Friction curve identified by the nonlinear subpart in the 11th layer.

7 Conclusion

This paper introduces a new approach for system identification using prior knowledge. Starting with a given continuous signal flow chart a discrete SDNN-model is constructed in two steps. First the continuous signal flow chart is redrawn in order to insert integrator blocks in the feedback loops. In the second step the redrawn chart is discretized by replacing all integrator blocks in the forward paths by the Implicit Euler approximation and all integrator blocks in the feedback paths by the

Explicit Euler approximation. The resulting chart contains time delays in all feedback connections and can be implemented as a SDNN without algebraic loops. Certain weights of the SDNN correspond to physical parameters, and nonlinear subparts of the SDNN model nonlinearities of the system. The suggested identification approach is tested with a nonlinear TMS. The results verify that the suggested approach enables us to identify the torques of inertia, the spring constant, the damping and the two friction curves of the TMS. Problems may occur if the optimization algorithm gets stuck in a local minimum. In this case the identification process has to be restarted with a different set of initial values. With the help of the mean error the success of the identification process can be validated.

References

[1] M.J. Brache, "Identification of Dynamic Systems Using Previous Knowledge," Diploma Theses, Lehrstuhl für Elektrische Antriebssysteme, Technische Universität München, 2008.

[2] C. Endisch, C. Hackl and D. Schröder, "Optimal Brain Surgeon For General Dynamic Neural Networks," in Lecture Notes in Artificial Intelligence (LNAI) 4874, Springer-Verlag Berlin, pp. 15-28, 2007.

[3] C. Endisch, C. Hackl and D. Schröder, "System Identification with General Dynamic Neural Networks and Network Pruning," International Journal of Computational Intelligence, vol. 4, no. 3, pp. 187–195, 2008.

[4] C. Endisch, P. Stolze, C. Hackl and D. Schröder, "Comments on Backpropagation Algorithms for a Broad Class of Dynamic Networks," IEEE Trans. on Neural Networks, vol. 20, no. 3, pp. 540–541, 2009.

[5] M. Hagan, B. M. Mohammed, "Training Feedforward Networks with the Marquardt Algorithm," IEEE Trans. on Neural Networks, vol. 5, no. 6, pp. 989–993, 1994.

[6] C. Hintz, B. Angerer and D. Schröder, "Online Identification of Mechatronic System with Structured Recurrent Neural Networks," Proceedings of the IEEE-ISIE 2002, L'Aquila, Italy, pp. 288–293, 2002.

[7] C. Hintz, "Identifikation nichtlinearer mechatronischer Systeme mit strukturierten rekurrenten Netzen," Dissertation, Lehrstuhl für Elektrische Antriebssysteme, Technische Universität München, 2003.

[8] O. De Jesús, M. Hagan, "Backpropagation Algorithms Through Time for a General Class of Recurrent Network," IEEE Int. Joint Conf. Neural Network, Washington, pp. 2638–2643, 2001.

[9] O. De Jesús, M. Hagan, "Forward Perturbation Algorithm For a General Class of Recurrent Network," IEEE Int. Joint Conf. Neural Network, Washington, pp. 2626–2631, 2001.

[10] O. De Jesús, "Training General Dynamic Neural Networks," Ph.D. dissertation, Oklahoma State University, Stillwater, OK, 2002.

[11] O. De Jesús, M. Hagan, "Backpropagation Algorithms for a Broad Class of Dynamic Networks," IEEE Trans. on Neural Networks, vol. 18, no. 1, pp. 14–27, 2007.

[12] O. Nelles, Nonlinear System Identification. Berlin Heidelberg New York: Springer-Verlag, 2001.

[13] D. Schröder, Elektrische Antriebe - Regelung von Antriebssystemen. 2nd edn., Berlin Heidelberg New York: Springer-Verlag, 2001.

[14] P.J. Werbos, "Backpropagation Through Time: What it is and how to do it," Proc. IEEE, vol. 78, no. 10, pp. 1550-1560, 1990.

[15] R.J. Williams, D. Zipser, "A Learning Algorithm for Continually Running Fully Recurrent Neural Networks," Neural Computing, vol. 1, pp. 270–280, 1989.