Multiple Local ARX Modeling for System Identification Using the Self-Organizing Map

Luís Gustavo M. Souza[1] and Guilherme A. Barreto[2] *

1,2- Federal University of Ceará
Department Teleinformatics Enginnering
Av. Mister Hull, S/N - Campus of Pici, Center of Technology
CP 6005, CEP 60455-760, Fortaleza, CE, Brazil

Abstract. In this paper we build global NARX (Nonlinear Auto-Regressive with eXogenous variables) models from multiple local linear ARX models whose state spaces have been partitioned through Kohonen's Self-Organizing Map. The studied models are evaluated in the task of identifying the inverse dynamics of a flexible robotic arm. Simulation results demonstrate that SOM-based multiple local ARX models perform better than a single ARX model and an MLP-based global NARX models.

1 Problem Formulation

Several complex dynamical systems which can be described by the NARX model:

$$y(t) = f[y(t-1), \ldots, y(t-n_y); u(t), u(t-1), \ldots, u(t-n_u+1)], \qquad (1)$$

where n_y and n_u are the (memory) orders of the dynamical model. In words, Eq. (1) states that the system output y at time t depends, on the past n_y output values and on the past n_u values of the input u. In many situations, it is also desirable to approximate the inverse mapping of a nonlinear plant, given by

$$u(t) = f^{-1}[y(t-1), \ldots, y(t-n_y); u(t-1), \ldots, u(t-n_u)]. \qquad (2)$$

In system identification, the goal is to obtain estimates of $f(\cdot)$ and/or $f^{-1}(\cdot)$ from available input-output time series data $\{u(t), y(t)\}_{t=1}^{M}$.

SOM-based local dynamic modeling and control approaches have been successfully applied to complex system identification and control tasks [1, 2, 3, 4, 5], but these contributions still remain widely unknown by the Machine Learning and Statistics communities. In this paper, we compare the performances of system identification techniques which rely on the self-organizing map (SOM) [6] for local function approximation. To the best of our knowledge, such an evaluation has not been reported elsewhere. In this sense, this is one of the main contributions of this paper.

For the SOM and other unsupervised networks to be able to learn dynamical mappings, they must have some type of *short-term memory* (STM) mechanism. That is, the SOM should be capable of temporarily storing past information about the system input and output vectors. There are several STM models, such

*The authors thank FUNCAP (grant #1469/07) and CAPES/PRODOC for the financial support.

as delay lines, leaky integrators, reaction-diffusion mechanisms and feedback loops [7, 8], which can be incorporated into the SOM to allow it to approximate a dynamical mapping $f(\cdot)$ or its inverse $f^{-1}(\cdot)$. In order to draw a parallel with standard system identification approaches, we limit ourselves to describe the VQTAM approach in terms of time delays as STM mechanisms.

The remainder of the paper is organized as follows. In Section 2, SOM architecture and its learning process are described. In Section 3, two SOM-based local ARX models are introduced. Simulations and performance are presented in Section 4. The paper is concluded in Section 5.

2 The Self-Organizing Map

The SOM is composed of two fully connected layers: an input layer and a competitive layer. The input layer simply receives the incoming input vector and forwards it to the competitive layer through weight vectors. The goal of SOM is to represent the input data distribution by the distribution of the weight vectors. Competitive learning drives the winning weight vector to become more similar to the input data. Throughout this paper, we represent the weight vector between input layer and neuron i as

$$\mathbf{w}_i = (w_{i,1}, w_{i,2}, \ldots, w_{i,j}, \ldots, w_{i,p})^T, \tag{3}$$

where $w_{i,j} \in \mathbb{R}$ denotes the weight connecting node j in the input layer with neuron i, and p, is the dimension of the input vector. In what follows, a brief description of the original SOM algorithm is given.

