NONLINEAR CONTROL SYSTEMS MODELLING USING LOCAL LINEAR MAPPINGS [1]

Andrzej Dzieliński, [*] **Waldemar Graniszewski** [*]

[*] *Institute of Control and Industrial Electronics, Warsaw University of Technology, Koszykowa 75, 00-662 Warsaw, Poland*

Abstract: In this paper new nonlinear control systems modelling methodology is presented. The nonlinear system is given in the form of NARX model. Modelling method is based on decomposition of the nonlinear function (right-hand side of the NARX model) into multiple local linear mappings. These mappings are represented by a self-organising neural network and a local linear mapping is associated with each neuron. The network is trained by a combination of vector quantisation and Winner Takes Most procedure. The application of the proposed method to modelling of nonlinear control system is also given.

Keywords: Nonlinear systems, System modelling, Local Linear Mappings, Neural Networks

1. INTRODUCTION

Modelling and control of nonlinear dynamical systems is one of the most important but also the most challenging areas of system theory. There has been a great deal of research activity in this area, mainly focused on approaches like neural networks and fuzzy logic. However, much of this work on identifying nonlinear input-output model produced methods with little or no insight into the underlying data generation process. On the other hand, there has been a great progress in the basic research into, so called local approaches to modelling. These included classical control approaches (Taylor linearisation), statistical methods, neuro- and fuzzy architectures (B-splines and Radial Basis Function networks) etc. (Johansen and Murray-Smith, 1997).

In this paper we consider a deterministic, non-linear, single-input single-output (SISO) system given by the discrete-time input-output Nonlinear AutoRegressive with eXogenous input (NARX) model

$$\begin{aligned} y(k+1) = f(y(k), \ldots, y(k-n+1), \\ u(k), \ldots, u(k-m+1)) \end{aligned} \tag{1}$$

with $y \in [a, b] \subset \mathbb{R}$, $u \in [c, d] \subset \mathbb{R}$ and $f \colon D \to [a, b]$ with the domain of definition $D = [a, b]^n \times [c, d]^m$. It is physically natural that y and u assume only finite values on a connected set and can attain their bounds. In fact D is a compact, connected and convex subset of $\mathbb{R}^{n+m}$.

In principle the method should be valid for MIMO systems. However due to more complicated visualisation, in this paper we proceed with the SISO case. The problem of MIMO system identification is the subject of work in progress.

Recall (Lakshmikantham and Trigiante, 1988) that the solution of (1) for given initial conditions exists and is unique for *any* function f, as it is built by straightforward iterations. This is in marking contrast to the non-trivial character of the initial value problem for ordinary differential equations (ODEs). There continuity is needed for existence and Lipschitz continuity for uniqueness. For ordinary difference equations (OΔEs) f maybe even discontinuous, while discontinuity complicates things enormously for ODEs. The main reason is that the left-hand side (LHS) of an ODE stands for a limiting process ($\dot{x}$), which may fail to be well-defined for certain f, while the LHS of an OΔE is obtained by an algebraic process of evaluating f.

[1] This work was partially supported by the Warsaw University of Technology grant

It should be mentioned that model (1) is usually obtained by discretisation (Kalkkuhl and Hunt, 1996) of a deterministic non-linear (Lipschitz) continuous-time SISO control system

$$\begin{aligned} \dot{x} &= f_1(x, u) \\ y &= h(x) \end{aligned} \qquad (2)$$

with $x \in X \subset \mathbb{R}^{n_1}$ and $y \in [a, b] \subset \mathbb{R}$, $u \in [c, d] \subset \mathbb{R}$ and an initial condition $x(t_0) = x_0$. This, in general, is an approximation process (Normand-Cyrot, 1987), as the expression for the input-output map $\lambda \colon X \times [c, d] \times \mathbb{R} \to [a, b]$, yielding $y(t) = \lambda(x_0, u(t), t)$, is unknown. Thus the fact that discretisation with the sampling period T assumes

$$\forall\ k \in \mathbb{Z}\ \ \forall\ t \in [kT, (k+1)T) \quad u(t) = \text{const}$$

cannot be directly used in deriving f from λ. The input-output model (1), derived from the discrete-time state-space description (Leontaritis and Billings, 1985; Chen and Billings, 1989), is valid only locally. Therefore it is not the ‘ultimate black-box’ and we cannot expect any mathematically nice properties of f, even if the underlying continuous-time model has them.

