Preprinted Version, Neurocomputing 175 (2016) 433–442

A combined B-Spline-Neural-Network and ARX Model for Online Identification of Nonlinear Dynamic Actuation Systems

Michele Folgheraiter

Robotics and Mechatronics Department
Nazarbayev University, School of Science and Technology
53 Kabanbay Batyr Ave, Astana, Kazakhstan
michele.folgheraiter@nu.edu.kz

Abstract

This paper presents a block oriented nonlinear dynamic model suitable for online identification.The model has the well known Hammerstein architecture where as a novelty the nonlinear static part is represented by a B-spline neural network (BSNN), and the linear static one is formalized by an auto regressive exogenous model (ARX). The model is suitable as a feed-forward control module in combination with a classical feedback controller to regulate velocity and position of pneumatic and hydraulic actuation systems which present non stationary nonlinear dynamics. The adaptation of both the linear and nonlinear parts is taking place simultaneously on a patter-by-patter basis by applying a combination of error-driven learning rules and the recursive least squares method. This allows to decrease the amount of computation needed to identify the model's parameters and therefore makes the technique suitable for real time applications. The model was tested with a silver box benchmark and results show that the parameters are converging to a stable value after 1500 samples, equivalent to 7.5s of running time. The comparison with a pure ARX and BSNN model indicates a substantial improvement in terms of the RMS error, while the comparison with alternative non linear dynamic models like the NNOE and NNARX, having the same number of parameters but greater computational complexity, shows comparable performances.

Keywords:
Hammerstein Model, B-Spline Neural Network, Nonlinear Dynamic Model,

Real-time Model Adaptation, Adaptive Control, Humanoid Robotics, Hydraulic Actuation Systems.

1. Introduction

When it comes to design the control system of highly redundant robots (e.g. humanoid robots, legged robots, continuum robots, etc.) it is crucial to have available an accurate model of their actuation system. The model can be used indirectly to find out the controller's parameters or it can be directly integrated into the control scheme in order to compensate for the dynamics and the nonlinearities of the system [1, 2]. Especially in this second situation having an accurate model, that can be kept updated while the operation of the robot, represents an important mechanism to avoid the degradation of the robot performances during time.

Hydraulic and pneumatic actuation systems are suitable for humanoid robotic applications due to their high power to weight ratio [3]. In comparison with electrical motor, they have the advantage of an higher power-to-weight ratio, a faster dynamics, and the possibility to be used in direct-drive mode. Nevertheless, they also present a nonlinear time-variant dynamic behavior that needs to be taken into account when designing the control system. Among the nonlinear phenomena that affect the hydraulic and pneumatic systems we have the saturation and the dead-band in the servo-valves [4, 5], the backlash and the friction of the actuator seal [6], the oil viscosity and flow characteristic through orifices and pipes.

Furthermore, deterioration of the mechanical parts (e.g. actuator sealing), change in the fluid temperature and viscosity, different load conditions,etc., demand for model-based control strategies that are capable to adapt to changes happening in the system.

Although most of the actuator's parameters necessary to formalize its mathematical model can be obtained from the product's data-sheet, in many cases it is necessary to identify them via ad-hoc and time-consuming experiments [7]. Moreover, it is often necessary to separate the subcomponents of the actuator (e.g. the electro-valve's solenoid/motor, its spool; the actuator's vane, its output shaft etc.) to access and measure the required variables.

A different way to represent the actuator is to consider it as a "black box" and use exclusively the input and output signals to identify its model [8]. This avoids to explicitly formalize the electromechanical characteristic

of the device and allows to easily integrate adaptation mechanisms in order to keep the model coherent with changes that may happen in the physical system [9].

The Wiener and the Hammerstein architectures represent an important class of block-oriented models that attracted particular attention in the control systems community [10, 11, 12, 13, 14]. This mainly due their structural simplicity and their suitability to represent a wide range of nonlinear dynamic systems. In the Hammerstein scheme (Fig. 1), the nonlinear static part is followed by the linear dynamic one, while in the Wiener scheme the two are inverted. In comparison with Hammerstein, the Wiener configuration brings more difficulties during the identification process. This is mainly due to the fact that the identification of the linear part depends on the estimation error of the the nonlinear parametric expression. However, if the nonlinear block is invertible, it is always possible to calculate the intermediate variable $n(t)$ and use it to identify the static part.

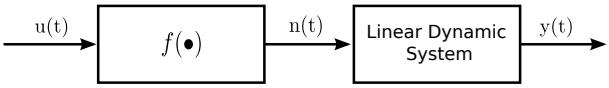

Figure 1: Hammerstein Scheme.

The model presented in this paper is based on the Hammerstein scheme. Although it is mainly intended to improve the control system of hydraulic and pneumatic actuation systems, it can also be applied to a wide range of applications where nonlinear dynamic systems need to be modeled and controlled. As novelty, the nonlinear memoryless part is based on a B-Spline Neural Network (BSNN) that encompasses a computationally low-expensive adaptation rule. The dynamic part is instead based on a classical Auto-Regressive eXogenous system (ARX) and it is identified using a Recursive Least Square algorithm (RLS).

Although b-spline functions were investigated as early as the nineteenth century, BSNNs attracted only recently the scientific community. They can be used as approximators of nonlinear continuous function [15] and applied for parameters identification and control of non linear actuation systems. In [16] and [17] a BSNN was used to estimated the nonlinear flux linkage

characteristic and the electromotive force of a switched reluctance motor by using position and current measurements as inputs. In [18] the flux linkage model identified by the BSNN was instead used to find the optimal design parameters of the same motor kind. A BSNN model was adopted in [19] to predict the torque and the speed of a permanent magnet synchronous motor and it was integrated with a PI controller in order to improve the transient response of the rotor speed. To estimate the nonlinear dynamics of a voice coil motor a BSNN was used in [20] coupled with a position controller, while in [21] the BSNN was trained online to minimize the current tracking error of a linear motor that actuate a reciprocating vapor compressor.

As an alternative to BSNNs classical Feed Forward Neural Networks (FFNN) based on the multilayer perceptron model can be used to identify the non linearities of a dynamic system. In [22] a FFNN was used to estimated the displacement of a hydraulic cylinder from pressure and flow rate measurements, while in [23] the displacement of a pneumatic piston was computed by the neural network from a tapped delay version of the voltage signal provided to the servo-valve connected to the actuator.

Once the input-output signals and the architecture of the BSNN or FFNN are defined different learning strategies and algorithms can be used to tune their parameters. In [16] and [21] the Least Mean Squares algorithm (LMS) was used to train the network with a computational complexity of $O(N^2)$ (N here represents the number of adapting parameter). In [18] the adaptation of the neural network's weights was based on the Levenberg-Marquardt (LM) algorithm which has even higher computational complexity, i.e. $O(N^3)$. In [20] to speed up the convergence of the tracking error a proportional-integral adaptation rule was adopted, while in [19] an efficient instantaneous local learning rule was used instead . Alternatively, in the case of applications that do not require to identify the model online, it is possible to use a classical batch back-propagation algorithm as was done in [22].

To tune the weights of the BSNN we propose a local instantaneous learning rule as in [19]. However, to avoid that the weights diverge and cause the system instability we also integrated a saturation mechanism. In comparison with the adaptation algorithms used in [16, 21, 18, 23] the algorithm we implemented presents a reduced computational complexity of $O(N)$. This, make it more suitable for online applications and in all the situations where the on-board computational power is limited. Furthermore, compared with the batch learning algorithm used in [22], which requires to use the entire input-output dataset different times (epochs) to guarantee the convergence

of the FFNN weights, the algorithm we implemented starts to adjust the BSNN weights from the first input-output sample available. By doing so it possible to design model based control systems that can steadily improve their performances from the moment they start to operate.

In [16, 22] the authors used a static nonlinear model to predict dynamic quantities. This, is possible by providing the BSNN with an input that is representative of the system state, i.e. the rotor angular position and the motor's current. The main drawback of this approach is represented by the fact that it is necessary to measure not only the position of the rotor, but also the absorbed current. The BSNN-ARX model we propose on the contrary is able to represent also the dynamic component of the system and therefore it is more general than a pure BSNN.

The rest of this document is organized as follow: Section 2 describes in detail the main blocks of the proposed BSNN-ARX model and indicates the necessary conditions to guarantee the convergence of the adaptation algorithm. Section 3 presents the model implementation and the experimental setup. Section 4 compares the performances of the combined BSNN-ARX architecture with the standalone ARX and BSNN models. Furthermore, it considers for comparison other two nonlinear dynamic models based on neural networks, the NNARX and the NNOE architectures. Finally, Section 5 draws the conclusions and gives indications about possible future research directions.

2. The B-spline Neural Network ARX Model

The two main blocks that compose the BSNN-ARX model are described in detail in the two following sections.