Firstly we use Euclidean distance metric to find the current winning neuron, $i^*(t)$, as given by the following expression:

$$i^*(t) = \arg \min_{\forall i \in \mathcal{A}} \|\mathbf{x}(t) - \mathbf{w}_i(t)\| \tag{4}$$

where $\mathbf{x}(t) \in \mathbb{R}^p$ denotes the current input vector, $\mathbf{w}_i(t) \in \mathbb{R}^p$ is the weight vector of neuron i, and t denotes the iterations of the algorithm. Secondly, it is necessary to adjust the weight vectors of the winning neuron and of those neurons belonging to its neighborhood:

$$\mathbf{w}_i(t+1) = \mathbf{w}_i(t) + \alpha(t)h(i^*, i; t)[\mathbf{x}(t) - \mathbf{w}_i(t)] \tag{5}$$

where $0 < \alpha(t) < 1$ is the learning rate and $h(i^*, i; t)$ is a gaussian weighting function that limits the neighborhood of the winning neuron:

$$h(i^*, i; t) = \exp\left(-\frac{\|\mathbf{r}_i(t) - \mathbf{r}_{i^*}(t)\|^2}{2\sigma^2(t)}\right) \tag{6}$$

where $\mathbf{r}_i(t)$ and $\mathbf{r}_{i^*}(t)$, are respectively, the positions of neurons i and i^* in a predefined output array where the neurons are arranged in the nodes, and $\sigma(t) > 0$ defines the radius of the neighborhood function at time t.

The variables $\alpha(t)$ and $\sigma(t)$ should both decay with time to guarantee convergence of the weight vectors to stable steady states. In this paper, we adopt an exponential decay for both, given by:

$$\alpha(t) = \alpha_0 \left(\frac{\alpha_T}{\alpha_0}\right)^{(t/T)} \quad \text{and} \quad \sigma(t) = \sigma_0 \left(\frac{\sigma_T}{\sigma_0}\right)^{(t/T)} \tag{7}$$

where α_0 (σ_0) and α_T (σ_T) are the initial and final values of $\alpha(t)$ ($\sigma(t)$), respectively. The operations defined by Eqs. (4) and (7) are repeated until a steady state of global ordering of the weight vectors has been achieved. In this case, we say that the map has converged.

The resulting map also preserves the topology of the input samples in the sense that adjacent input patterns are mapped into adjacent neurons on the map. Due to this topology-preserving property, the SOM is able to cluster input information and spatial relationships of the data on the map. This clustering ability of the SOM has shown to be quite useful for the identification of nonlinear dynamical systems [9]. However, the number of neurons required by the SOM to provide a good approximation of a given input-output mapping is very high, specially when compared to the MLP and RBF neural networks. To alleviate this limitation of the plain SOM algorithm to some extent, we introduce two SOM-based multiple local ARX models.

3 Multiple Local ARX Models Based on the SOM

In this section, we describe two approaches to the system identification problem that use the SOM as a building block. The basic idea behind both is the partitioning of the input space into non-overlapping regions, called Voronoi cells, whose centroids correspond to the weight vectors of the SOM. Then an interpolating hyperplane is associated with each Voronoi cell or to a small subset of them, in order to estimate the output.

3.1 Local Linear Mapping

The first architecture to be described is called *Local Linear Mapping* (LLM) [10]. The basic idea of the LLM is to associate each neuron in the SOM with a conventional FIR/LMS linear filter. The SOM array is used to quantize the input space in a reduced number of prototype vectors (and hence, Voronoi cells), while the filter associated with the winning neuron provides a local linear estimator of the output of the mapping being approximated.

Thus, for the inverse modeling task of interest, each input vector $\mathbf{x}(t) \in \mathbb{R}^{p+q}$ is defined as

$$\mathbf{x}(t) = [u(t-1), \ldots, u(t-q); y(t-1), \ldots, y(t-p)]^T. \tag{8}$$

Clustering (or vector quantization) of the input space $\mathcal{X}$ is performed by the LLM as in the usual SOM algorithm, with each neuron i owning a prototype vector $\mathbf{w}_i$, $i = 1, \ldots, N$.

Additionally, there is a coefficient vector $\mathbf{a}_i \in \mathbb{R}^{p+q}$ associated to each weight vector $\mathbf{w}_i$, which plays the role of the coefficients of an (linear) ARX model:

$$\mathbf{a}_i(t) = [b_{i,1}(t), \ldots, b_{i,q}(t), a_{i,1}(t), \ldots, a_{i,p}(t)]^T. \tag{9}$$

The output value is provided by one of the local ARX model as follows

$$\begin{aligned} \hat{u}(t) &= \sum_{k=1}^{q} b_{i^*,k}(t)u(t-k) + \sum_{l=1}^{p} a_{i^*,l}(t)y(t-l) \\ &= \mathbf{a}_{i^*}^T(t)\mathbf{x}(t), \end{aligned} \tag{10}$$

where $\mathbf{a}_{i^*}(t)$ is the coefficient vector associated with the winning neuron $i^*(t)$. From Eq. (10), one can easily note that the coefficient vector $a_{i^*}(t)$ is used to build a local linear approximation of the output of the desired nonlinear mapping.

