



Cinvestav

Unidad Guadalajara

Discrete-Time Recurrent Neural Control

Edgar N. Sanchez, Alma Y. Alanis, and Alexander G. Loukianov

NSF, USA – Conacyt, Mexico, Workshop on ADP
Cocoyoc, Morelos, Mexico, April 2006

OUTLINE

- Introduction
- Discrete-Time Induction Motor Model
- Discrete-Time RHONN
- The EKF training algorithm
- Discrete-Time Adaptive Neural Control
 - *Backstepping technique*
 - *Sliding mode control using Discrete-Time RHONN*
 - *Luenberger-Like neural observer (RHONO)*
 - *Sliding mode control using Discrete-Time RHONO*

e-mail address

1. Introduction

- For discrete-time systems, the control problem is complex due to the couplings among subsystems, inputs and outputs. Besides, the noncausal problem is another difficulty which has to be solved when constructing adaptive controllers for discrete-time systems in strict feedback form.
- Few analytical results have been published in comparison with those for continuous-time domain. On the other hand, discrete-time neural networks are more convenient for real-time applications.

2. Induction Motor

The six-order discrete-time induction motor model in the stator fixed reference frame (α, β) , under the assumptions of equal mutual inductances and linear magnetic circuit, is given by

$$\omega(k+1) = \omega(k) + \frac{\mu}{\alpha}(1-a)M(i^\beta(k)\psi^\alpha(k) - i^\alpha(k)\psi^\beta(k)) - \left(\frac{\Delta T}{J}\right)T_L(k)$$

$$\psi^\alpha(k+1) = \cos(n_p\theta(k+1))\rho_1(k) - \sin(n_p\theta(k+1))\rho_2(k)$$

$$\psi^\beta(k+1) = \sin(n_p\theta(k+1))\rho_1(k) + \cos(n_p\theta(k+1))\rho_2(k)$$

$$i^\alpha(k+1) = \varphi^\alpha(k) + \frac{\Delta T}{\sigma}u^\alpha(k)$$

$$i^\beta(k+1) = \varphi^\beta(k) + \frac{\Delta T}{\sigma}u^\beta(k)$$

$$\begin{aligned} \theta(k+1) = & \theta(k) + \omega(k)\Delta T - \left(\frac{T_L(k)}{J}\right)(\Delta T)^2 \\ & + \frac{\mu}{\alpha}\left[\Delta T - \frac{(1-a)}{\alpha}\right]M(i^\beta(k)\psi^\alpha(k) - i^\alpha(k)\psi^\beta(k)) \end{aligned}$$

3. Discrete-Time Recurrent High Order Neural Networks (RHONN)

Consider the following discrete-time recurrent high order neural network (RHONN):

$$x_i(k + 1) = w_i^T z_i(x(k), u(k)), \quad i = 1, \dots, n$$

where:

x_i is the state of the i – th neuron,

w_i is the respective on-line adapted weight vector

$z_i(x(k), u(k))$ is given by:


$$z_i = \begin{bmatrix} z_1 \\ \vdots \\ z_L \end{bmatrix} = \begin{bmatrix} \prod_{j \in I_1} y_j^{d_j(1)} \\ \vdots \\ \prod_{j \in I_L} y_j^{d_j(L)} \end{bmatrix}, \quad y = \begin{bmatrix} y_1 \\ \vdots \\ y_n \\ \vdots \\ y_{n+m} \end{bmatrix} = \begin{bmatrix} S(x_1) \\ \vdots \\ S(x_n) \\ \vdots \\ u_m \end{bmatrix}$$