2.1. ***The linear dynamic part***

The linear dynamic part of the Hammerstein model proposed in this work is represented by an ARX system (see Eq. 1), where $n(t)$ and $y(t)$ are the input and the output of the system respectively, B and A are two polynomials in z^{-1} of degree m and n respectively which expressions are: $B(z^{-1}) = b_m z^{-m} + b_{m-1} z^{(-m+1)} + .. + b_1 z^{-1}$ and $A(z^{-1}) = a_n z^{-n} + a_{n-1} z^{n-1} + ... + a_1 z^{-1} + 1$, where $[b_1, .., b_m]^T \in \mathbb{R}^m$ and $[a_1, .., a_n]^T \in \mathbb{R}^n$. We assume that the system is realizable, this traduces in the condition $m \leq n$, thus the transfer function has a number of poles equal or bigger to the number of zeros.

$$y(t) = \frac{B(z^{-1})}{A(z^{-1})} n(t) \tag{1}$$

It is worth to mention here that an ARX model is not as general as an ARMAX or a state space model [24] e.g. expressed by Eq.2, where $x \in \mathbb{R}^n$, $y \in \mathbb{R}^p$, $u \in \mathbb{R}^m$, $A \in \mathbb{R}^{n \times n}$, $B \in \mathbb{R}^{n \times m}$, $C \in \mathbb{R}^{p \times n}$).

$$\begin{cases} x(k+1) = A\,x(k) + B\,u(k) \\ y(k) = C\,x(k) \end{cases} \tag{2}$$

Whereas an ARMAX and the state space model are equivalent, i.e. it is always possible to rewrite an ARMAX model into a state space form [25, 26], and vice-versa it is always possible, using the Cayley-Hamilton theorem, to eliminate the states from Eq. 2 and reach the ARMAX form; compared to an ARMAX the ARX model is not able to catch properly the noise dynamics present in the process, and therefore its performances degrades in case of a poor signal to noise ratio. However, a loss in generality can be justified by the fact that the structure of the parameters space of an ARX model is simpler if compared with that one of the ARMAX or a space-state model, and therefore more suitable for parameters identification methods [27] like the Recursive less Squares (RLS) algorithm. This is a very important especially if the algorithm has to be executed meanwhile the process is running.

To identify the ARX model a RLS algorithm is used, that is detailed described among others in [28]. Eq. 1 can be written, after elementary passages, as Eq. 3.

$$y(t) = -a_1 y(t-1) - ... - a_n y(t-n) + b_1 n(t-1) + ... + b_m n(t-m) \tag{3}$$

The estimated parameters are defined by the vector $\hat{\theta}(t-1)^T = [\hat{a}(t-1)^T, \hat{b}(t-1)^T]$ where $\hat{a}(t-1)^T = [\hat{a}_1(t-1), ..., \hat{a}_n(t-1)]$, and $\hat{b}(t-1)^T = [\hat{b}_1(t-1), ..., \hat{b}_m(t-1)]$. Considering the regressor vector $\phi(t-1)^T$ written as in Eq. 4 it is possible to rewrite Eq. 3 in a compact form at the time $t+1$ as a priori prediction (superscript index $-^o$) by Eq. 5 .

$$\phi(t-1)^T = [-y(t-1), ..., -y(t-n), n(t-1), ..., n(t-m)] \tag{4}$$

$$\hat{y}^o(t+1) = \hat{\theta}(t)^T \phi(t) \tag{5}$$

Assuming to know at each instant $t+1$ the value of the process output $y(t+1)$ it is possible to calculate the (a priori) prediction error $\varepsilon^o(t+1)$ using Eq. 6

$$\varepsilon^o(t+1) = y(t+1) - \hat{y}^o(t+1) \tag{6}$$

At this point the adaptation can be computed as Eq. 7,

$$\hat{\theta}(t+1) = \hat{\theta}(t) + F(t+1)\phi(t)\varepsilon^o(t+1) \tag{7}$$

where the adaptation gain is adapted according to Eq. 8.

$$F(t+1) = F(t) - \frac{F(t)\phi(t)\phi(t)^T F(t)}{1 + \phi(t)^T F(t)\phi(t)} \tag{8}$$

2.2. ***The nonlinear static part***

The memoryless nonlinear part used in the BSNN-ARX model consists of a B-spline neural network which adaptation mechanism is based on a error-driven learning rule. To the best of the author knowledge none of the Hammerstein models presented so far in the literature use such an adaptation paradigm. Most of the authors have considered as nonlinear static part for the Hammerstein model a polynomial of finite and know order [10]. Others have used fuzzy systems [12], neural networks [14], or piecewise linear functions [11]. Although B-spline were already used in [29] where the input variable of a process is transformed via a B-spline basis matrix and feed in a linear state-space model, and in [30, 31] where a single layer B-spline neural network is combined with an ARX model and together identified using a least square method; none of them consider to separate the identification of the static part from the dynamic one. The novelty of our approach consists in combining a classical RLS algorithm, to identify the parameters of the ARX model, with a local instantaneous learning mechanism to adapt the BSNN. This brings advantages in terms of a reduced computational complexity (see Section 3.2) and therefore an improved real-time adaptation capability.

BSNNs belong to the class of single layer Feed Forward Neural Networks (FFNN) and similarly to a Support Vector Machine (SVN) [32] can be used to define a mapping from an input $u(t)$ to an output $n(t)$ (Fig. 2) by using B-splines as activation functions. As a classical FFNN the signal propagates from the input toward the output without loops. However, as main difference, not the inputs of the neurons, but their outputs are weighted and superimposed to generate the outcome signal $n(t)$.

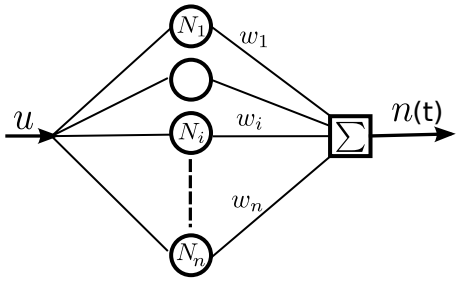

Figure 2: Architecture of a single input single output BSNN.

The B-spline activation functions are piecewise polynomials that, if compared with pure polynomials, have the advantages to better fit hard nonlinearities. This is possible thanks to their intrinsic local nature. Given a defined range $[u^{min}\, u^{max}]$ for the input signal $u(t)$, at first it is necessary to define an ordered set of knot points $\mathbf{K} = \{k_1, k_2, ..., k_p\}$ where $k_i \in \mathbb{R}$ and $u^{min} = k_1 < k_2 < ... < k_p = u^{max}$. These can be either equally distributed along the input signal range or located with higher frequency where the output signal $n(t) = f[u(t)]$ presents its highest non linearities.

In the developed model p equally distributed knot points and quadratic B-spline as activation function were considered. Each neuron N_i in the network is modeled by Eq. 9, where $k_i \in \mathbf{K} - \{k_{p-2}, k_{p-1}, k_p\}$, $\Delta k = (u^{max} - u^{min})/p$.

$$N_i(u) = \begin{cases} \frac{(u-k_i)^2}{2(\Delta k)^2} & k_i \leq u < k_i + \Delta k \\ \frac{(u-k_i)(k_i+2\Delta k-u)}{2(\Delta k)^2} + \frac{(k_i+3\Delta k-u)(u-k_i+\Delta k)}{2(\Delta k)^2} & k_i + \Delta k \leq u < k_i + 2\Delta k \\ \frac{(k_i+3\Delta k-u)^2}{2(\Delta k)^2} & k_i + 2\Delta k \leq u < k_i + 3\Delta k \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

The network output is calculated as Eq. 10, where the b represents a bias and that is adapted according to the same rule used to adapt w_i.

$$n(u) = \sum_{i=1}^{p-3} (w_i \cdot N_i(u)) + b \tag{10}$$

The specific architecture of a BSNN has a big impacts on the adaptation algorithm. Due to the fact that the nonlinearity is located before the synapses it is not necessary to compute the derivative of the activation function. In

particular the learning step, a modified version of the delta rule, is performed on a patter-by-pattern basis according to Eq.11 and Eq. 12, where η is the learning constant, and $\overline{n}$ and n are the target and actual output of the BSNN respectively.

$$w_i(t+1) = w_i(t) + \Delta w_i \tag{11}$$

$$\Delta w_i = \eta \cdot N_i(u) \cdot (\overline{n} - n) \cdot (1 - |w_i(t)|) \tag{12}$$

How it possible to notice Δw_i not only depends on the prediction error and the neuron's output, but also on the term $(1 - |w_i(t)|)$ that has the function to limit the weight in the range $(-1, +1)$ (i.e. a saturation mechanism).

The adaptation step can be computed for each single neuron separately, this has the advantage that the computational resources can be delocalized if required.

The described model can also be generalized to deal with multi inputs dynamic systems $\mathbf{u} \in \mathbb{R}^q$ (i.e. MISO systems). In order to do that a possible solution is to replicate q-times the architecture of Fig. 2 and superimpose the output NN_j of each single j^{th}-BSNN (see Fig. 3). The output of the neural network in this case can be computed using Eq.13, where $n = p - 3$.