Since the adjustable parameters of the LLM algorithm are the set of prototype vectors $\mathbf{w}_i(t)$ and their associated coefficient vectors $\mathbf{a}_i(t), i = 1, 2, \ldots, p+q$, we need two learning rules. The rule for updating the prototype vectors $\mathbf{w}_i$ follows exactly the one given in Eq. (5). The learning rule of the coefficient vectors $\mathbf{a}_i(t)$ is an extension of the normalized LMS algorithm, that also takes into account the influence of the neighborhood function $h(i^*, i; t)$:

$$\mathbf{a}_i(t+1) = \mathbf{a}_i(t) + \alpha' h(i^*, i; t)\Delta\mathbf{a}_i(t), \tag{11}$$

where $0 < \alpha' \ll 1$ denotes the learning rate of the coefficient vector, and $\Delta\mathbf{a}_i(t)$ is the error correction rule of Widrow-Hoff, given by

$$\Delta\mathbf{a}_i(t) = \left[u(t) - \mathbf{a}_i^T(t)\mathbf{x}(t)\right] \frac{\mathbf{x}(t)}{\|\mathbf{x}(t)\|^2}, \tag{12}$$

where $u(t)$ is the desired output of the inverse mapping being approximated.

3.2 Prototype-Based Local Least-Squares Model

The algorithm to be described in this section, called K-winners SOM (KSOM), was originally applied to nonstationary time series prediction [5]. In this paper we aim to evaluate this architecture in the context of nonlinear system identification. For training purposes, the KSOM algorithm depends on the VQTAM (*Vector-Quantized Temporal Associative Memory*) model [9], which is a simple extension of the SOM algorithm that simultaneously performs vector quantization on the input and output spaces of a given nonlinear mapping.

In the VQTAM model, the input vector at time step t, $\mathbf{x}(t)$, is composed of two parts. The first part, denoted $\mathbf{x}^{in}(t) \in \mathbb{R}^{p+q}$, carries data about the input of the dynamic mapping to be learned. The second part, denoted $x^{out}(t) \in \mathbb{R}$, contains data concerning the desired output of this mapping. The weight vector of neuron i, $\mathbf{w}_i(t)$, has its dimension increased accordingly. These changes are formulated as follows:

$$\mathbf{x}(t) = \begin{pmatrix} \mathbf{x}^{in}(t) \\ x^{out}(t) \end{pmatrix} \text{ and } \mathbf{w}_i(t) = \begin{pmatrix} \mathbf{w}_i^{in}(t) \\ w_i^{out}(t) \end{pmatrix}, \tag{13}$$

where $\mathbf{w}_i^{in}(t) \in \mathbb{R}^{p+q}$ and $w_i^{out}(t) \in \mathbb{R}$ are, respectively, the portions of the weight (prototype) vector which store information about the inputs and the outputs of the desired mapping. Depending on the variables chosen to build the vector $\mathbf{x}^{in}(t)$ and scalar $x^{out}(t)$ one can use the SOM algorithm to learn the forward or the inverse mapping of a given plant (system). For instance, if the interest is in inverse identification, then we define

$$\mathbf{x}^{in}(t) = [u(t-1), \ldots, u(t-q); y(t-1), \ldots, y(t-p)]^T \quad \text{and} \quad x^{out}(t) = u(t). \tag{14}$$

The winning neuron at time step t is determined based only on $\mathbf{x}^{in}(t)$, i.e.

$$i^*(t) = \arg \min_{\forall i \in \mathcal{A}} \{\|\mathbf{x}^{in}(t) - \mathbf{w}_i^{in}(t)\|\}. \tag{15}$$

For updating the weights, however, both $\mathbf{x}^{in}(t)$ and $x^{out}(t)$ are used:

$$\Delta\mathbf{w}_i^{in}(t) = \alpha(t)h(i^*, i; t)[\mathbf{x}^{in}(t) - \mathbf{w}_i^{in}(t)] \tag{16}$$

$$\Delta w_i^{out}(t) = \alpha(t)h(i^*, i; t)[x^{out}(t) - w_i^{out}(t)] \tag{17}$$

where $0 < \alpha(t) < 1$ is the learning rate, and $h(i^*, i; t)$ is a time-varying Gaussian neighborhood function defined as in Eq. (6).