with $d_j(k)$ nonnegative integers,

L the respective number of high-order connections,

and $\{I_1, I_2, \dots, I_i\}$ a collection of non-ordered subsets

of $\{1, 2, \dots, n\}$

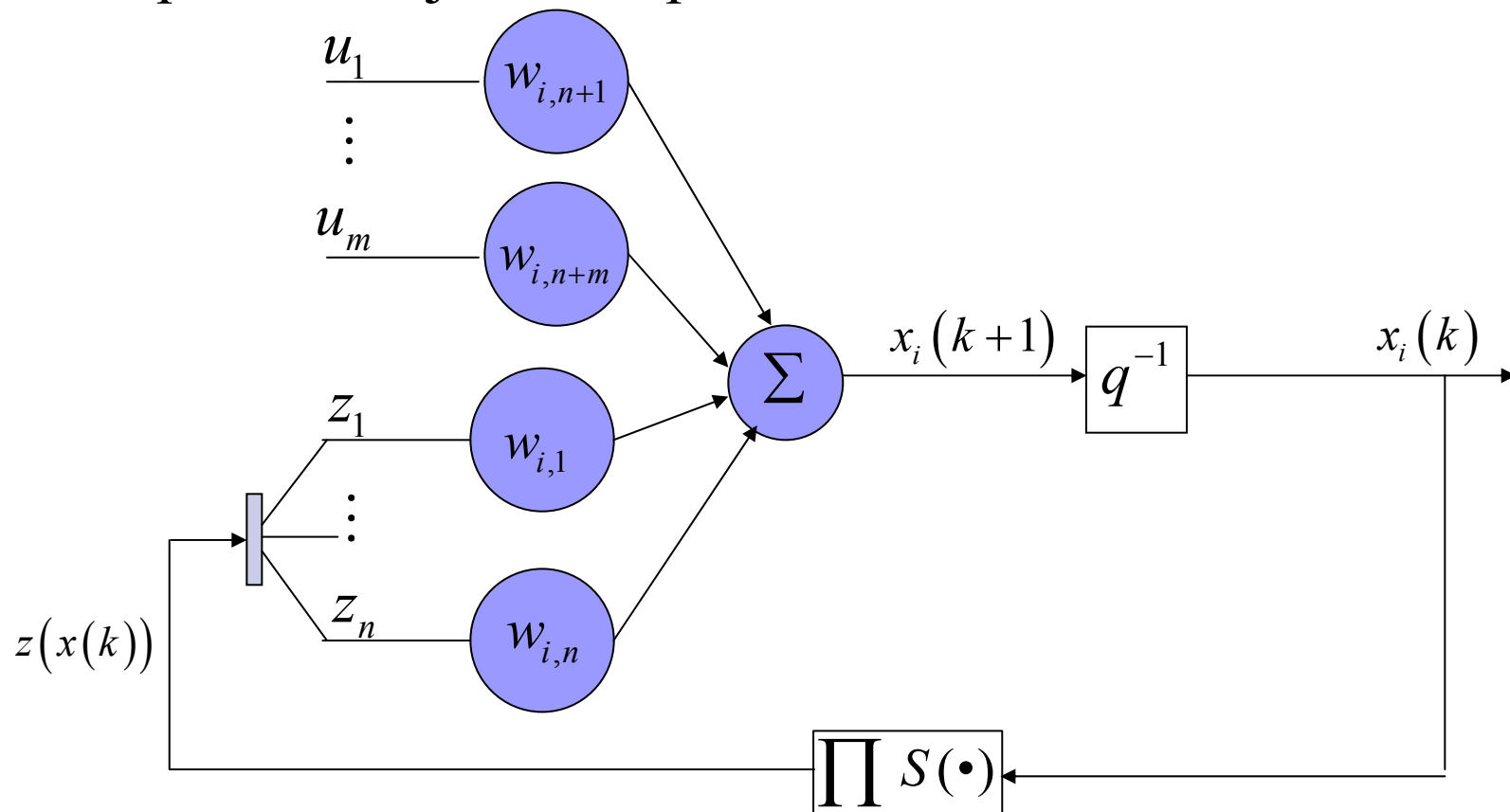


$u = [u_1 \quad u_2 \quad \cdots \quad u_m]^T$ is the external input vector to the network. The function $S(\cdot)$ is monotone-increasing, differentiable and is usually represented by a sigmoid function.

This kind of NN exhibits similar properties as the continuous-time one and it suits well for use in modelling, identification and control of dynamic complex systems, because of its relatively easy implementation and simple structure.

Some properties of the discrete-time RHONN are:


- They enable more efficient modeling of complex dynamics
- They are good candidates for identification and control
- Easy implementation and relatively simple structure
- Capable to adjust their parameters on-line



4 The EKF Training Algorithm

Kalman filtering (KF) estimates the state of a linear system with additive state and output white noises.

For KF-based neural network training, the network weights become the states to be estimated, with the error between the neural network output and the desired output being considered as additive white noise.



Due to the fact that the neural network mapping is nonlinear, extended Kalman Filtering (EKF) is required.

The training goal is to find the weight values, which minimize the prediction errors (the differences between the desired outputs and the neural network outputs).



*EXTENDED KALMAN FILTER ALGORITHM
FOR NEURAL NETWORK TRAINING*

The ideal neural network can be described by

$$w(k+1) = w(k)$$

$$y(k) = \varphi(w(k), u(k))$$

where the unknown weights can be seen as the state of the system.

This state will be estimated by an EKF

$$K(k) = P(k)H(k)[R(k) + H(k)P(k)H(k)]^{-1}$$

$$\hat{w}(k+1) = \hat{w}(k) + K(k)[y(k) - \hat{y}(k)]$$

$$P(k+1) = P(k) - K(k)H(k)P(k) + Q(k)$$

where

$P(k) \in \mathfrak{R}^{L \times L}$ is the prediction error covariance matrix,

$\hat{w} \in \mathfrak{R}^L$ is the estimated weight vector,

L is the respective number of neural network weights,

$y \in \mathfrak{R}^m$ is the desired output vector, and

$\hat{y} \in \mathfrak{R}^m$ is the neural network output vector.

$K \in \mathfrak{R}^{L \times m}$ is the Kalman gain vector,

$Q \in \mathfrak{R}^{L \times L}$ is the NN weight estimation noise covariance matrix,

$R \in \mathfrak{R}^{m \times m}$ is the error noise covariance,

$H \in \mathfrak{R}^{m \times L}$ is a matrix, in which

each entry H_{ij} is the derivative of i -th neural

network output \hat{y}_i , with respect to ij -th neural network

weight, \hat{w}_{ij} , as follows

$$H_{ij}(k) = \left[\frac{\partial \hat{y}_i(k)}{\partial \hat{w}_{ij}(k)} \right]; \quad i = 1, \dots, m; \quad j = 1, \dots, L$$

P , R and Q are initialized as diagonal matrices,

with values $P(0)$, $R(0)$ and $Q(0)$, respectively.