The method proposed in the paper has the advantage of not using any *a’priori* knowledge about the plant. We use self-organising structure that can match the data distribution and according to this distribution can produce a local linear mapping.

2. MODELLING METHODOLOGY

2.1 *Function approximation using local linear mappings*

In this section we would like to introduce an approach for function approximation using artificial neural network with local linear mapping. This problem was attacked from various angles in the field of neural networks: (Ritter, 1991; Martinetz *et al.*, 1993; Nelles, 1996). From mathematical point of view our model can be described as follows. Let $\mathcal{V}$ be a subset of $\mathbb{R}^n$, $\mathcal{V} \subseteq \mathbb{R}^n$ and $f : \mathcal{V} \to \mathbb{R}$ is a function, such that for $\boldsymbol{v} \in \mathcal{V}$: $\boldsymbol{v} = (v_1, \dots, v_n)$ value of $f(\boldsymbol{v}) = f(v_1, \dots v_n)$. The goal is to approximate this function through a sum of K weighted linear functions $\hat{f}_i(\boldsymbol{v})$, i.e.

$$\tilde{f}(\boldsymbol{v}) = \sum_{i=1}^{K} (\hat{f}_i(\boldsymbol{v}) \cdot \Phi_i(\boldsymbol{v})) \qquad (3)$$

and $\Phi_i(\boldsymbol{v})$ is a weighting function. Let us divide subset $\mathcal{V}$ into k sub-subsets that satisfy condition: $\mathcal{V} = \mathcal{V}_1 \cup \mathcal{V}_2 \cup \dots \cup \mathcal{V}_K$. Let the mapping $F \colon \mathcal{V} \to \mathbb{R}^K$ be defined as a collection $(\hat{f}_1, \dots, \hat{f}_K)$ of K - functions $\hat{f}_i \colon \mathcal{V}_i \to \mathbb{R}$. We approximate, locally, function f in each sub-subset $\mathcal{V}_i$, $i = 1, \dots, K$ through a function $\hat{f}_i$ given by the equation

$$\hat{f}_i(\boldsymbol{v}) = \boldsymbol{a}_i \cdot \boldsymbol{v} + b_i, \qquad (4)$$

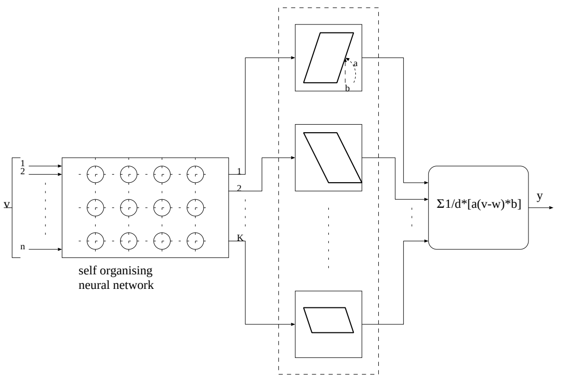

Fig. 1. Architecture of neural network for local linear approximation.

where $\boldsymbol{a}_i$ denotes a slope of the hyperplane and b_i is an offset. For $\boldsymbol{v} \in \mathcal{V}_i$ function $\hat{f}_i$ is an n - dimensional hyperplane, i.e., if $\dim \mathcal{V} = 1$, then $\hat{f}_i$ is a line, if $\dim \mathcal{V} = 2$, then $\hat{f}_i$ is a plane etc.