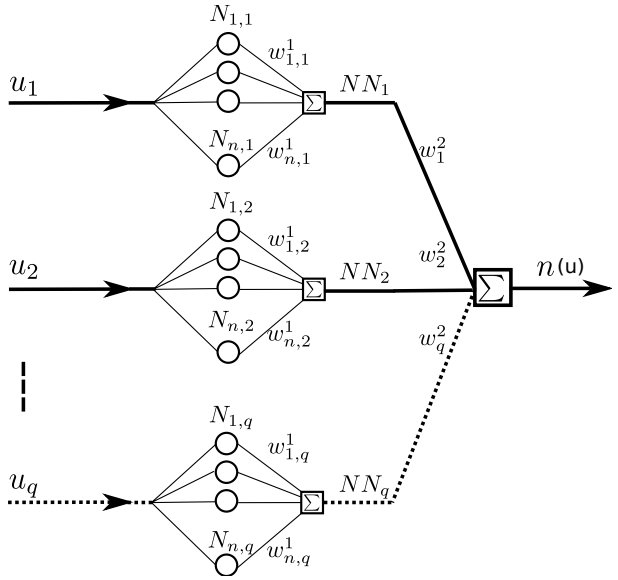

Figure 3: Architecture of a multi-input single-output BSNN

$$n(u) = \sum_{j=1}^{q}(w_j^2 \cdot \sum_{i=1}^{n} w_{i,j}^1 \cdot N_{i,j}(u_j)) \tag{13}$$

Due to the fact that each input u_j may have a different range of values, it is necessary to compute the knot-point distance separately for each BSNN, i.e. $\Delta k_j = (u_j^{max} - u_j^{min})/p$. Also the adaptation mechanism needs to be revised; the learning rule for the second layer in this case is expressed by equation 14

$$\Delta w_j^2 = \eta_2 \cdot NN_j \cdot (\overline{n} - n) \cdot (1 - |w_j^2(t)|) \tag{14}$$

where $j = 1, 2, .., q$. For the first layer, the adaptation mechanism is a bit more complicated, at first it is necessary to back propagate the error signal at each BSNN output, i.e. $\delta_j = w_j^2 \cdot (\overline{n} - n)$, then it is possible to calculate the adaptation step by Eq. 15, where $i = 1, 2, ..., n$ and $j = 1, 2, .., q$.

$$\Delta w_{i,j}^1 = \eta_{1,j} \cdot N_{i,j}(u_j) \cdot \delta_j \cdot (1 - |w_{i,j}^1(t)|) \tag{15}$$

2.3. ***Convergence of the BSNN Adaptation Algorithm***

The BSNN-ARX model has the well know Hammerstein architecture. In order to guarantee the convergence of the overall adaptation algorithm it is necessary to demonstrate that the prediction error relative to the BSNN part is decreasing over time, i.e. $|e(t+1) < |e(t)|$. From Eq. 10 we can rewrite the error in a matrix format as Eq. 16 where the adapting bias b is included considering an additional b-spline which generates a fixed output 1 and by adding a new synaptic weight.

$$e(t) = \overline{n}(t) - n(u(t)) = \overline{y}(t) - \mathbf{W}^T(t)\mathbf{N}(u(t)) \tag{16}$$

Representing the error as a Taylor series expansion stopped at the first order we can rewrite $e(t+1)$ as in Eq. 17.

$$e(t+1) = e(t) + \frac{\partial e(t)}{\partial \mathbf{W}^T(t)}\Delta\mathbf{W}(t) + H.O.T \tag{17}$$

By calculating the partial derivative of Eq. 16 and representing the weights update in Eq. 12 in a vector format we obtain

$$\frac{\partial e(t)}{\partial \mathbf{W}^T(t)} = -\mathbf{N}^T(u), \tag{18}$$

$$\Delta \mathbf{W}^T(t) = \eta e(t) \mathbf{N}(u) \odot (\mathbf{1} - |\mathbf{W}^T(t)|), \tag{19}$$

where the symbol $\odot$ represents a component wise multiplication of two column vectors, and $|\cdot|$ the component wise absolute value of a vector. By substituting equations 18 and 19 in Eq. 17 we obtain

$$e(t+1) = e(t)(1 - \eta \mathbf{N}^T(u)\mathbf{N}(u) \odot (\mathbf{1} - |\mathbf{W}^T(t)|)). \tag{20}$$

To ensure the convergence of the adaptation algorithm it should be verified that $|1 - \eta \mathbf{N}^T(u)\mathbf{N}(u) \odot (\mathbf{1} - |\mathbf{W}^T(t)|)| < 1$, this brings a constraint on the learning constant η as

$$0 < \eta < \frac{2}{\mathbf{N}^T(u)(\mathbf{N}(u) \odot (\mathbf{1} - |\mathbf{W}^T(t)|)}. \tag{21}$$

We can find a conservative value for η by calculating the minimum of the expression representing its upper bound limit (Eq. 21). By observing that it is always verified that $0 \leq (\mathbf{1} - |\mathbf{W}^T(t|) \leq 1$ and $\mathbf{0} \leq |\mathbf{N}(u)| \leq \mathbf{1}$, we can assure the convergence of the BSNN adaptation process if

$$0 < \eta < \frac{2}{n}, \tag{22}$$

where n represents the number of splines used in the model.

3. Model Implementation and Experimental Setup

The combined BSNN-ARX model was implemented partially as matlab code and partially as simulink code (see figure 4). In particular, the parameters initialization, the validation and the models comparison phases were performed by matlab scripts, while the training phase was carried out by Simulink simulations. On the one hand, Simulink's GUI (Graphical User Interface) brings the advantage to monitor and tune all the system parameter while executing the simulation. On the other hand, the matlab scripts facilitate the implementation of different simulations in batch modality (e.g. different instances of the BSNN-ARX model having different initial conditions).

In the experimental setup the input-output pairs series $\{u_t, y_t\}_{t=1}^{T}$, where T represents the total number of samples, was generated with a Silverbox

model. The time series was then divided in two sets, i.e. the training set and the validation set consisting of T_T and T_V samples respectively.

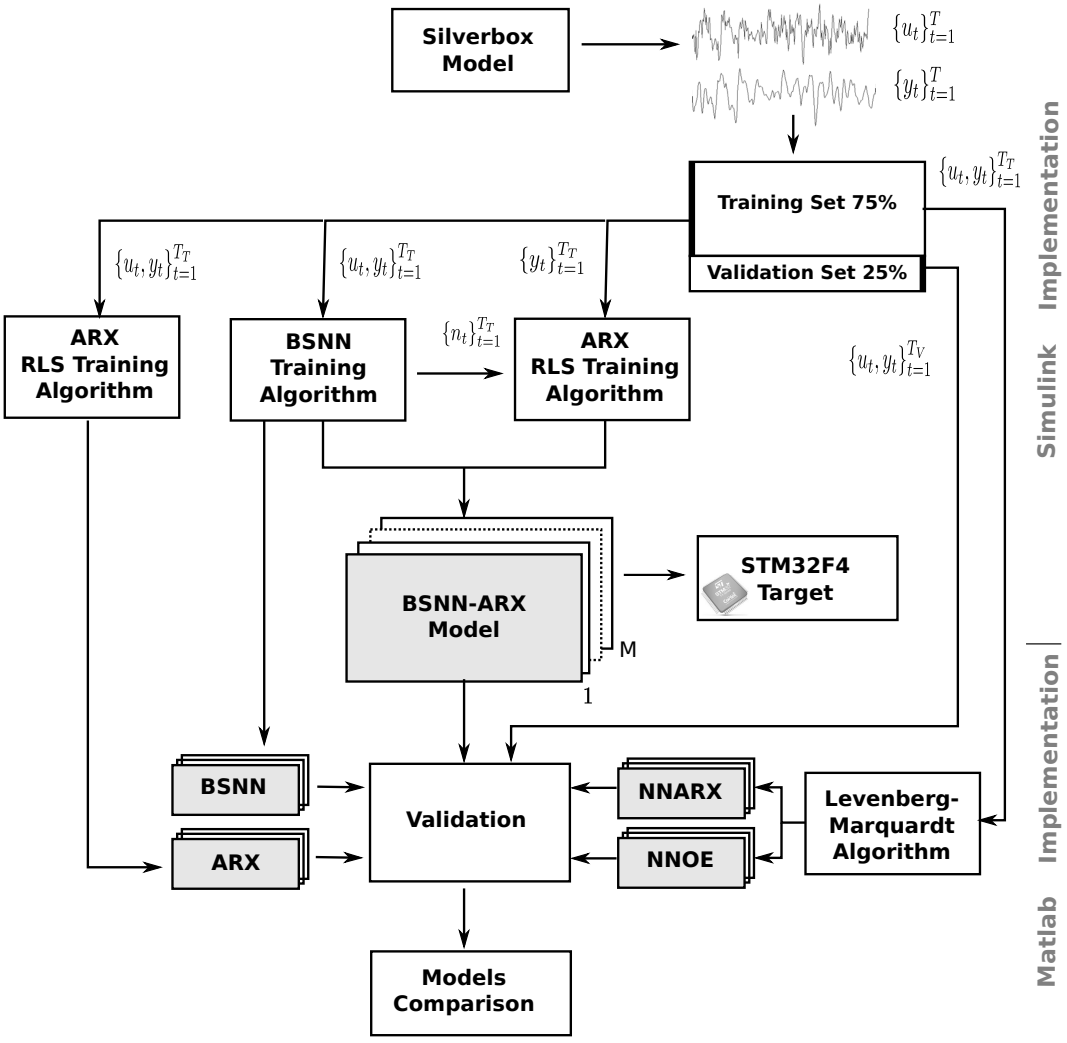

Figure 4: Schema representing the experimental setup used for the BSNN-ARX model training and validation. In gray color are reported the different models considered in the comparison.