RHONN CASE

$$\hat{y}(k) = \phi(x(k), u(k))$$

Hence

$$\frac{\partial \hat{y}(k)}{\partial \hat{w}(k)} = \frac{\partial \hat{y}(k)}{\partial x(k)} \frac{\partial x(k)}{\partial \hat{w}(k)}$$

where x is the NN state

For the RHONN model, we have:

$$x(k+1) = w(k)z(x(k), u(k))$$

Then, it is possible to rewrite this expression as

$$x(k+1) = T(x(k), u(k), w(k))$$

Consequently

$$\frac{\partial x(k+1)}{\partial \hat{w}(k)} = \frac{\partial T(x(k), u(k), w(k))}{\partial x(k)} \frac{\partial x(k)}{\partial \hat{w}(k)} + \frac{\partial T(x(k), u(k), w(k))}{\partial \hat{w}(k)}$$

where $T(\cdot, \cdot, \cdot)$ is a nonlinear function of $x(k)$, $u(k)$ and $\hat{w}(k)$

Since the initial NN state does not depend on the weights,

$$H_{ij}(0) = 0$$

5. Discrete Time Recurrent Neural Control

Control of nonlinear systems is a major application area for neural networks. The control design problem could be approached in two ways:

- Direct design methods

The neural network directly implements the controller

- Indirect design methods

The control synthesis is based on a neural network model of the system to be controlled

5.1 Backstepping technique

Now, the block strict feedback form (BSFF) is applied to separate the control law synthesis problem into a number of sub-problems of lower order, which can be solved independently.

Once this partition is achieved, then the backstepping technique is used to design a suitable controller. Afterwards the resulting controller is approximated by a HONN. The implementation is simple and the training is performed on line by means of an extended Kalman Filter (EKF).

The model of many practical nonlinear systems can be expressed in or transformed into a special state-space form named block strict feedback form (BSFF)

$$\begin{aligned}x^i(k+1) &= f^i(x^i(k)) + g^i(x^i(k))x^{i+1}(k) + d^i(k) \\x^r(k+1) &= f^r(x^r(k)) + g^r(x^r(k))u(k) + d^r(k)\end{aligned}\tag{5.1}$$

$$y(k) = x^1(k)$$

where $1 \leq i \leq r-1$, $r \geq 2$, r is the number of blocks,

$u(k) = [u_1(k), \dots, u_m(k)]^T \in \mathfrak{R}^m$ are the system inputs;

$y(k) = [y_1(k), \dots, y_m(k)]^T \in \mathfrak{R}^m$ are the system outputs;

$d^i(k)$ and $d^r(k)$ are the bounded disturbance vectors;

$X(k) = [x^{1^T}, \dots, x^{r^T}(k)]^T$ are the state variables;

$\bar{x}^i(k) = [x^{1^T}, x^{2^T}, \dots, x^{i^T}]^T$; $x^i \in \mathfrak{R}^{n_i}$; $f^i(\bullet)$ and $g^i(\bullet)$ are smooth nonlinear functions and the set of numbers (n_1, \dots, n_r) define the system structure as

$$n_1 \leq n_2 \leq \dots \leq n_r \leq m$$

Once (4.1) is defined, we apply the well known backstepping technique.

Now, we can define the desired virtual controls and the ideal practical controls for each subsystem, as follows:

$$\alpha^{1*}(k) \triangleq x^2(k+r-1) = \frac{1}{G^1(k)} \left[y_d(k+r_j) - F^1(k) \right]$$

$$\alpha^{2*}(k) \triangleq x^3(k+r-1) = \frac{1}{G^2(k)} \left[\alpha^{2*}(k) - F^2(k) \right]$$

⋮


$$\alpha^{r-1*}(k) \triangleq x^r(k+r-1) = \frac{1}{G^{r-1}(k)} \left[\alpha^{r-1*}(k) - F^{r-1}(k) \right]$$

$$u^*(k) \triangleq \frac{1}{g^r(k)} \left[\alpha_r^*(k) - F^r(k) \right]$$

$$y(k) = x^1(k)$$

It is easy to see that the above equation stabilize the system in each step without the causality problem. It can be further written as

$$\begin{aligned}\alpha^{1*}(k) &\triangleq x^2 = \varphi^1(\bar{x}^1(k), y_d(k + n_j)) \\ \alpha^{2*}(k) &\triangleq x^3 = \varphi^2(\bar{x}^2(k), \alpha^{1*}(k)) \\ &\vdots \\ \alpha^{r-1*}(k) &\triangleq x^r = \varphi^{r-1}(\bar{x}^{r-1}(k), \alpha^{r-2*}(k)) \\ u^*(k) &\triangleq \varphi^r(X(k), \alpha^{r-1*}(k)) \\ y(k) &= x^1(k)\end{aligned}\tag{5.2}$$



where φ^j ($1 \leq j \leq r$) are nonlinear functions. It is obvious that the desired virtual controls $\alpha^{i*}(k)$ and the ideal control $u^*(k)$ will drive the output of the j th block to track the desired signal only if the exact system model is known and without disturbances.

However in practical applications these two conditions cannot be satisfied. In the following, neural networks will be used to approximate the desired virtual controls, as well as the desired practical controls, when the conditions establish above are not satisfied.