2.2 *Neural network architecture*

Neural network with local linear mapping for function approximation is a hybrid network consisting of K-neurons. On the block-diagram in Figure 1 one may see that in the first phase neurons organise themself and build a self organising structure (Kohonen, 1995). In the second phase, each neuron from the structure described above, is associated with a local linear function. Three parameters are associated with each neuron:

- $\boldsymbol{w}_i$ - neuron’s weight that describe position of the neuron in $\mathcal{V} \subseteq \mathbb{R}^n$
- n - dimensional hyperplane associated with each neuron which is described through:
  - · $\boldsymbol{a}_i$ - a slope of the hyperplane
  - · b_i - a offset of the hyperplane

Neurons are changing their positions i.e., their weights, according to input vectors $\boldsymbol{v}$. This causes that the input data set $\mathcal{V} \subseteq \mathbb{R}^n$ will be mapped into limit set of the K neurons weights $\mathcal{W}$. This can be denoted as $W : \mathcal{V} \to \mathcal{W}$ where $\mathcal{W} = \{\boldsymbol{w}_1, \boldsymbol{w}_2, \dots, \boldsymbol{w}_K\}$ and $\boldsymbol{w}_i \in \mathbb{R}^n$ for $i = 1, \dots, K$. Each input vector $\boldsymbol{v}$ is mapped into one of the neurons weights $\boldsymbol{w}_i$. Mapping criterion, is that Euclidean distance between input vector $\boldsymbol{v}$, and neuron’s weight $\boldsymbol{w}_i$, defined as $d_i(\boldsymbol{v}, \boldsymbol{w}_i) = \|\boldsymbol{v} - \boldsymbol{w}_i\|$, is smaller then the distance $d_j(\boldsymbol{v}, \boldsymbol{w}_j) = \|\boldsymbol{v} - \boldsymbol{w}_j\|$, between input vector and other neurons weights, i.e., $d_i(\boldsymbol{v}, \boldsymbol{w}_i) \leq d_j(\boldsymbol{v}, \boldsymbol{w}_j) \quad | \quad \forall j = 1 \dots K$. The neuron closest to the given vector is called a ”winner neuron”. Using such mapping some input data $\boldsymbol{v} \in V$ will be mapped into the same neuron $\boldsymbol{w}_i$. Input data space will be divided into K subspaces, i.e. $\mathcal{V} = \mathcal{V}_1 \cup \mathcal{V}_2 \cup \dots \cup \mathcal{V}_K$, and each subspace will be mapped through the weight (position) of the corresponding neuron $\mathcal{V}_i \to \boldsymbol{w}_i$. This technique is called vector quantisation (Gray, 1984).

If we have distributed neurons using vector quantisation technique we could associate a linear function with each neuron. Formally we describe this below. Let subset $Y \subset \mathbb{R}$ be an image of the subset $\mathcal{V} \subseteq \mathbb{R}^n$. Let element $y \in Y$ satisfy the equation $y = f(\boldsymbol{v})$. If we map subspace $\mathcal{V}_i$ into weights of i-th neuron i.e., $\mathcal{V}_i \to \boldsymbol{w}_i$, then we can also map elements of the image Y into $\tilde{y}$ according to equation

$$\tilde{y_i} = \boldsymbol{a}_i \cdot (\boldsymbol{v} - \boldsymbol{w}_i) + b_i \quad , \tag{5}$$

where: $\tilde{y_i}$ is an approximated value of the function $y = f(\boldsymbol{v})$ for $\boldsymbol{v} \in \mathcal{V}_i$. The triplet $\boldsymbol{w}_i$,$\boldsymbol{a}_i$ and b_i, denote winner neuron's parameters: its weight satisfy condition: $d_i(\boldsymbol{v}, \boldsymbol{w}_i) = ||\boldsymbol{v} - \boldsymbol{w}_i|| = \min d_j(\boldsymbol{v}, \boldsymbol{w}_j)$ for $j = 1, \ldots, K$.

2.3 *Learning procedure*

During learning process input-output data pairs $(\boldsymbol{v}, y)$ are introduced to the network, where $\boldsymbol{v}$ is a learning vector and y is an expected output. The goal of the learning process is to minimise the error between network output and expected output

$$e = y - \tilde{y}. \tag{6}$$

We minimise this error by adjusting neuron's parameters i.e. $\boldsymbol{w}_i$, $\boldsymbol{a}_i$ and b_i. There are many algorithms for changing the neuron's weights for self organising structure, e.g. K-means algorithm (Linde *et al.*, 1980), Kohonen's algorithm (Kohonen, 1995), Conscience Winner Takes All algorithm (DeSieno, 1988). In adaptation of neurons weights we used winner takes most - WTM learning rule also called "neural gas" algorithm introduced by Martinetz and Schulten (Martinetz and Schulten, 1991), because of a good convergence properties described by the Authors. The new neuron position is calculated from the equation