The training of the BSNN and ARX models were carried out simultaneously. However, while the BSNN training algorithm (see figure 4) receives as input the complete input-output series $\{u_t, y_t\}_{t=1}^{T_T}$, the ARX RLS training

algorithm receives only the output part $\{y_t\}_{t=1}^{T_T}$ and uses instead as reference input the signal generated by the BSNN, i.e. $\{n_t\}_{t=1}^{T_T}$. Furthermore, for comparison purposes a second instance of the ARX model was included in the simulink code which is trained with the same input-output pairs as the overall BSNN-ARX model.

To allow evaluating the statistical significance of the model, M instances of the BSNN-ARX (having each a different weights initialization) are sequentially trained and their performances evaluated separately. Finally a comparison with alternative models is performed in terms of different statistical quantities.

The schema represented in figure 4 includes also a branch which represents a possible practical implementation of the BSNN-ARX model and its adaptation algorithm. A quite straightforward real-time realization can be obtained by using additional Simulink packages that can be installed for most of the commercially available rapid prototyping boards, e.g. STM32F4 Discovery board, Raspberry Pi 2 board, BeagleBone board, etc. Generally, each package includes specific Simulink blocks to configure and access the device's peripherals (Digital I/O, ADC, PWM, etc.) without requiring a deep knowledge of the hardware.

3.1. ***The Silverbox Benchmark and the Model Parameters***

As a benchmark to validate the model we used a Silverbox process fed with filtered Gaussian noise. In particular, the second order differential equation, Eq. 23, can be interpreted as a mass-spring-damper system where $y(t)$ represents the displacement of the mass m, $u(t)$ an exciting force, b the damping constant, and k_1 and k_2 the two elastic constants of the spring's linear and nonlinear components respectively.

$$m\ddot{y}(t) + b\dot{y}(t) + k_1 y(t) + k_2 y^3(t) sin^2 y(t) = u(t) \tag{23}$$

As an example of physical device behaving according to Eq. 23 we may consider the actuation system of the robotic leg represented in figure 5, where in this case the non linear element is represented by the spring connected in parallel with the damper and the hydraulic actuator.

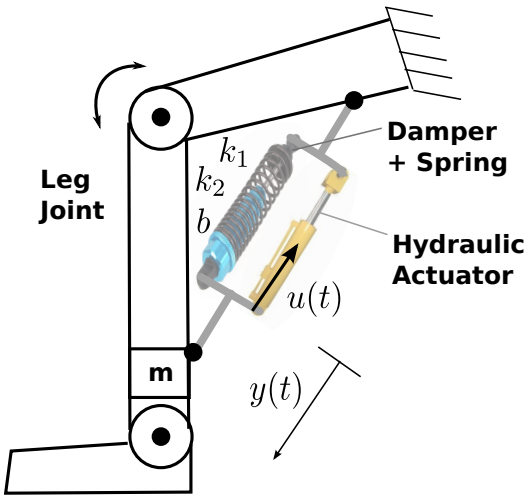

Figure 5: Example of actuation system for a humanoid robot leg. The spring, the damper and the hydraulic actuator are connected in parallel.

In our experimental setup the input signal $u(t)$ consists of a filtered Gaussian noise having a probability density distribution with standard deviation $\sigma^2 = 0.07$ and mean value $\mu = 0.7$. In particular, a first order low-pass filter was used with a cut-off frequency of $16Hz$.

The implemented BSNN-ARX (see Fig. 6) consists of a single layer B-spline neural network having 8 neurons connected in cascade with an ARX model with dimension $n = 3$ and $m = 3$ (see section 2.1). The activation function of each neuron is represented by a B-spline as in Eq. 9 having parameters $u^{min} = 0$ and $u^{max} = 0.8$.

The weights and bias of the BSNN are initialized to uniformly distributed random numbers within the range $W^1_{i,j}(t_0), b_i \in [-0.25, +0.25]$, while the learning constant $\gamma_2 = 0.004$.

The first 75% of the samples are used for adaptation while the last 25% for validation. Overall the system runs for $10s$ with a sampling time of $0.005s$ and generates a total of 2000 input-output pairs. The adaptation process takes place only one time. This brings considerable advantages compared with other algorithms that can work only in batch modality. For these algorithms the same input-output pairs set is presented to the network multiple times (epochs) making the process computationally expensive and not suitable for online applications.

Although to speed up the adaptation process it is generally advisable to

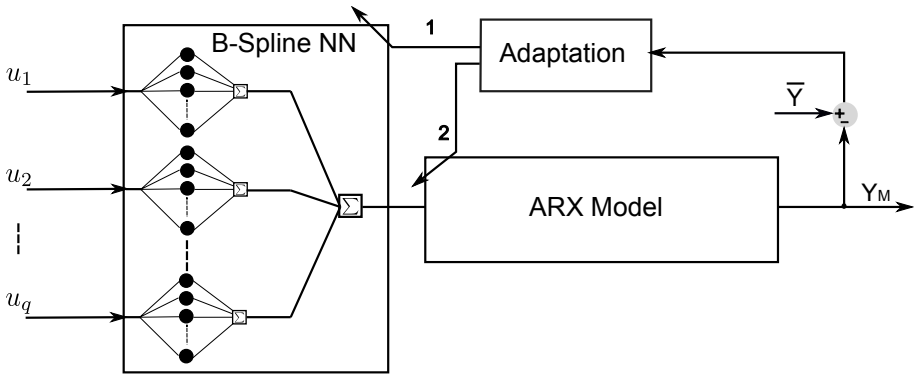

Figure 6: The BSNN-ARX model architecture.

remove the mean and to normalize the signal [33], this preprocessing here is avoided in order to cope with a more realistic situation where the range of the signal is not known a priori and the amount of computation available is limited.

3.2. ***The parameters identification algorithm and its complexity***

In order to identify the parameters of both BSNN and the ARX models a combination of the algorithms described in Sections 2.1 and 2.2 is applied. The overall BSNN-ARX model is depicted in Figure 6 and the exact sequence of the adaption steps is resumed in the Algorithm 1.

The adaptation goes on for the entire training time, i.e. $t \leq t_L$, after the adaptation-module stops to operate and only the forward step of the model is computed on the base of the last instance of the parameters (lines 12 and 13).

If the model needs to be adapted online it is crucial to estimate the computational complexity of the adaptation phase; this in order to choose a suitable hardware. In first approximation we can evaluate it counting the number of multiplications required at each adaptation step both for the ARX and BSNN part. If we consider that, with the most simple algorithm, multiplying two vectors of dimension N and multiplying a matrix and a vector of dimensions $N \times N$ and N requires N and N^2 multiplications respectively, the RLS algorithm with N parameters requires a total of $5 \cdot N^2 + 3 \cdot N$ multiplications and therefore its complexity is $O(N^2)$. On the other hand the adaptation step of a single input BSNN having N neurons requires $5 \cdot N$

Algorithm 1 The adaptation algorithm

Initialize the BSNN and ARX parameters: $w_{i,j}^k(t_0)$, $\hat{\theta}(t_0)$, b_i
$t \leftarrow t_0$
while $t \leq t_L$ **do**
    Compute the BSNN output Y_{BSNN} using $w_{i,j}^k(t)$
    Compute the ARX model output using $\hat{\theta}(t)$
    Calculate the model error: $\varepsilon^o(t+1) = \overline{Y}(t+1) - Y_M(t+1)$
    Update the ARX parameters: $\hat{\theta}(t+1) = \hat{\theta}(t) + F(t+1)\phi(t)\varepsilon^o(t+1)$
    Update weights and bias of the BSNN: $w_{i,j}^k(t+1)$, b_i
    $t \leftarrow t+1$
end while
while $t \leq t_T$ **do**
    Compute the BSNN output Y_{BSNN} using $w_{i,j}^k(t_L)$
    Compute the ARX model output using $\hat{\theta}(t_L)$
    $t \leftarrow t+1$
end while

multiplications, therefore its complexity increases linearly with the number of parameters, $O(N)$. It is clear at this point that if we split the total number of parameters of our model in $N1$ for the ARX part and $N2$ for the BSNN part, a pure RLS adaptation algorithm would require a number of multiplications in the order of $(N1+N2)^2$, while our combined algorithm $(N1)^2+N2$. This brings considerable advantages when the model is integrated within a control loop that has strict time constraints, especially if it is implemented on small computational units like a micro-controller or mini calculator (e.g. STM32F4, Raspberry Pi 2).