The learning rule in Eq. (16) performs topology-preserving vector quantization on the input space, while the rule in Eq. (17) acts similarly on the output space of the mapping being learned. As the training proceeds, the SOM learns to associate the input prototype vectors $\mathbf{w}_i^{in}$ with the corresponding output prototype vectors $\mathbf{w}_i^{out}$. The SOM-based associative memory implemented by the VQTAM can then be used for function approximation purposes.

Since the VQTAM is essentially a vector quantization algorithm, it requires too many neurons to provide small prediction errors when approximating continuous mappings. This limitation can be somewhat alleviated through the use of interpolation methods specially designed for the SOM architecture, such as geometric interpolation [11] and topological interpolation [12]. Another possibility is to devise a local linear interpolation strategy over the neighborhood of the winning neuron. For example, after training the VQTAM model, the coefficient vector $\mathbf{a}(t)$ of a local ARX model for estimating the mapping output is computed for each time step t by the standard least-squares estimation (LSE) technique, using the weight vectors of the K ($K \gg 1$) neurons closest to the current input vector, instead of using the original data vectors.

Let the set of K winning weight vectors at time t to be denoted by $\{\mathbf{w}_{i_1^*}, \mathbf{w}_{i_2^*}, \ldots, \mathbf{w}_{i_K^*}\}$. Recall that due to the VQTAM training style, each weight vector $\mathbf{w}_i(t)$ has a portion associated with $\mathbf{x}^{in}(t)$ and other associated with $x^{out}(t)$. So, the KSOM uses the corresponding K pairs of prototype vectors $\{\mathbf{w}_{i_k^*}^{in}(t), w_{i_k^*}^{out}(t)\}_{k=1}^K$, with the aim of building a local linear function at time t:

$$w_{i_k^*}^{out} = \mathbf{a}^T(t)\mathbf{w}_{i_k^*}^{in}(t), \qquad k = 1, \ldots, K \tag{18}$$

where $\mathbf{a}(t) = [b_1(t), \ldots, b_q(t), a_1(t), \ldots, a_p(t)]^T$ is a time-varying coefficient vector. Equation (18) can be written in a matrix form as

$$\mathbf{w}^{out}(t) = \mathbf{R}(t)\mathbf{a}(t), \tag{19}$$

where the output vector $\mathbf{w}^{out}$ and the regression matrix $\mathbf{R}$ at time t are defined as follows

$$\mathbf{w}^{out}(t) = [w^{out}_{i_1^*,1}(t) \;\; w^{out}_{i_2^*,1}(t) \;\; \cdots \;\; w^{out}_{i_K^*,1}(t)]^T \tag{20}$$

and

$$\mathbf{R}(t) = \begin{pmatrix} w^{in}_{i_1^*,1}(t) & w^{in}_{i_1^*,2}(t) & \cdots & w^{in}_{i_1^*,p+q}(t) \\ w^{in}_{i_2^*,1}(t) & w^{in}_{i_2^*,2}(t) & \cdots & w^{in}_{i_2^*,p+q}(t) \\ \vdots & \vdots & \vdots & \vdots \\ w^{in}_{i_K^*,1}(t) & w^{in}_{i_K^*,2}(t) & \cdots & w^{in}_{i_K^*,p+q}(t) \end{pmatrix}. \tag{21}$$

The coefficient vector $\mathbf{a}(t)$ is then computed by the following Tikhonov-regularized pseudoinverse (minimum norm) procedure

$$\mathbf{a}(t) = \left(\mathbf{R}^T(t)\mathbf{R}(t) + \lambda\mathbf{I}\right)^{-1}\mathbf{R}^T(t)\mathbf{w}^{out}(t), \tag{22}$$

where $\mathbf{I}$ is a identity matrix of order K and $\lambda > 0$ (e.g. $\lambda = 0.001$) is a small regularization constant. Once $\mathbf{a}(t)$ is estimated, we can locally approximate the output of the nonlinear mapping by the output of the following ARX model:

$$\hat{u}(t) = \sum_{k=1}^{q} b_k(t)u(t-k) + \sum_{l=1}^{p} a_l(t)y(t-l) = \mathbf{a}^T(t)\mathbf{x}^{in}(t)$$

The KSOM can be considered a local (linear) ARX model due to the use of a subset of K weight vectors chosen from the whole set of N weight vectors. This is one of the differences between KSOM and the LLM approaches. While the former uses $K \ll N$ prototype vectors to build the local linear model, the latter uses a single prototype. Another difference is that the LLM approach uses a LMS-like learning rule to update the coefficient vector of the winning neuron. Once training is completed all coefficient vectors $\mathbf{a}_i$, $i = 1, \ldots, N$, are frozen for posterior use. The KSOM, instead, uses a LSE-like procedure to find the coefficient vector $\mathbf{a}(t)$ each time an input vector is presented, so that a single linear mapping is built at each time step.