$$\Delta \boldsymbol{w}_i = \epsilon \cdot h_\lambda(k(\boldsymbol{v}, \boldsymbol{w})) \cdot (\boldsymbol{v} - \boldsymbol{w}_i) \quad i = 1, \ldots, K, \tag{7}$$

where: $\epsilon \in [0, 1]$ is learning rate parameter and h_λ is a parameter that depends on neighbourhood ranking.

As we have described it earlier, with each neuron there is associated a hyperplane, which locally maps an input of the network to network's output and describes a neuron $\boldsymbol{w}_i$ response for a given input vector $\boldsymbol{v}$.

Adjustment of the slope $\boldsymbol{a}_i$ and offset b_i, is performed according to:

$$\Delta \boldsymbol{a}_i = \epsilon' \cdot (y - b_i - \boldsymbol{a}_i \cdot (\boldsymbol{v} - \boldsymbol{w}_i)) \cdot (\boldsymbol{v} - \boldsymbol{w}_i)^T \tag{8}$$

$$\Delta b_i = \epsilon'' \cdot (y - b_i) \quad i = 1, \ldots, K. \tag{9}$$

where: ϵ' i ϵ'' are learning rates $[0, 1]$. After learning process we receive a set of K - neurons described by 3 parameters: $\{\boldsymbol{w}_i, \boldsymbol{a}_i, b_i\}$.

2.4 *Network validation test*

Once trained, a network can be tested by introduction of the test vector $\boldsymbol{v}_{test}$. Trained network finds the neuron which weight is the closest to given vector, i.e. $d_i(\boldsymbol{v}_{test}, \boldsymbol{w}_i) = \min ||\boldsymbol{v}_{test} - \boldsymbol{w}_i||$, and for the winner neuron a response according to equation (10) is calculated

$$\tilde{y_i} = \boldsymbol{a}_i \cdot (\boldsymbol{v}_{test} - \boldsymbol{w}_i) + b_i. \tag{10}$$

Calculating response from the equation (10) can cause discontinuity of approximation on the border of influence of neighbouring neurons. For this reason we calculate the output of the network considering the outputs of all neurons according to their distance to $\boldsymbol{v}_{test}$ from the equation

$$\tilde{y} = \frac{\sum_{i=1}^{K} \frac{1}{d_i^m} \boldsymbol{a}_i \cdot (\boldsymbol{v}_{test} - \boldsymbol{w}_i) + b_i}{\sum_{i=1}^{K} \frac{1}{d_i^m}}, \tag{11}$$

where: $d_i = ||\boldsymbol{v}_{test} - \boldsymbol{w}_i||$ is Euclidean distance between test vector and i-th neuron's weights, m - arbitrary integer.

3. APPLICATION

Neural network for nonlinear function's approximation using local linear model has been implemented in software written in MATLAB. In addition, for testing the dynamic characteristics of neural network model, SIMULINK Toolbox for MATLAB [2] has been also used. We have applied neural network described in previous sections for modeling a car's speed control system with following data set values:

- $u(k)$ - control signal at time k - throttle angle at time k
- $y(k)$ - output signal at time k - car's speed at time k
- $y(k+1)$ - output signal at time $k+1$ - car's speed at time $k+1$

Behavior of the model can be described by the equation

$$y(k+1) = f(u(k), y(k)). \tag{12}$$

In this case dimension of the input data set is 2, i.e. $\mathcal{V} \subseteq \mathbb{R}^2$ and $\boldsymbol{v} = (u(k), y(k))^T$. We have used two data sets:

(1) 733 raw data
(2) 294 pre-processed data.