Therefore, we construct the controls via embedded backstepping without causality contradiction. Let select the virtual controls and practical controls as follows:

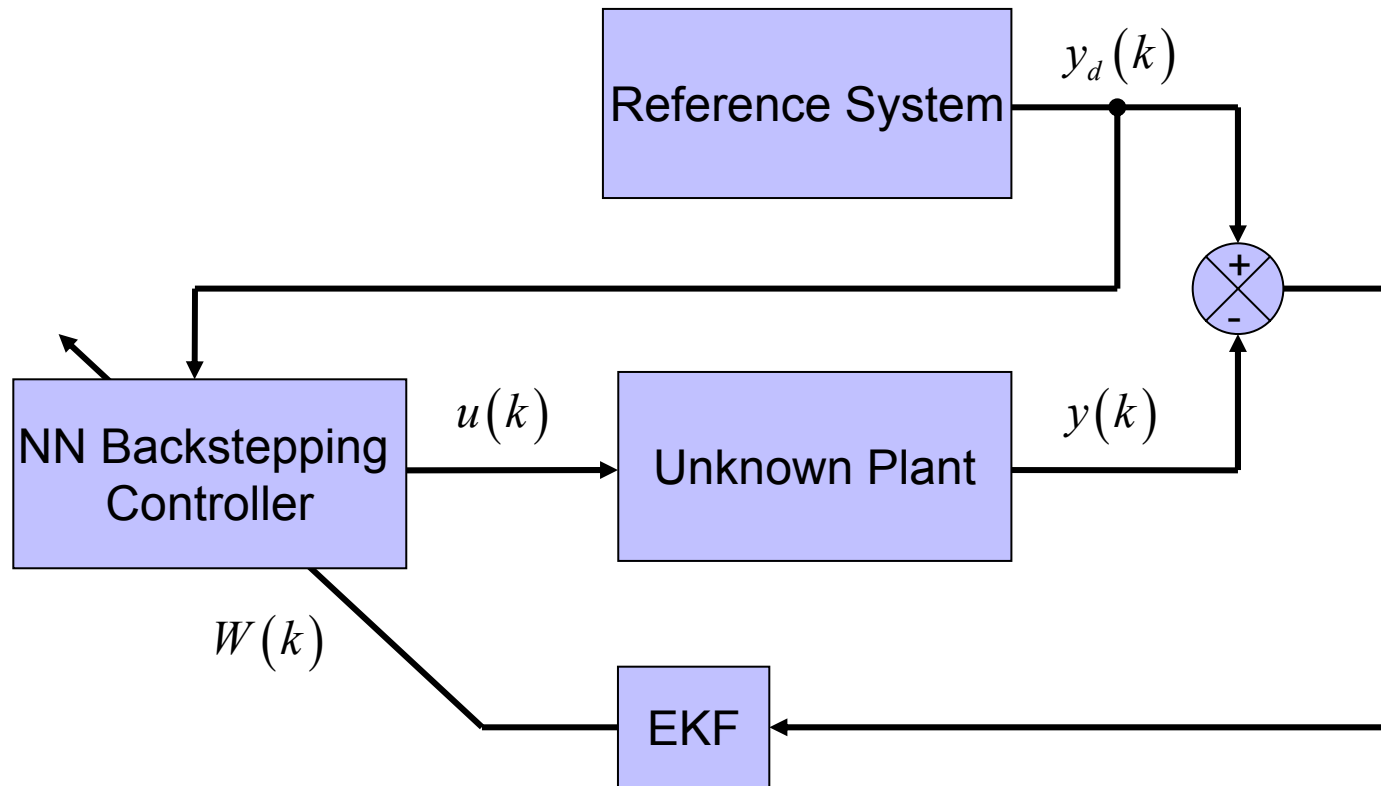
$$\begin{aligned}\alpha^i &= w^i S^i(z^i(k)) \\ u &= w^r S^r(z^r(k))\end{aligned}\tag{5.3}$$

with

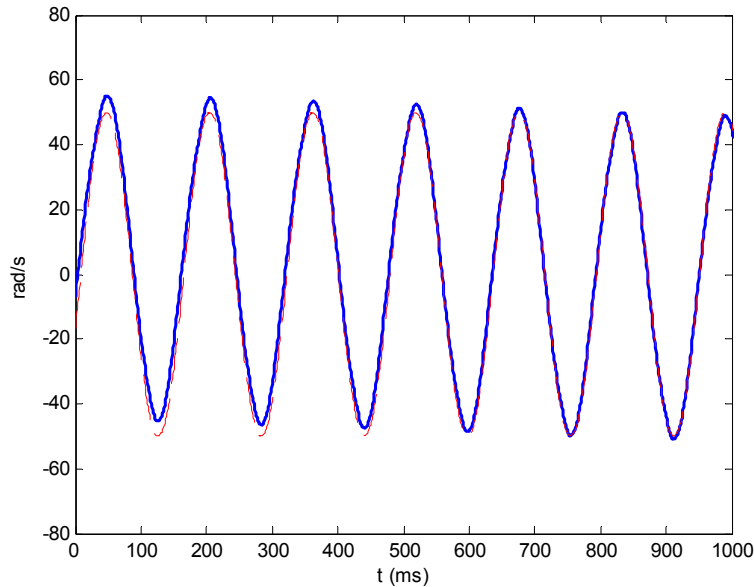
$$\begin{aligned}z^i(k) &= \left[\bar{x}^{iT}(k), y_d(k + n_j) \right]^T \\ z^r(k) &= \left[X(k), \alpha^{r-1}(k) \right]^T \quad i = 1, \dots, r-1\end{aligned}$$

Theorem 1: For system (5.1), the HONN (5.3), trained with an EKF-based algorithm, to approximate the control law (5.2), ensures that the tracking error is semiglobally uniformly ultimately bounded (SGUUB); additionally the HONN weights are also bounded.