Cho *et al.* [3] proposed a neural architecture that is equivalent to the KSOM in the sense that the coefficient vector $\mathbf{a}(t)$ is computed from K prototype vectors of a trained SOM using the LSE technique. However, the required prototype vectors are not selected as the K nearest prototypes to the current input vector, but rather automatically selected as the winning prototype at time t and its $K-1$ topological neighbors. If topological defects are present, as usually occurs for multidimensional data, the KSOM provides more accurate results.

Chen and Xi [13] also proposed a local linear regression model whose coefficient vectors are computed using the prototypes of a competitive learning network through the recursive least-squares algorithm. However, the competitive network used by Chen and Xi does not have the topology-preserving properties of the SOM algorithm, which has shown to be important for system identification purposes [9].

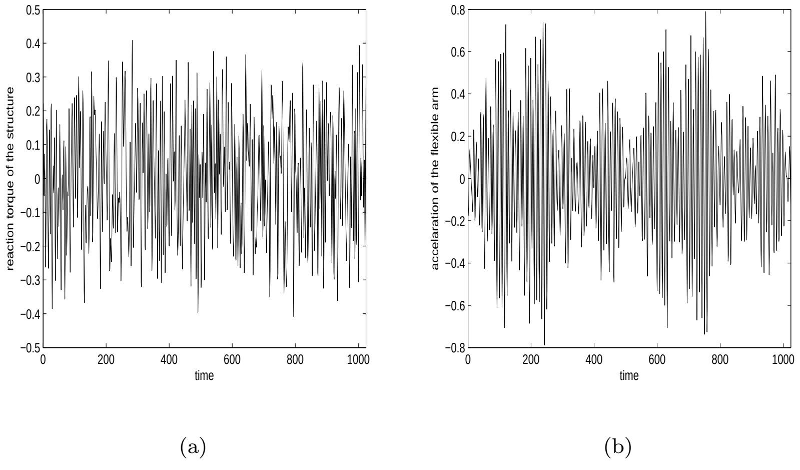

Fig. 1: Measured values of reaction torque of the structure (a) and acceleration of the flexible arm (b).

4 Computer Simulations and Discussion

The proposed SOM-based multiple local ARX models are evaluated in the identification of the inverse dynamics of a flexible robot arm. The arm is installed on an electric motor. We want to model the input-output mapping from the measured reaction torque of the structure on the ground to the acceleration of the flexible arm[1]. Figure 1 shows the measured values of the reaction torque of the structure (input time series, $\{u(t)\}$) and the acceleration of the flexible arm (output time series, $\{y(t)\}$).

For the sake of completeness, the LLM- and KSOM-based local ARX models are compared with an one-hidden-layer MLP trained by the standard backpropagation algorithm (MLP-1h), another one-hidden-layer MLP trained by the Levenberg-Marquardt (MLP-LM) algorithm and, finally, a two-hidden-layer MLP (MLP-2h) trained by the standard backpropagation algorithm. All these global NARX models are also compared with the linear *Auto-Regressive with eXogenous variables* (ARX) model, trained on-line through the plain LMS algorithm.

For all MLP-based global NARX models, the activation function of the hidden neurons is the hyperbolic tangent function, while the output neuron uses a linear one. After some experimentation, the best configuration of the MLP-1h and MLP-LM models have 30 hidden neurons. For the MLP-2h, the number of

[1]These data were obtained in the framework of the Belgian Programme on Interuniversity Attraction Poles (IUAP-nr.50) initiated by the Belgian State.

Table 1: Performances of the global and local models for the robotic arm data.