Distribution of raw and pre-processed data set is not uniform as is shown in Figure 2. Values of throttle angle $u(k) \in [0.05, 0.95]$. Before the data sets are introduced to the neural network controller, they were normalised. Next the data pairs $(\boldsymbol{v}, y)$, in the form of the input vector $\boldsymbol{v} = [u(k), y(k)]^T$ and expected output $y = [y(k+1)]$, are presented to the network. The process is repeated until neural network responds with expected value, i.e., output error is less than the pre-assigned error. Experiments were performed for

[2] MATLAB and SIMULINK are registered trademarks of The MathWorks, Inc.

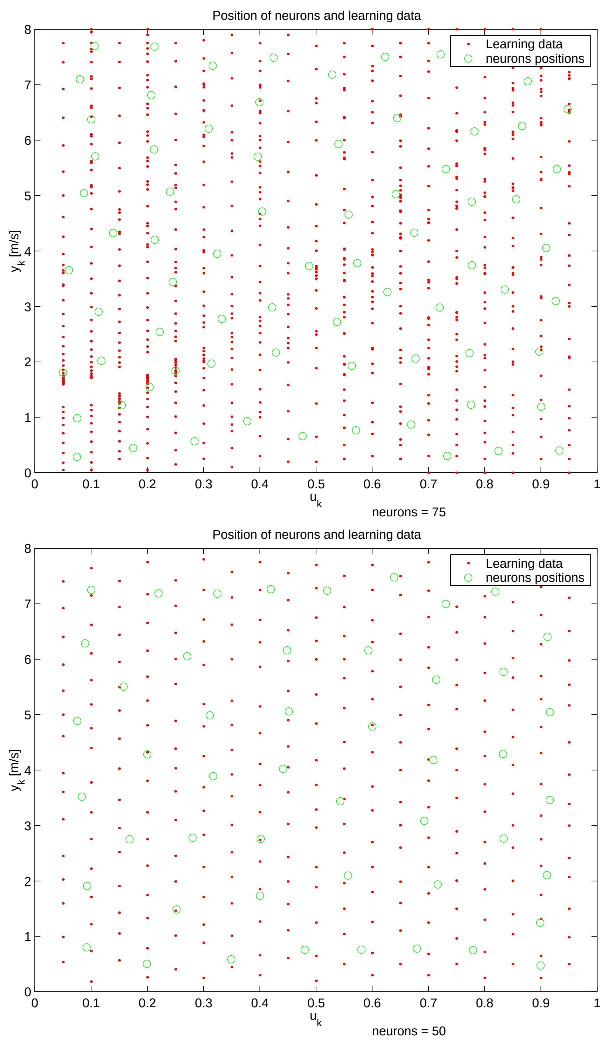

Fig. 2. Distribution of data set and neurons positions shown in projection into input's data space . Network with 75 neurons learned with raw data (top). Network with 50 neurons learned with pre-processed data (bottom).