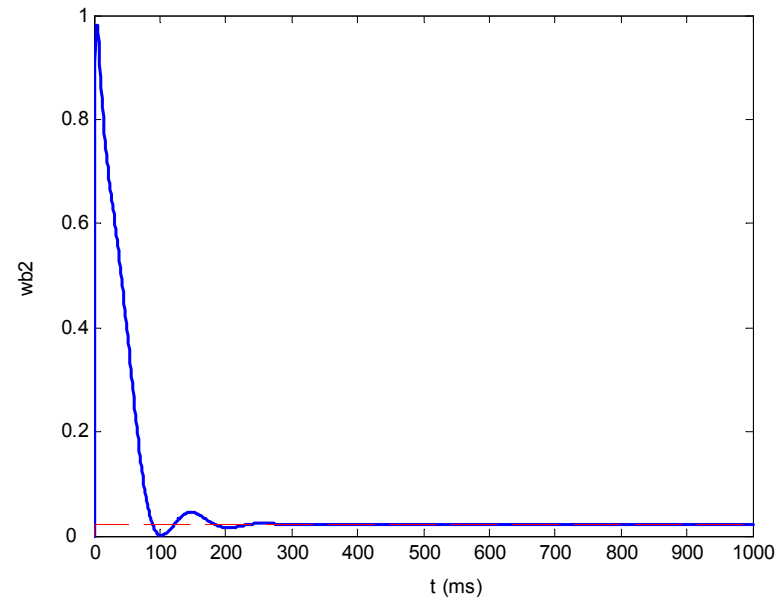
NN Discrete-Time Backstepping



Simulation results for a discrete-time induction motor model

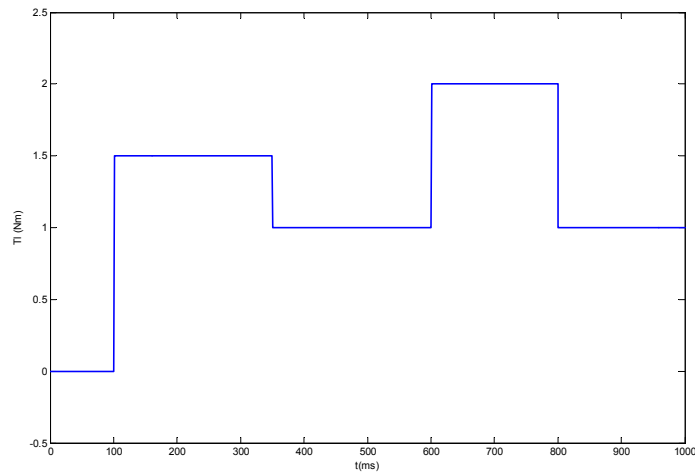


Tracking performance
 $\omega(k)$ (blue line) and
 $\omega_d(k)$ (red line)

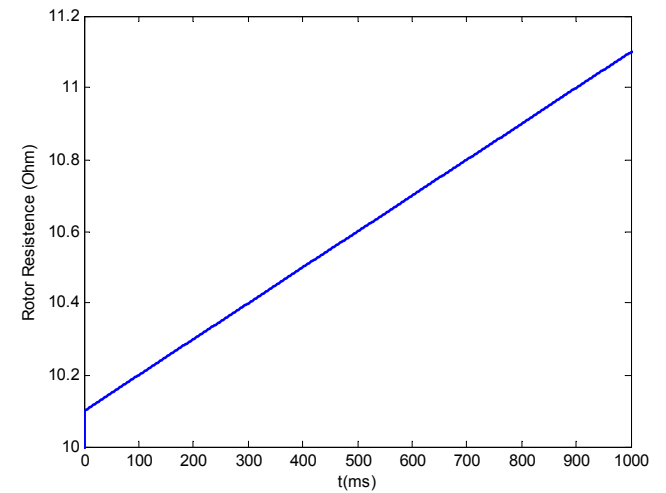


Tracking performance
 $\psi(k)$ (blue line) and
 $\psi_d(k)$ (red line)

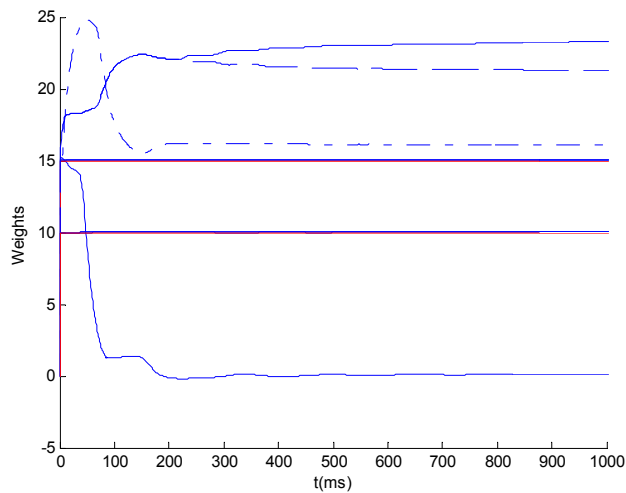
Load torque $T_L (k)$



Rotor resistance (R_r)



Weight evolution



5.2 Sliding mode control using Discrete-Time RHONN

Above, we develop an algorithm to control a Discrete-Time MIMO nonlinear system based on the backstepping technique; now, we develop a different algorithm for the same class of system, based on the sliding mode and the block control techniques.