Neural Models	**NMSE**			
	mean	*min*	*max*	*variance*
KSOM	0.0064	0.0045	0.0117	1.83×10^{-6}
KSOM-PL	0.0187	0.0082	0.0657	8.47×10^{-5}
MLP-LM	0.1488	0.0657	0.4936	0.0107
MLP-1h	0.1622	0.1549	0.1699	1.03×10^{-5}
VQTAM-T	0.1669	0.1199	0.2263	6.14×10^{-4}
LLM	0.3176	0.2685	0.3558	2.23×10^{-4}
ARX	0.3848	0.3848	0.3848	0.0445
VQTAM-G	0.4968	0.3700	0.6458	0.0024
MLP-2h	0.6963	0.5978	1.5310	0.0368

neurons in second hidden layer is heuristically set to half the number of neurons in the first hidden layer. The learning rate for the MLPs was set to 0.1.

During the prediction phase, the neural models should compute the estimation error (residuals) $e(t) = u(t) - \hat{u}(t)$, where $u(t)$ is desired output and $\hat{u}(t)$ is the estimate provided by each neural model. The performances of all models are assessed through the *normalized mean squared error* (NMSE):

$$NMSE = \frac{\sum_{t=1}^{M} e^2(t)}{M \cdot \hat{\sigma}_u^2} = \frac{\sum_{t=1}^{M}(u(t) - \hat{u}(t))^2}{M \cdot \hat{\sigma}_u^2} \tag{23}$$

where $\hat{\sigma}_u^2$ is the variance of the original time series $\{u(t)\}_{t=1}^{M}$ and M is the length of the sequence of residuals.

The models are trained using the first 820 samples of the input-output signal sequences (approximately, 80% of the total) and tested with the remaining 204 samples. The input and output memory orders are set to $p = 4$ and $q = 5$, respectively. The obtained results are shown in Table 1, where are displayed the mean, minimum, maximum and variance of the NMSE values, measured along the 100 training/testing runs. The weights of the neural models were randomly initialized at each run. In this table, the models are again sorted in increasing order of the mean NMSE values.

The number of neurons for all SOM-based local ARX models is set to 30. For the KSOM-based local NARX model, K is equal to 25. For each SOM-based model, the initial and final learning rates are set to $\alpha_0 = 0.5$ and $\alpha_T = 0.01$. The initial and final values of radius of the neighborhood function are $\sigma_0 = N/2$ and $\sigma_T = 0.001$, where N, the number of neurons in the SOM, is set to 30. The learning rate α' is set to 0.1.

For the sake of curiosity, the VQTAM with topological (VQTAM-T) and geometric (VQTAM-G) interpolations have been tested with the hope of improving the approximation accuracy of the plain VQTAM model. The KSOM-based local ARX model was also implemented using the recently proposed *Parameter-less* SOM (PLSOM) architecture [14], which requires no annealing of the learning

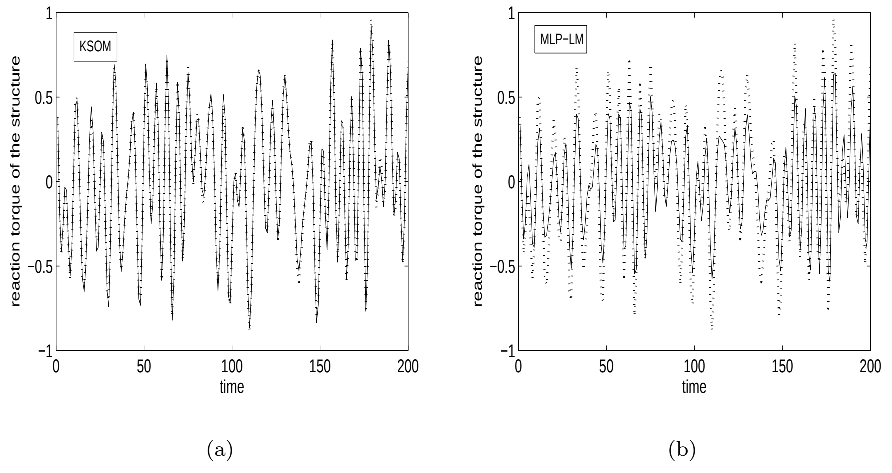

Fig. 2: Typical estimated sequences of reaction torque of the structure provided by the KSOM and MLP-LM models. Dashed lines denote actual sample values, while the solid line indicates the estimated sequence.

rate and neighborhood width parameters. The fundamental difference between the PLSOM and the SOM is that while the SOM depends on the learning rate and neighborhood size to decrease over time, e.g., as a function of the number of iterations of the learning algorithm, the PLSOM calculates these values based on the local quadratic fitting error of the map to the input space.