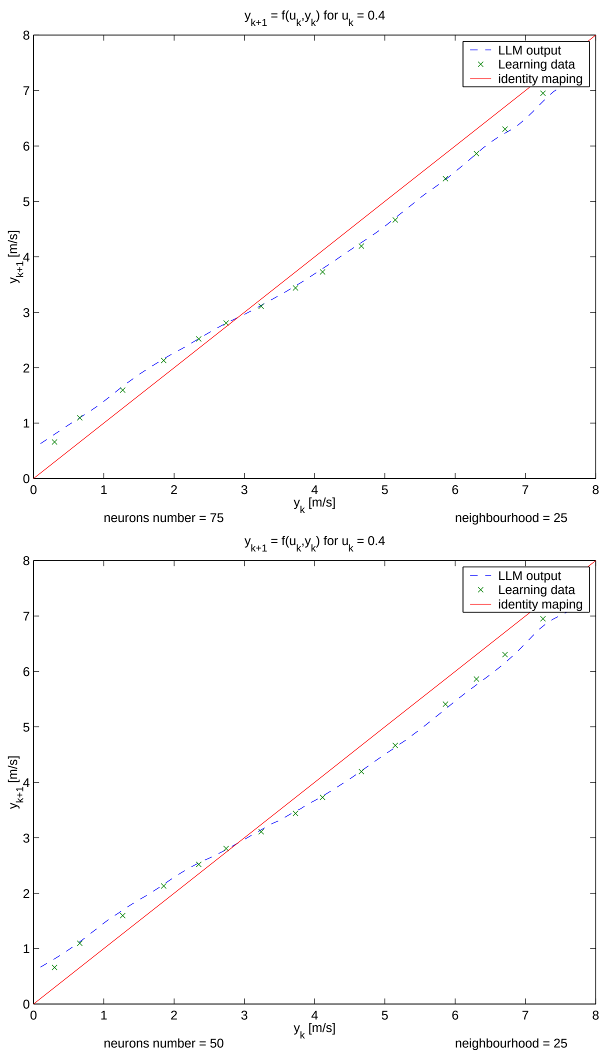

Fig. 3. Cross-section of the approximation surface for throttle angle $u_k = 0.4$ and neighbourhood's radius 25 neurons. Network with 75 neurons learned with raw data (top). Network with 50 neurons learned with pre-processed data (bottom).

$45, 50, 55, 60, 65, 70, 75$ neurons. In addition, we have also changed the neighbourhood's radius, by calculation of the model output according to equation (11). Neighbourhood's radius has the influence on weighted function Φ from equation (3). We simply apply this variable by changing a number of neurons taken into consideration from one to all neurons for the network. We have tested both static and dynamic behaviour of neural network model. Static characteristic was tested through a presentation of test value of u - throttle's angle and speed for the same time sample. In this way we have received an approximation surface for our neural model. A cross-section for approximation surface for throttle angle $u = 0.4$ is shown in Figure 3.

Dynamic characteristics of the model was tested through throttle's angle step function. We have simulated this behaviour with SIMULINK S-function. Because of the non-linearity of the plant, both an acceleration from low speed to the given speed and deceleration from high speed to the same given speed has been tested. Thus, we have received, for each step function, two characteristics as shown in Figure 4 respectively. Acceleration and deceleration were investigated for different initial speed, i.e. different y_{t0} value. Results for neural network with 75 neurons trained with raw data set and for network with 50 neurons, trained with pre-processed data, 25 neighbours and throttle's angle $u = 0.4$ accordingly, are shown in Figure 3 and 4. Approximation error for steady-state was calculated from equation (13), see Figure 5.

$$e_j = \frac{y_j - \tilde{y_j}}{y_j} \tag{13}$$

We have evaluated the dynamical character of the approximation via Square Error (SE) at time instant k for given neighbourhood's radius and throttle's angle u_j, calculated according to equation

$$e_{j_{SE}}(k) = \|y_j(k) - \tilde{y}_j(k)\|^2. \tag{14}$$

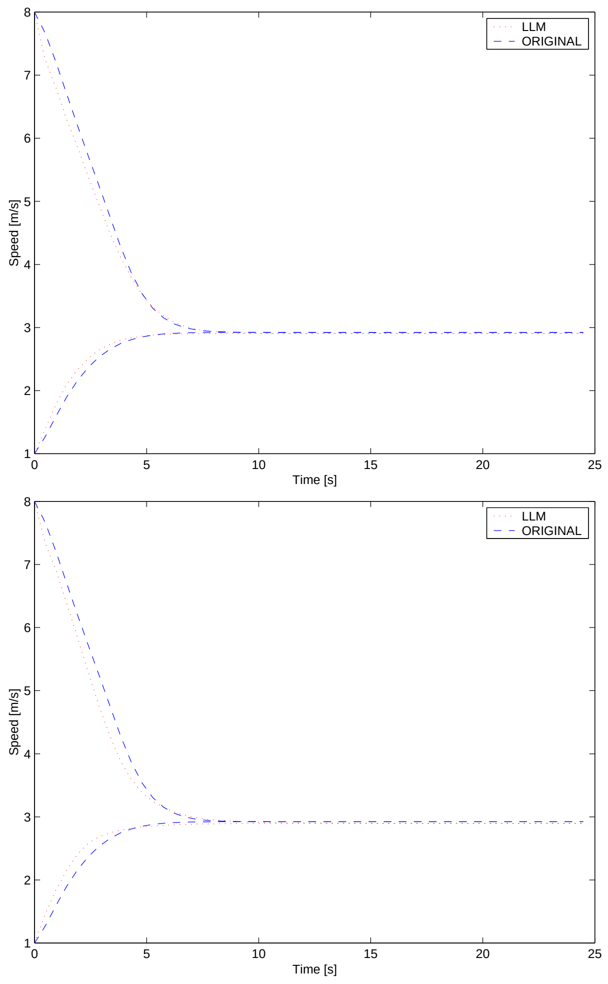

Fig. 4. Response of the neural network model for step function and neighbourhood's radius 25 neurons. Network with 75 neurons trained with raw data (top). Network with 50 neurons trained with pre-processed data (bottom).