4. Experimental Results

Before testing the combined BSNN-ARX model the single BSNN and ARX models were tested separately with the same data set. The adaptation of the BSNN's weights can be observed in Figure 7, a similar trend can also be observed for the adaption of the neurons' threshold (bias). How it is possible to notice from the topside plot of Fig. 7 the weights are converging toward a stable value after $300 samples$ (1.5s). At this point the signal predicted by the BSNN is tracking the process output (bottom plot of Figure 7).

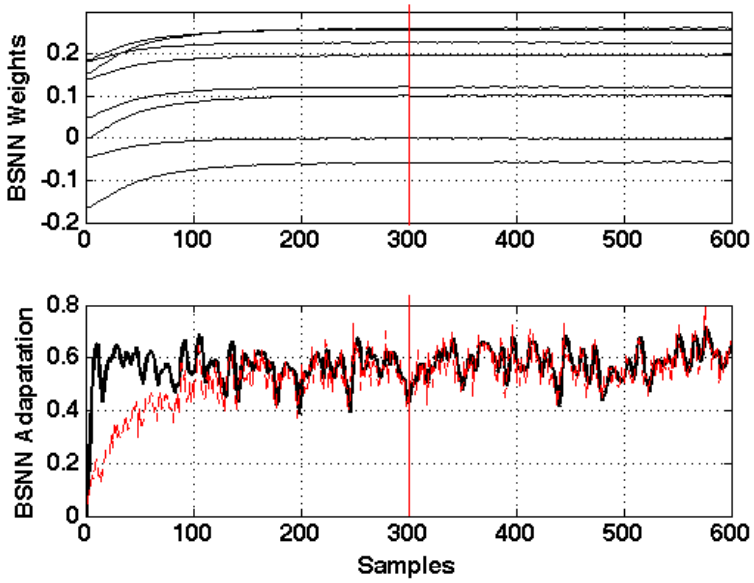

Figure 7: Weights adaptation of the BSNN.

The top side of Fig. 8 shows the process's output (black line) and the prediction performed by the BSNN model (red dashed line) on the validation set. How it is possible to notice the signal generated by the BSNN overshoots the reference signal and presents very narrow picks. This, is typical for a memoryless model where only an algebraic relationship between input and output can be established. The bottom side of Fig. 8 reports the residual at each sample as the difference between the process output and the prediction.

Figure 9 reports the output signal and the residual of the ARX model when fed with the same input of the process. As it is possible to notice, in comparison with the BSNN, the ARX model performs better. Although the predictions overshot the reference, in this case the output signal presents a smoother trend. Overall the error is still too high and the performances not satisfactory. This is clear considering the fact that the dynamic model we want to identify is nonlinear and therefore can not be properly represented by a pure ARX model.

As a second step the overall BSNN-ARX was tested. As main difference now the ARX model does not receive, as input, the process input, but instead the output generated by the BSNN (Fig. 4). While in this last case the adap-

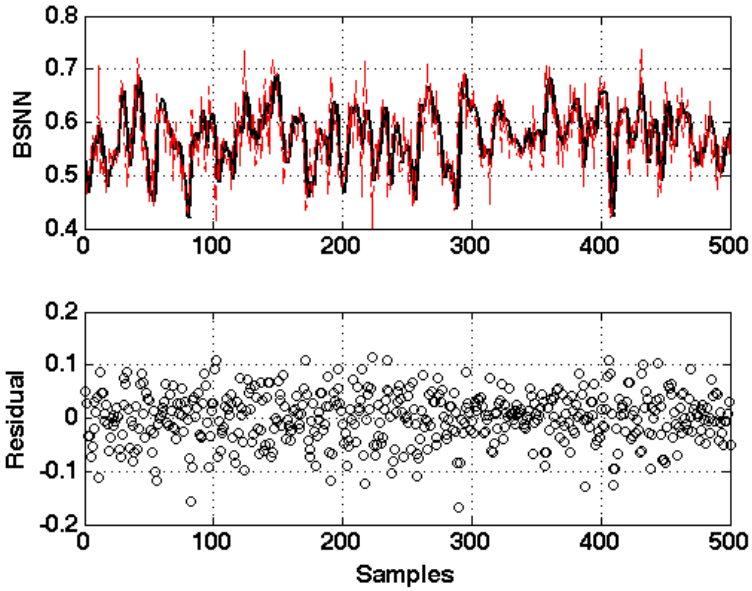

Figure 8: Performances on the validation set for the BSNN with a single layer of 8 neurons.

tation of the BSNN is not effected by the new architecture, the adaptation of the ARX model depends on the adaptation of the BSNN. This is clearly visible from the plots of Fig. 10 where it is possible to notice that in the second case (bottom graph) the adaptation of the vector $\hat{\theta}(t)$ requires more time (1500 samples). The predictions of an exemplary BSNN-ARX model on the test set are reported in Fig. 11. How it is possible to notice now the combined model is predicting the target signal with higher accuracy if compared with a pure BSNN or ARX model.

To increase the statistical significance of our results [34, 35] the adaptation process of the BSNN-ARX, BSNN, and ARX models were repeated $M = 100$ times. In each simulation instance the weights of the BSNN and the ARX's parameters were reinitialized according to the procedure explained in section 3.1. Figure 12 reports the RMS error computed using the validation set for each of the BSNN-ARX models, where the blue line represents the mean of all the 100 values.

Comparing the three models in terms of $mean\{\mathbf{RMS_i}\}_{\mathbf{i=1}}^{\mathbf{M}}$ and $var\{\mathbf{RMS_i}\}_{\mathbf{i=1}}^{\mathbf{M}}$, calculated with the validation set, we can confirm the superiority of the

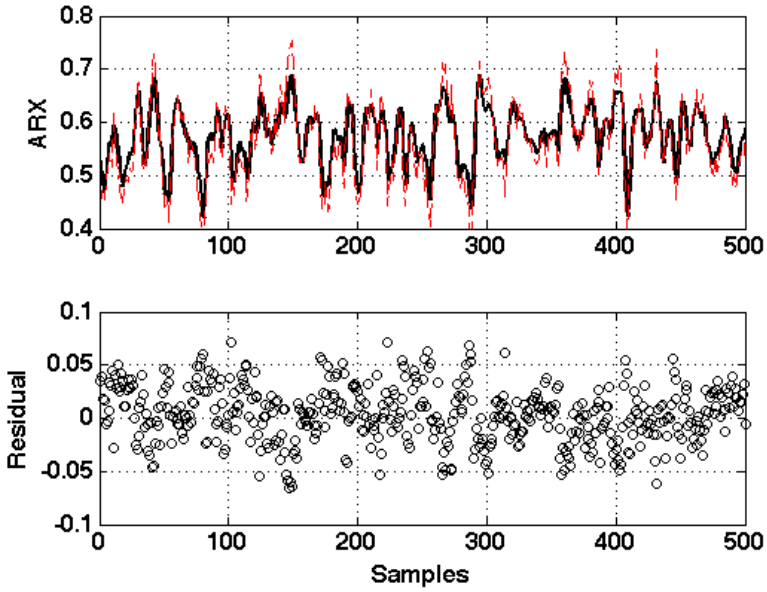

Figure 9: Performances on the validation set for the ARX model having n=3 and m=3.

BSNN-ARX model. The value of the $mean\{\mathbf{RMS_i}\}_{\mathbf{i=1}}^{\mathbf{M}}$ passes from 0.0479 for the BSNN model and 0.0174 for the ARX model to 0.0116 for the BSNN-ARX model, while the $var\{\mathbf{RMS_i}\}_{\mathbf{i=1}}^{\mathbf{M}}$ passes from $8.26e^{-8}$ for the BSNN model and $1.91e^{-7}$ for the ARX model to $2.17e^{-6}$ for the BSNN-ARX model.

As last step the BSNN-ARX model was compared with other two predictors having similar complexity, the NNOE and the NNARX architectures. These models (see Fig. 15) consist of a FFNN having two or more layers of neurons and as inputs the full regression vector $\phi(t-1)^T$ as in equation 4. However, if in the NNARX model the output of the real process composes the regression vector in case of a NNOE model the output of the predictor is feedback to the input layer to form the regression vector. To train the NNOE and NNARX model the Levenberg-Marquardt algorithm was used with a total of 10 iterations.