This implementation is done as follows: first, a RHONN is trained with the EKF algorithm in order to identify the dynamic behavior of the system; then, based on this NN model, a sliding mode controller is synthesized using the Block Controllable Form (BCF).

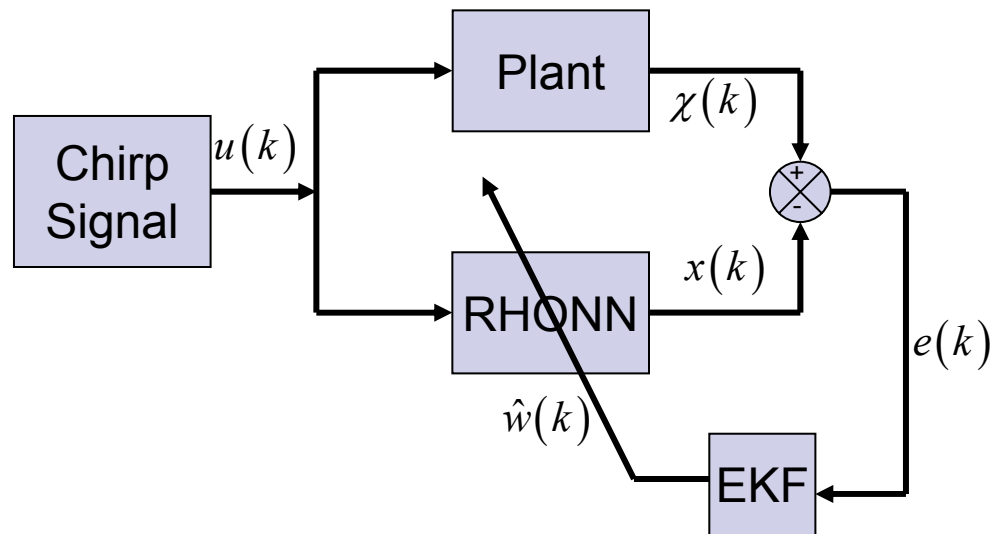
5.2.1 RHONN Identifier

The identification problem consists of choosing an appropriate identification model and adjusting its parameters according to some adaptive law, such that the response of the model to an input signal (or class of input signals), approximates the response of the real system to the same input.


In this section, we use an Extended Kalman Filter (EKF)-based training algorithm for the RHONN, in order to identify MIMO discrete-time nonlinear systems.

We consider a discrete-time nonlinear system

$$\chi(k+1) = F(\chi(k), u(k)) \quad (5.4)$$

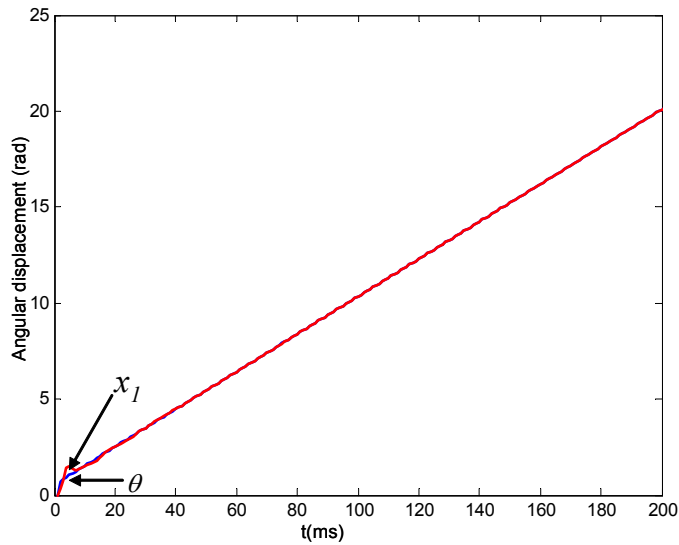


Identification scheme

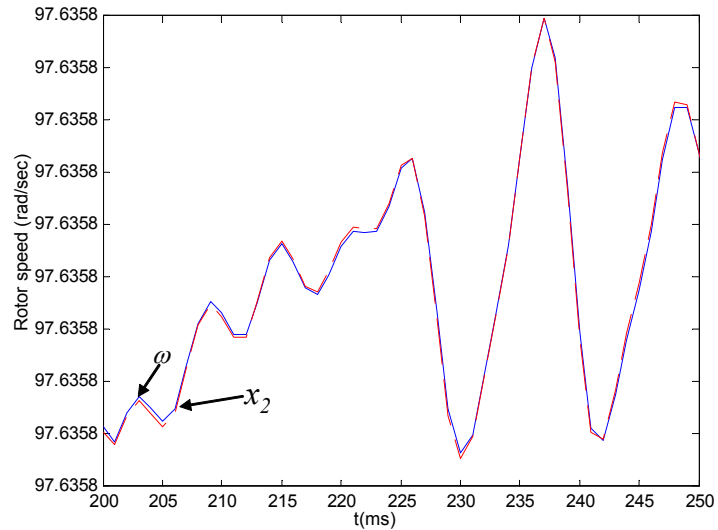


Theorem 2: The RHONN, trained with the EKF-based algorithm to identify a nonlinear plant (5.4), ensures that the identification error is semiglobally uniformly ultimately bounded (SGUUB); additionally the RHONN weights are also bounded.