The performance of KSOM-based local ARX model on this real-world application is by far the best one, even better than the MLP-based global NARX models. A better performance of the KSOM-based model in comparison to the LLM-based model is also verified. This can be partly explained by the fact that the parameters of the KSOM-based local model are estimated in a batch mode from the K closest prototypes, while the parameters of the LLM-based model are estimated in an online mode.

The LLM-based local ARX model performed only better than the ARX, VQTAM-G and MLP-2h models. The performances of these three models were very poor. The performance of the VQTAM-T is statistically equivalent to that of the MLP-1h model. Among the MLP-based models, the use of second-order learning algorithm was crucial to the good performance of the MLP-LM model.

Finally, Figure 2 shows typical sequences of estimated values of the reaction torque of the structure provided by the best local and global NARX models. Figure 2a shows the sequence generated by the KSOM-based model, while Figure 2b shows the sequence estimated by the MLP-LM model.

5 Conclusion

We have attempted to tackle the problem of nonlinear system identification using the local linear modeling methodology. For that purpose we presented two multiple local ARX models based on Kohonen's self-organizing map and evaluated them in the identification of the inverse dynamics of one real-world data set, a robot arm. The first local ARX model builds a fixed number of local ARX models, one for each Voronoi region associated with the prototype vectors of the SOM. The second one builds only a single local ARX model using the prototypes vectors closest to the current input vector. It has been shown for the robot arm data set the KSOM-based local ARX model presented the best performance among all models.

References

[1] J. Cho, J. Principe, D. Erdogmus, and M. Motter. Quasi-sliding mode control strategy based on multiple linear models. *Neurocomputing*, 70(4-6):962–974, 2007.

[2] I. Díaz-Blanco, A. A. Cuadrado-Vega, A. B. Diez-González, J. J. Fuertes-Martínez, M. Domínguez-González, and P. Reguera-Acevedo. Visualization of dynamics using local dynamic modelling with self-organizing maps. *Lecture Notes on Computer Science*, 4668:609–617, 2007.

[3] J. Cho, J. Principe, D. Erdogmus, and M. Motter. Modeling and inverse controller design for an unmanned aerial vehicle based on the self-organizing map. *IEEE Transactions on Neural Networks*, 17(2):445–460, 2006.

[4] J. Lan, J. Cho, D. Erdogmus, J. C. Principe, M. A. Motter, and J. Xu. Local linear PID controllers for nonlinear control. *International Journal of Control and Intelligent Systems*, 33(1):26–35, 2005.

[5] G.A. Barreto, J.C.M. Mota, L.G.M. Souza, and R.A. Frota. Nonstationary time series prediction using local models based on competitive neural networks. *Lecture Notes in Computer Science*, 3029:1146–1155, 2004.

[6] T. K. Kohonen. *Self-Organizing Maps*. Springer-Verlag, Berlin, Heidelberg, 2nd extended edition, 1997.

[7] J. C. Principe, N. R. Euliano, and S. Garani. Principles and networks for self-organization in space-time. *Neural Networks*, 15(8–9):1069–1083, 2002.

[8] G. A. Barreto and A. F. R. Araújo. Time in self-organizing maps: an overview of models. *International Journal of Computer Research*, 10(2):139–179, 2001.

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

[10] J. Walter, H. Ritter, and K. Schulten. Non-linear prediction with self-organizing map. In *Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN'90)*, volume 1, pages 587–592, 1990.

[11] J. Göppert and W. Rosenstiel. Topology preserving interpolation in selforganizing maps. In *Proceedings of the NeuroNIMES'93*, pages 425–434, 1993.

[12] J. Göppert and W. Rosenstiel. Topological interpolation in som by affine transformations. In *Proceedings of the European Symposium on Artificial Neural Networks (ESANN'95)*, pages 15–20, 1995.

[13] J.-Q. Chen and Y.-G. Xi. Nonlinear system modeling by competitive learning and adaptive fuzzy inference system. *IEEE Transactions on Systems, Man, and Cybernetics-Part C*, 28(2):231–238, 1998.

[14] E. Berglund and J. Sitte. The parameterless self-organizing map algorithm. *IEEE Transactions on Neural Networks*, 17(2):305–316, 2006.