Figure 16 reports the predictions and the residual relative to an exemplary NNOE model having an input layer of dimension 6, a hidden layer of dimension two, and an output layer of dimension one. In total the model has 14 adjustable parameters (synapses) as the BSNN-ARX model. The neurons

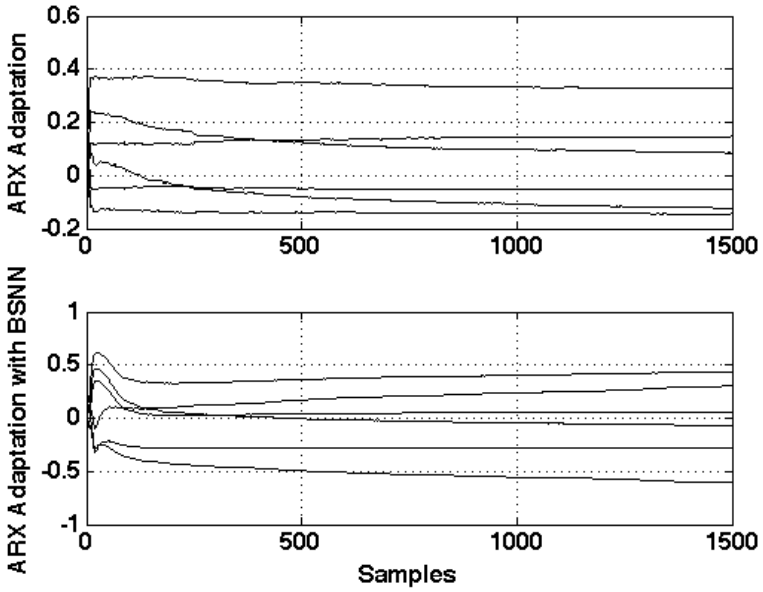

Figure 10: Adaptation of the ARX model as stand alone (upper plot) and combined with the BSNN model (bottom plot).

of the hidden layer have a sigmoidal activation function while the neurons of the output layer present a linear activation function. As for the BSNN-ARX model we computed the $mean\{\mathbf{RMS_i}\}_{\mathbf{i=1}}^{\mathbf{M}}$ and $var\{\mathbf{RMS_i}\}_{\mathbf{i=1}}^{\mathbf{M}}$ over 100 instances of the NNOE model obtaining the values 0.0133 and $6.02e^{-8}$ respectively. In terms of $mean\{\mathbf{RMS_i}\}_{\mathbf{i=1}}^{\mathbf{M}}$ the NNOE model performs worse than the BSNN-ARX model (0.0133 vs. 0.0116), however it has a much smaller variance ($6.02e^{-8}$ vs. $2.17e^{-6}$). This can be explained by the fact that while the adaptation process for the BSNN-ARX model is performed presenting the training set only one time, the Levemberg-Marquardt algorithm requires more iterations to train the NNOE model.

The predictions of an exemplary NNARX model are reported in Fig. 17. Considering all the 100 instances in this case we obtained a $mean\{\mathbf{RMS_i}\}_{\mathbf{i=1}}^{\mathbf{M}} = 0.0137$ and a $var\{\mathbf{RMS_i}\}_{\mathbf{i=1}}^{\mathbf{M}} = 1.06e^{-7}$. Its performances are therefore similar to that one of a NNOE model and slightly worse if compared, in terms of RMS errors, with a BSNN-ARX model.

Figure 13 represents the three models in terms of RMS errors reporting for each model the median value together with the 25th and the 75th

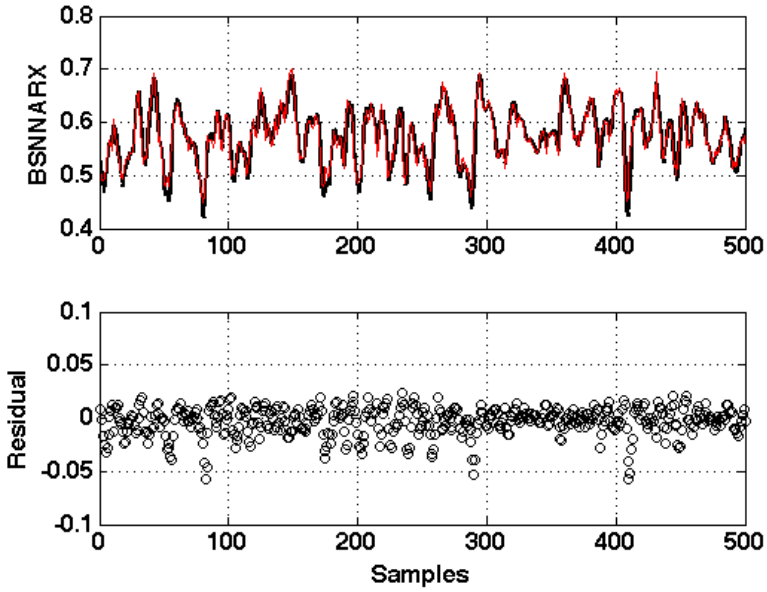

Figure 11: Performances using the validation set for the BSNNARX model.

percentile, the lower and upper adjacent and eventual outliers (calculated over 100 instances for each model kind). While Fig. 14 reports the same quantities calculated for the R^2 value.

Table 1 finally shows the mean and the variance for the $\{\mathbf{RMS_i}\}_{\mathbf{i=1}}^{\mathbf{M}}$ errors and $\{\mathbf{R_i^2}\}_{\mathbf{i=1}}^{\mathbf{M}}$ values, this for all the five models we considered (i.e. BSNN, ARX, BSNN-ARX, NNARX and NNOE). Furthermore, all the quantities are calculated for both the training and the validation sets. It is possible to observe that for the first three models the performances on the validation set are better than the one on the training set, while for the last two the performances are comparable. This is due to the fact that for the BSNN, ARX and BSNN-ARX models the adaption process is instantaneous and the residuals are calculated during the adaptation, while for the NNARX and NNOE models the adaptation process is of a batch kind and the residuals are calculated only once the model is completely adapted.

Considering only the validation set it is possible to notice that we have comparable performances for the BSNN-ARX and the NNARX and NSSIF models. However, it is also crucial to consider the computational complex-

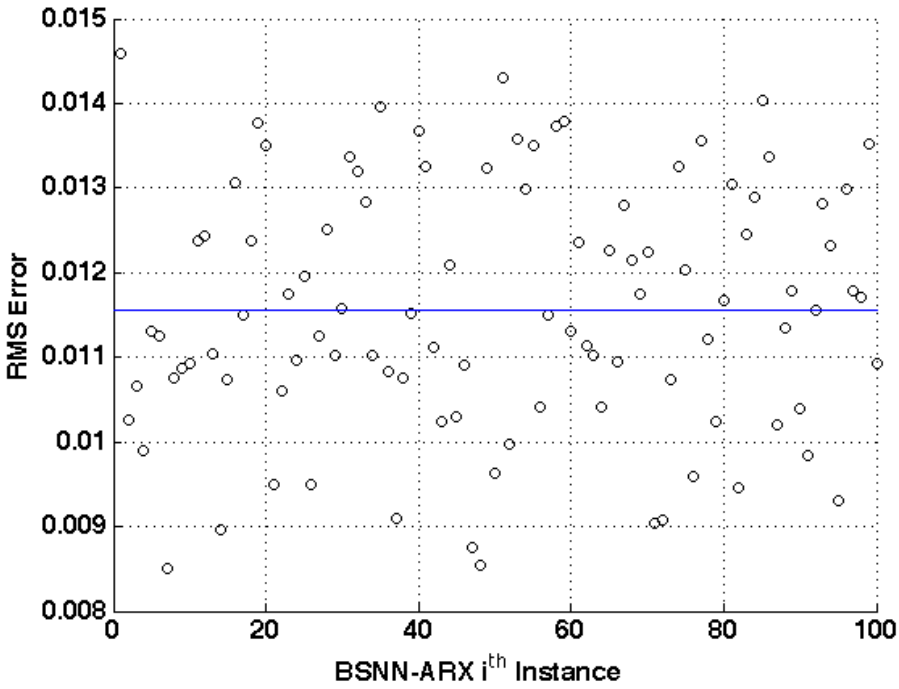

Figure 12: RMS error reported for each of the 100 BSNN-ARX model instances.