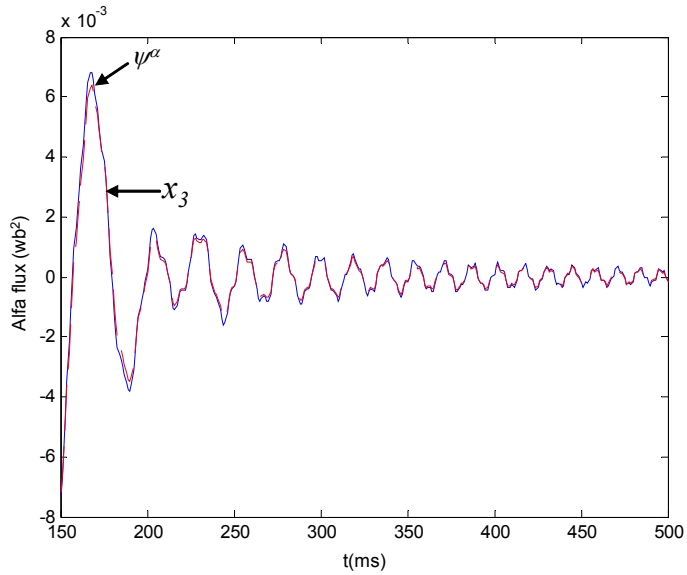
Simulation results for a discrete-time induction motor model