We have also calculated a Mean Square Error (MSE) for given neighbourhood's radius and throttle's angle during acceleration and deceleration in discrete time instants $k, k+1, \ldots, k+p$ according to equation

$$e_{MSE} = \frac{1}{p+1} \sum_{i=0}^{i=p} e_{j_{SE}}(k+i). \qquad (15)$$

MSE and error in steady-state for variable throttle's angle and constant neighbourhood's radius for network learned with raw data and pre-processed data are shown in Figure 5. We have compared approximation by network with 75 neurons, trained with raw data and network with 50 neurons trained with pre-processed data. We have achieved good results for neighbourhood's radius from 3 to 50 neurons taken into account by network's output calculation. In both cases we have achieved mean approximation's error in steady-state about 2% (see Table 1, 2).

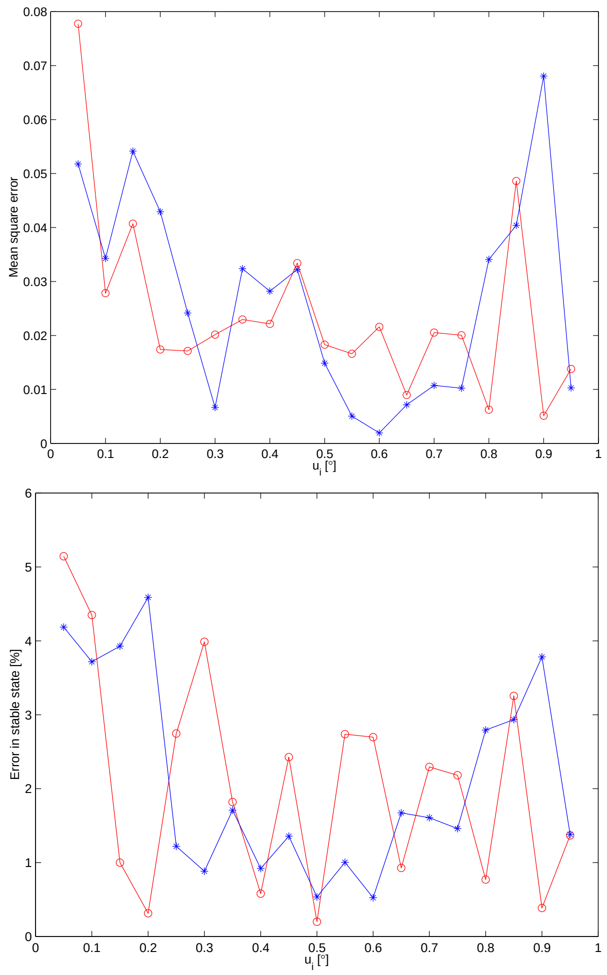

Fig. 5. Approximation error for neural network model for neighbourhood's radius 25 neurons. MSE for different throttle angle (top). Error in steady-state for different throttle (bottom). $'\circ'$ network with 75 neurons trained with raw data; $'\star'$ network with 50 neurons learned with pre-processed data.

Table 1. Mean error in steady-state for network with 75 neurons trained with raw data and for various neighbourhood's radius.

Neurons no.	1	2	3	5	30	50	75
decel. [%]	3.6	2.8	2	2.3	2.1	1.8	4.7
accel. [%]	3.6	2.8	2.5	2.4	2.1	1.8	4.7

Table 2. Mean error in steady-state for network with 50 neurons learned with pre-processed data and for various neighbourhood's radius.