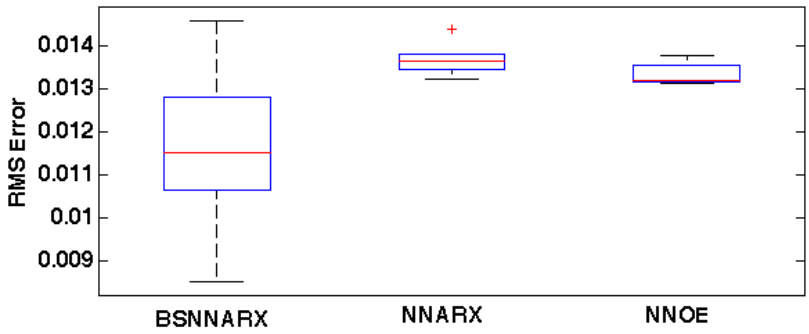

Figure 13: Comparison of the three models by considering the RMS error relative to the validation set. For each model is reported the median value together with the 25th and the 75th percentile, the lower and upper adjacent and the eventual outliers

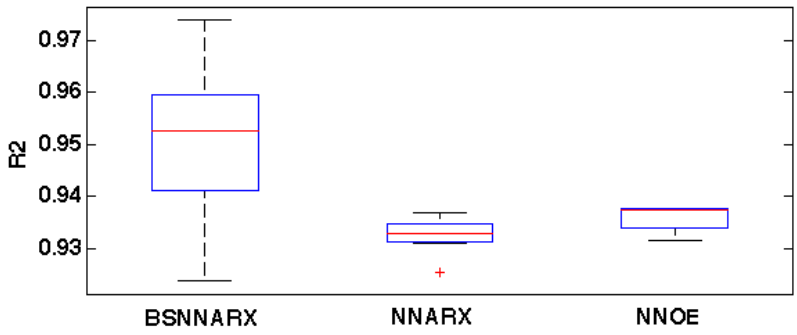

Figure 14: Comparison of the three models by considering the R^2 value relative to the validation set. For each model is reported the median value together with the 25th and the 75th percentile, the lower and upper adjacent and the eventual outliers

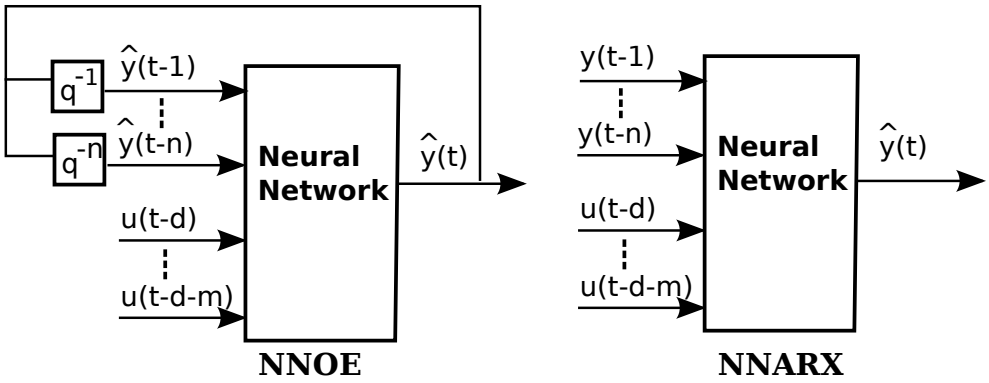

Figure 15: The NNOE and NNARX model structures.

ity of the learning algorithms they use. If the complexity of the combined algorithm used to train the BSNN-ARX model was calculated in section 3.2 as $O(N^2)$, the complexity of the the Levenberg-Marquardt (LM) algorithm, used to train the NNARX and NSSIF models, is estimated in $O(N^3)$. Due to the fact that it is necessary to compute the inverse of Hessian matrix at each weight update, often the LM algorithm is used off-line in the batch version. Online iterative implementations are still prohibitive especially for models with a large number of parameters, even though recently there were attempts to improve its computation [36] and to implement it by using ded-

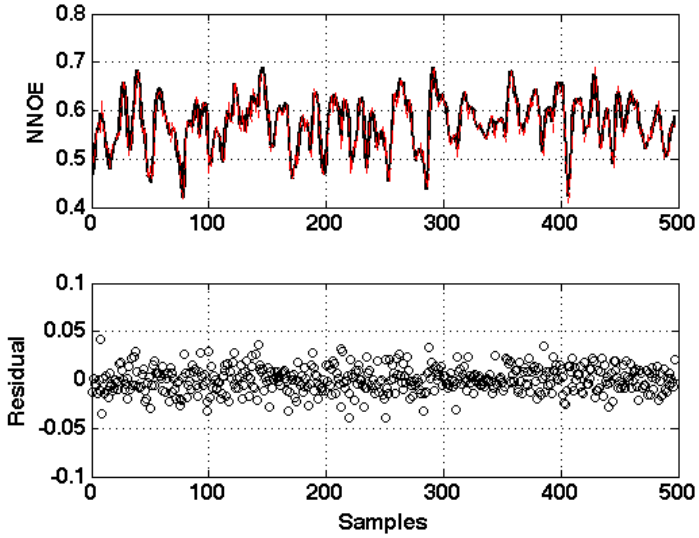

Figure 16: Performances using the validation set for the NNOE model.

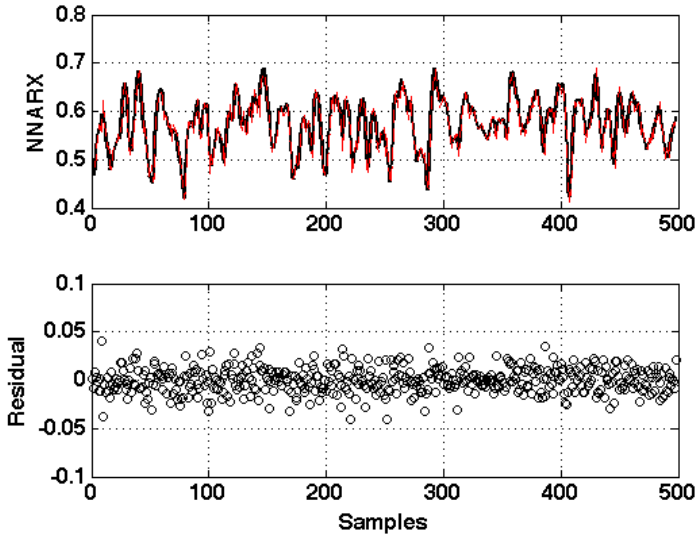

Figure 17: Performances using the validation set for the NNARX model.

Table 1: Comparison between the different models in terms of mean and variances of the RMS and R^2 values computed for all the 100 models instances. Data are reported for both the training and validation sets, NS stands for non significant.

TRAINING-SET	$\{\mathbf{RMS_i}\}_{\mathbf{i=1}}^{\mathbf{M}}$		$\{\mathbf{R_i^2}\}_{\mathbf{i=1}}^{\mathbf{M}}$	
	mean	**var**	**mean**	**var**
BSNN	0.1150	2.40e-06	NS	NS
ARX	0.0232	6.69e-06	0.8539	1.02e-03
BSNN-ARX	0.0233	7.49e-06	0.8526	1.09e-03
NNARX	0.0137	1.50e-07	0.9501	8.29e-06
NNOE	0.0133	9.07e-08	0.9528	4.68e-06
VALIDATION-SET	$\{\mathbf{RMS_i}\}_{\mathbf{i=1}}^{\mathbf{M}}$		$\{\mathbf{R_i^2}\}_{\mathbf{i=1}}^{\mathbf{M}}$	
	mean	**var**	**mean**	**var**
BSNN	0.0479	8.26e-08	0.1768	9.78e-05
ARX	0.0174	1.91e-07	0.8910	3.01e-05
BSNN-ARX	0.0116	2.17e-06	0.9513	1.48e-04
NNARX	0.0137	1.06e-07	0.9326	1.05e-05
NNOE	0.0133	6.02e-08	0.9360	5.65e-06

icated hardware [37].

5. Conclusions

A new Hammerstein architecture that combines a B-spline neural network and an ARX model suitable for nonlinear dynamic models identification is presented. The BSNN-ARX model is intended to be integrated in a feedback control scheme to compensate the nonlinear and time variant dynamic behavior of hydraulic and pneumatic actuation systems suitable for humanoid robotic applications. The learning process of the nonlinear static part and the linear dynamic one is taking place simultaneously based on a error-driven learning rule and the recursive least square algorithm respectively. The B-spline neural network consists of a single layer of neural units represented by a quadratic B-spline which outputs are linearly combined by a set of adaptive synapses. Its architecture can be easily generalized to identify multi-input dynamic systems by replicating the BSNN structure and introducing a two steps adaptation rule.

The implemented BSNN, in comparison with a classical FFNN adapted using conventional training techniques (e.g. back propagation, LM, LS, RLS) [38], is able to identify nonlinear mappings much faster due to the local and instantaneous nature of its weights adaptation mechanism.

The BSNN-ARX model was validated with a silver box benchmark excited with a filtered Gaussian noise. In comparison with a pure ARX or BSNN model it performs better in terms of the RMS error, while the comparison with alternative nonlinear dynamic models based on neural networks, i.e. the NNARX and NNOE architectures, shows similar performances. Furthermore, its lower computational complexity, linear in the number of the parameters for the memoryless part and quadratic for the dynamic part, makes it more suitable for online implementations.

Future work have to be dedicated to validate the proposed model in a real setup in order to demonstrate its efficacy to identify online the model of an hydraulic or pneumatic actuation system. It will be interesting to test its capability to adapt and to compensate for changes happening in the controlled system. In this context it will be necessary to implement a robust mechanism to tune the learning constants that allow a fast adaptation when the model is engaged for the first time and a slower one when it is required only to track slow changes in the dynamic system.

Acknowledgments

This work was supported by the Ministry of Education and Science of the Republic of Kazakhstan under the grant and target funding scheme (Project Title: "Development of a humanoid robot with neuromorphic control system for applications in household and public environments", agreement #220/073-2015). The author would like also to thank M.Sc Luis Manuel Vaca Benítez for his suggestions and support.

References

[1] H. Abdellatif, B. Heimann, Advanced model-based control of a 6-dof hexapod robot: A case study, Mechatronics, IEEE/ASME Transactions on 15 (2) (2010) 269–279. doi:10.1109/TMECH.2009.2024682.

[2] S. J. Yoo, J. B. Park, Y. H. Choi, Adaptive output feedback control of flexible-joint robots using neural networks: Dynamic surface design

approach, Neural Networks, IEEE Transactions on 19 (10) (2008) 1712–1726. doi:10.1109/TNN.2008.2001266.

[3] M. Folgheraiter, G. Gini, Human-like reflex control for an artificial hand, BioSystem Journal 76 (1-3) (2004) 65–74.

[4] A. Mohanty, B. Yao, Integrated direct/indirect adaptive robust control of hydraulic manipulators with valve deadband, IEEE/ASME Transactions on Mechatronics 16 (4) (2011) 707 –715.

[5] F. Rovira-Más, Q. Zhang, A. C. Hansen, Dynamic behavior of an electro-hydraulic valve: Typology of characteristic curves, Mechatronics 17 (10) (2007) 551–561.

[6] L. Heyns, J. Kruger, Describing function-based analysis of a nonlinear hydraulic transmission line, IEEE Transactions on Control Systems Technology 2 (1) (1994) 31 –35.

[7] M. Khoshzaban Zavarehi, P. Lawrence, F. Sassani, Nonlinear modeling and validation of solenoid-controlled pilot-operated servovalves, IEEE/ASME Transactions on Mechatronics 4 (3) (1999) 324 –334.

[8] M. Folgheraiter, M. Jordan, L. V. Benítez, F. Grimminger, S. Schmidt, J. Albiez, A highly integrated low pressure fluid servo-valve for applications in wearable robotic systems, in: International Conference on Informatics in Control, Automation and Robotics (ICINCO-2010), June 15-18, Madeira, Portugal, o.A., 2010.

[9] R.-J. Wai, P.-C. Chen, Robust neural-fuzzy-network control for robot manipulator including actuator dynamics, Industrial Electronics, IEEE Transactions on 53 (4) (2006) 1328–1349. doi:10.1109/TIE.2006.878297.

[10] W. Greblicki, Nonparametric identification of wiener systems by orthogonal series, IEEE Transactions on Automatic Control 39 (1994) 2077–2086.

[11] H.-F. Chen, Recursive identification for wiener model with discontinuous pice-wise linear function, IEEE Transactions on Automatic Control 51 (2006) 390–400.

[12] R. Liutkevicius, Fuzzy hammerstein model of nonlinear plant, Nonlinear Analysis: Modelling and Control 13 (2) (2008) 201–212.

[13] A. Balestrino, A. Landi, M. Ould-Zmirli, L. Sani, Automatic nonlinear auto-tuning method for hammerstein modelling of electrical drives, IEEE Transactions on Industrial Electronics 48 (3) (2001) 645655.

[14] Y. Fang, T. W. S. Chow, Orthogonal wavelet neural networks applying to identification of wiener model, IEEE Transactions on Circuits and Systems- I: Fundamental Theory and Applications 47 (2000) 591–593.

[15] S. H. Lane, M. Flax, D. Handelman, J. Gelfand, Multi-layer perceptrons with b-spline receptive field functions, in: R. Lippmann, J. Moody, D. Touretzky (Eds.), Advances in Neural Information Processing Systems 3, Morgan-Kaufmann, 1991, pp. 684–692.

[16] Z. Lin, D. Reay, B. Williams, X. He, Online modeling for switched reluctance motors using b-spline neural networks, Industrial Electronics, IEEE Transactions on 54 (6) (2007) 3317–3322. doi:10.1109/TIE.2007.904009.

[17] Z. Lin, D. Reay, B. Williams, X. He, High-performance current control for switched reluctance motors based on on-line estimated parameters, Electric Power Applications, IET 4 (1) (2010) 67–74. doi:10.1049/iet-epa.2009.0016.

[18] A. Kechroud, J. Paulides, E. Lomonova, B-spline neural network approach to inverse problems in switched reluctance motor optimal design, Magnetics, IEEE Transactions on 47 (10) (2011) 4179–4182. doi:10.1109/TMAG.2011.2151183.

[19] O. Aguilar Mejia, G. Tellez R, R. Tapia O, J. Templos S, Adaptive controller method for permanent magnet synchronous motor speed-regulation, in: Power Electronics, Machines and Drives (PEMD 2014), 7th IET International Conference on, 2014, pp. 1–6. doi:10.1049/cp.2014.0509.

[20] C.-F. Hsu, Y.-C. Chen, Microcontroller-based b-spline neural position control for voice coil motors, Industrial Electronics, IEEE Transactions on 62 (9) (2015) 5644–5654. doi:10.1109/TIE.2015.2416347.

[21] Z. Lin, J. Wang, D. Howe, A learning feed-forward current controller for linear reciprocating vapor compressors, Industrial Electronics, IEEE Transactions on 58 (8) (2011) 3383–3390. doi:10.1109/TIE.2010.2089948.

[22] H. Al-Assadi, M. Hayawi, A. Mat Ias, A. Jaffar, A. Omar, Adaptive learning controlling algorithm for a hydraulic servosystem, in: Mechatronics and Machine Vision in Practice (M2VIP), 2012 19th International Conference, 2012, pp. 168–172.

[23] N. T. Trung, D. Q. Truong, K. K. Ahn, Identification of a pneumatic actuator using non-linear black-box model, in: Control, Automation and Systems (ICCAS), 2011 11th International Conference on, 2011, pp. 1576–1581.

[24] R. E. Kalman, A new approach to linear filtering and prediction problems, Journal Of Basic Engineering 82 (Series D) (1960) 35–45.

[25] A. Lindquist, Comments on "canonical matrix fraction and state-space descriptions for deterministic and stochastic linear systems", Automatic Control, IEEE Transactions on 22 (2) (1977) 286 – 288.

[26] V. Popov, Some properties of the control systems with irreducible matrix transfer functions, in: J. Yorke (Ed.), Seminar on Differential Equations and Dynamical Systems, II, Vol. 144 of Lecture Notes in Mathematics, Springer Berlin / Heidelberg, 1970, pp. 169–180.

[27] A. Zeileis, Detecting longitudinal heterogeneity in generalized linear models, in: A. Taudes (Ed.), Adaptive Information Systems and Modelling in Economics and Management Science, Springer-Verlag, Wien, 2005, pp. 253–259, ISBN 3-211-20684-1.

[28] I. Landau, G. Zito, Digital Control Systems - Design, Identification and Implementation, Springer, 2006.

[29] J. Ramos, J.-F. Durand, Identification of nonlinear systems using a b-splines parametric subspace approach, in: American Control Conference, 1999. Proceedings of the 1999, Vol. 6, 1999, pp. 3955 –3959 vol.6.

[30] X. Hong, R. Mitchell, S. Chen, Modelling and control of hammerstein system using b-spline approximation and the inverse of de boor

algorithm, International Journal of Systems Science (2011) 1–9doi:10.1080/00207721.2011.564320.

[31] X. Hong, S. Chen, The system identification and control of hammerstein system using non-uniform rational b-spline neural network and particle swarm optimization, Neurocomputing 82 (0) (2012) 216 – 223. doi:10.1016/j.neucom.2011.11.016.

[32] W.-M. Hung, W.-C. Hong, Application of svr with improved ant colony optimization algorithms in exchange rate forecasting (2009).

[33] Y. Le Cun, I. Kanter, S. A. Solla, Eigenvalues of covariance matrices: Application to neural-network learning, Physical Review Letters 66 (18) (1991) 2396.

[34] F. X., Diebold, R. S. Mariano, Comparing predictive accuracy, Journal of Business and Economic Statistics (13) (1995) 253–265.

[35] A. Flexer, Statistical evaluation of neural network experiments: Minimum requirements and current practice, Cybernetics and Systems Research (1996) 1005–1008.

[36] B. Wilamowski, H. Yu, Improved computation for levenberg-marquardt training, Neural Networks, IEEE Transactions on 21 (6) (2010) 930–937. doi:10.1109/TNN.2010.2045657.

[37] J. Shawash, D. Selviah, Real-time nonlinear parameter estimation using the levenberg-marquardt algorithm on field programmable gate arrays, Industrial Electronics, IEEE Transactions on 60 (1) (2013) 170–176. doi:10.1109/TIE.2012.2183833.

[38] D. Saad (Ed.), On-Line Learning in Neural Networks, Cambridge University Press, 1999, cambridge Books Online.
URL http://dx.doi.org/10.1017/CBO9780511569920