Neurons no.	1	2	3	5	10	25	50
decel. [%]	2.7	2.3	2	2	2.3	3.1	4.4
accel. [%]	2.3	2.3	1.9	2.1	2.3	2.1	4.4

4. CONCLUSIONS

We have introduced the neural network with local linear mapping for approximation of non-linear control system. We have achieved good results in estimation of system's parameters and shown, that we can use also raw data without any pre-processing (data's normalization excluded). The drawback of raw data "approach" is that we have needed to use more neurons for neural model. The problem of neurons number can be omitted by using a neural network with variable neurons number which we would like to apply in the future work.

During our investigations we have noticed some approximation errors:

(1) steady-state error at some setpoints (values for acceleration and deceleration may differ).
(2) slight differences in transients between plant and neural model.
(3) over-shoot in plant trajectory as opposed to monotonic neural trajectory for u around 0.95.

Two first errors are caused by quantisation error and errors connected with output calculation using weighted local linear mapping. As a result of the quantisation problem, neural model trajectory has changed according to neurons positions or around them sometimes ending in different stable points, consider weighting function Φ. Approximation through a weighted linear system has no high order elements in equations as opposed to the original non-linear system. That can cause the difference in over-shoot. The ways to improve the approximation accuracy are among the topics of further research.

ACKNOWLEDGEMENTS

Waldemar Graniszewski would also like to acknowledge Professor Dr Christoph von der Malsburg, Professor Dr Thomas Martinetz, Professor Dr Werner von Seelen, Doctor Christoph Engels, Doctor Stephan Gehlen and Doctor Wolfgang Konen for their interesting suggestions, discussions and help during his stay in ZN Vision Technologies AG (Zentrum für Neuroinformatik).

5. REFERENCES

Chen, S. and S. A. Billings (1989). Representation of non-linear systems: the NARMAX model. *International Journal of Control* **49**, 1013–1032.

DeSieno, D. (1988). Adding a conscience to competitive learning. *Neural Networks* **1**, 117–124.

Gray, R.M. (1984). Vector quantization. *IEEE ASSP Mag.* **1**(2), 4–29.

Johansen, T. A. and R. Murray-Smith (1997). The operating regime approach. In: *Multiple Model Approaches to Modelling and Control* (R. Murray-Smith and T. A. Johansen, Eds.). Taylor & Francis. London.

Kalkkuhl, J. C. and K. J. Hunt (1996). Discrete-time neural model structures for continuous nonlinear systems: Fundamental properties and control aspects. In: *Neural Adaptive Control Technology* (R. Żbikowski and K. J. Hunt, Eds.). Robotics and Intelligent Systems. World Scientific. Singapore, London.

Kohonen, T. (1995). *Self-organizing map*. Springer Verlag. Berlin.

Lakshmikantham, V. and D. Trigiante (1988). *Theory of Difference Equations: Numerical Methods and Applications*. Academic Press. Boston.

Leontaritis, I. J. and S. A. Billings (1985). Input-output parametric models for non-linear systems. Part I: deterministic non-linear systems. *International Journal of Control* **41**, 303–328.

Linde, Y., A. Buzo and Gray R. (1980). An algorithm for vector quantizer design. *IEEE Trans. Comm.* **28**, 84–95.

Martinetz, T. and K. Schulten (1991). A "Neural-Gas" network learns topologies. In: *Proceedings of the International Conferrence on Artificial Neural Networks*. Elsevier. Helsinki.

Martinetz, T., S. Berkovich and K. Schulten (1993). "Neural-Gas" network for vector quantization and its application to time-series prediction. *IEEE Transactions on Neural Networks* **4**(4), 558–569.

Nelles, O. (1996). Local linear Model Trees for On-Line Identification of Time-Variant Nonlinear Dynamic Systems. In: *Proceedings of the International Conferrence on Artificial Neural Networks (ICANN)*. Bochum.

Normand-Cyrot, D. (1987). Eléments d'analyse et de synthèse des systèmes non linéaires. Notes de cours du D.E.A. "Automatique et Traitement du Signal".

Ritter, H. (1991). Learning with the Self-Organizing Map. In: *Proceedings of the International Conferrence on Artificial Neural Networks*. Elsevier. Helsinki.