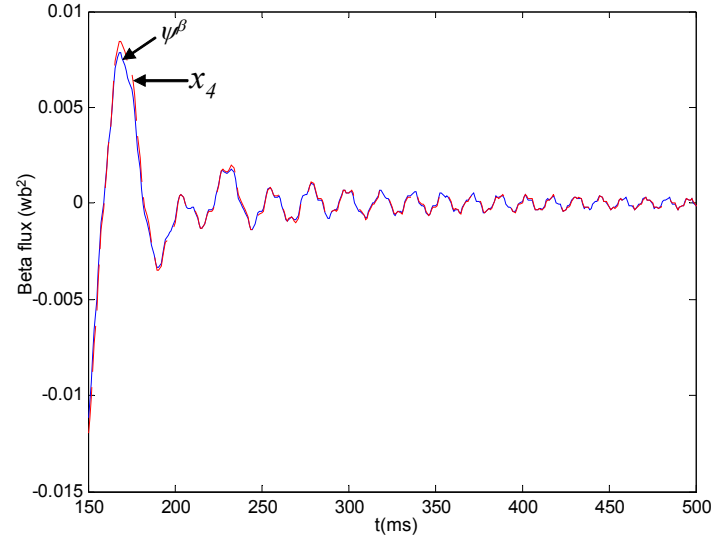
Angular displacement identification



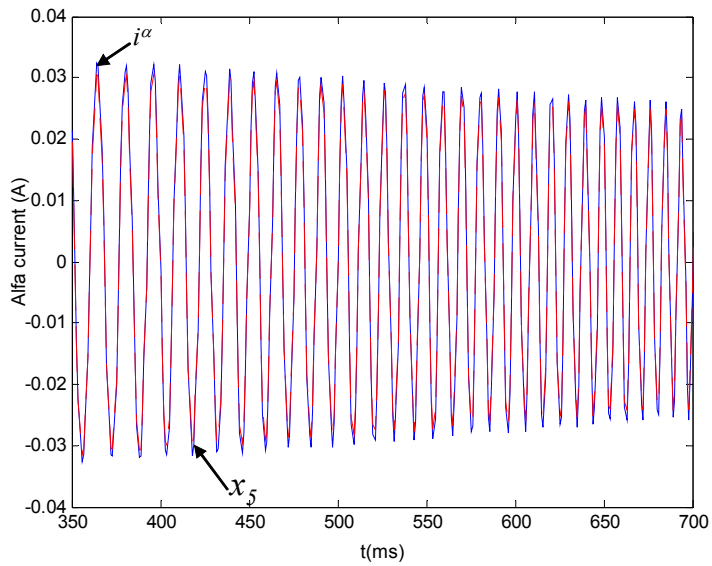
Rotor speed identification



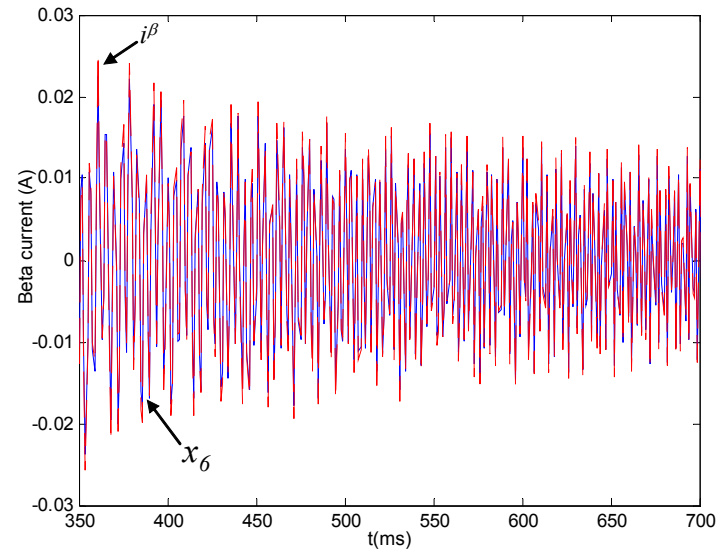
Alpha-flux identification



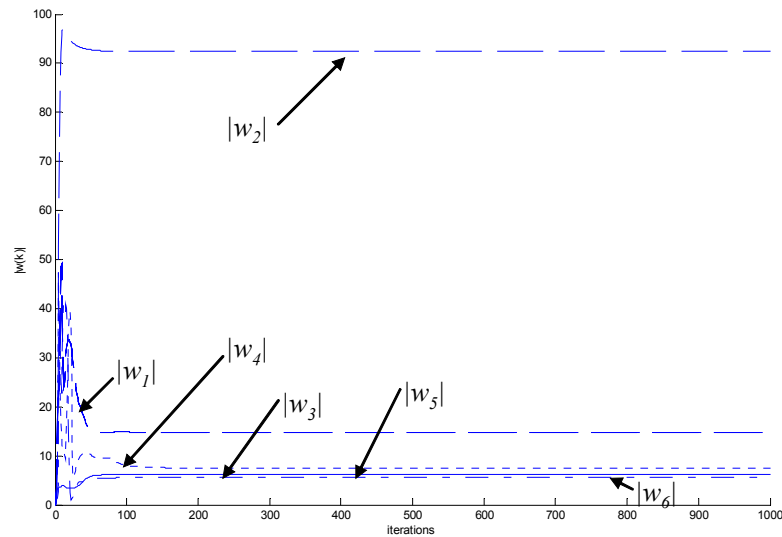
Beta-flux identification



Alpha-current identification



Beta-current identification



Weight evolution

5.2.2 Sliding Mode Block Control

Consider the following system:

$$x(k+1) = f(x(k)) + B(x(k))u(k) + d(k)$$

$$y(k) = Cx(k)$$

where

$x \in \mathfrak{R}^n$ is the state vector of the system,

$u \in \mathfrak{R}^m$ is the input vector,

$y \in \mathfrak{R}^p$ is the output vector, and

$d(\cdot)$ is a disturbance vector.

The vector $f(\cdot)$, the columns of $B(\cdot)$

and $d(\cdot)$ are smooth vector fields.

By means of non-singular transformation, this system can be represented in the block controllable form as follows:

$$\begin{aligned}
 x_i(k+1) &= f_i(x_i(k)) + B_i(x_i(k))x_{i+1}(k) + d_i(k) \\
 x_r(k+1) &= f_r(x(k)) + B_r(x(k))u(k) + d_r(k) \\
 y(k) &= x_1(k), \quad i = 1, \dots, r-1
 \end{aligned} \tag{5.5}$$

where $x(k) = [x_1(k) \cdots x_i(k) \cdots x_r(k)]^T$,

$d(k) = [d_1(k) \cdots d_i(k) \cdots d_r(k)]^T$ and

$\bar{x}_i(k) = [x_1(k) \cdots x_i(k)]^T$, $i = 1, \dots, r-1$, and the set of numbers

(n_1, \dots, n_r) , which define the structure of the system, satisfy

$$n_1 \leq n_2 \leq \cdots \leq n_r \leq m.$$

Define the following transformation:

$$z_1(k) = x_1(k) - x_d(k)$$

⋮

$$z_i(k) = x_i(k) - x_i^d(k) \quad i = 1, \dots, r$$

with x_i^d as the desired value of x_i as

$$x_i^d(k) = [B_{i-1}(x_{i-1}(k))]^{-1} (K_{i-1}z_{i-1}(k)) \\ + [B_{i-1}(x_{i-1}(k))]^{-1} (f_{i-1}(x_{i-1}(k)) + d_{i-1}(k))$$

$y_d(k) = x_d(k)$ as the desired trajectory for tracking. The system can be rewritten as

$$z_1(k+1) = K_1z_1(k) + B_1z_2(k)$$

⋮

$$z_{r-1}(k+1) = K_{r-1}z_{r-1}(k) + B_{r-1}z_r(k)$$

$$z_r(k+1) = f_r(x(k)) + B_r(x(k))u(k) + d_r(k) - x_r^d(k)$$

To design the control law, we use the sliding mode block control technique. The sliding surface is derived from the block control procedure with $S(k) = z_r(k) = 0$.

$$z_1(k+1) = K_1 z_1(k) + B_1 z_2(k)$$

⋮

$$z_{r-1}(k+1) = K_{r-1} z_{r-1}(k) + B_{r-1} S(k)$$

$$S(k+1) = f_r(x(k)) + B_r(x(k))u(k) + d_r(k) - x_r^d(k)$$

Once the sliding surface is selected, the next step is to define $u(k)$, as

$$u(k) = \begin{cases} u_{eq}(k) & \text{for } \|u_{eq}(k)\| \leq u_0 \\ u_0 \frac{u_{eq}(k)}{\|u_{eq}(k)\|} & \text{for } \|u_{eq}(k)\| > u_0 \end{cases} \quad (5.6)$$

where the equivalent control is calculated from $S(k+1) = 0$, as

$$u_{eq}(k) = [B_r(x(k))]^{-1} (-f_r(x(k)) + x_r^d(k+1) - d_r(k))$$

Theorem 3. The control law (5.6) ensures the sliding surface $S(k) = z_r(k) = 0$ for system (5.5).

Proposition 1. Given a desired output trajectory x_d , a dynamic system with output $y = \chi_1$, and a neural network with output x_1 , the following inequality holds:


$$\|x_d - \chi_1\| \leq \|x_1 - \chi_1\| + \|x_d - \chi_1\|$$

where:

$x_d - \chi_1$ is the system output tracking error,

$x_1 - \chi_1$ is the output identification error and

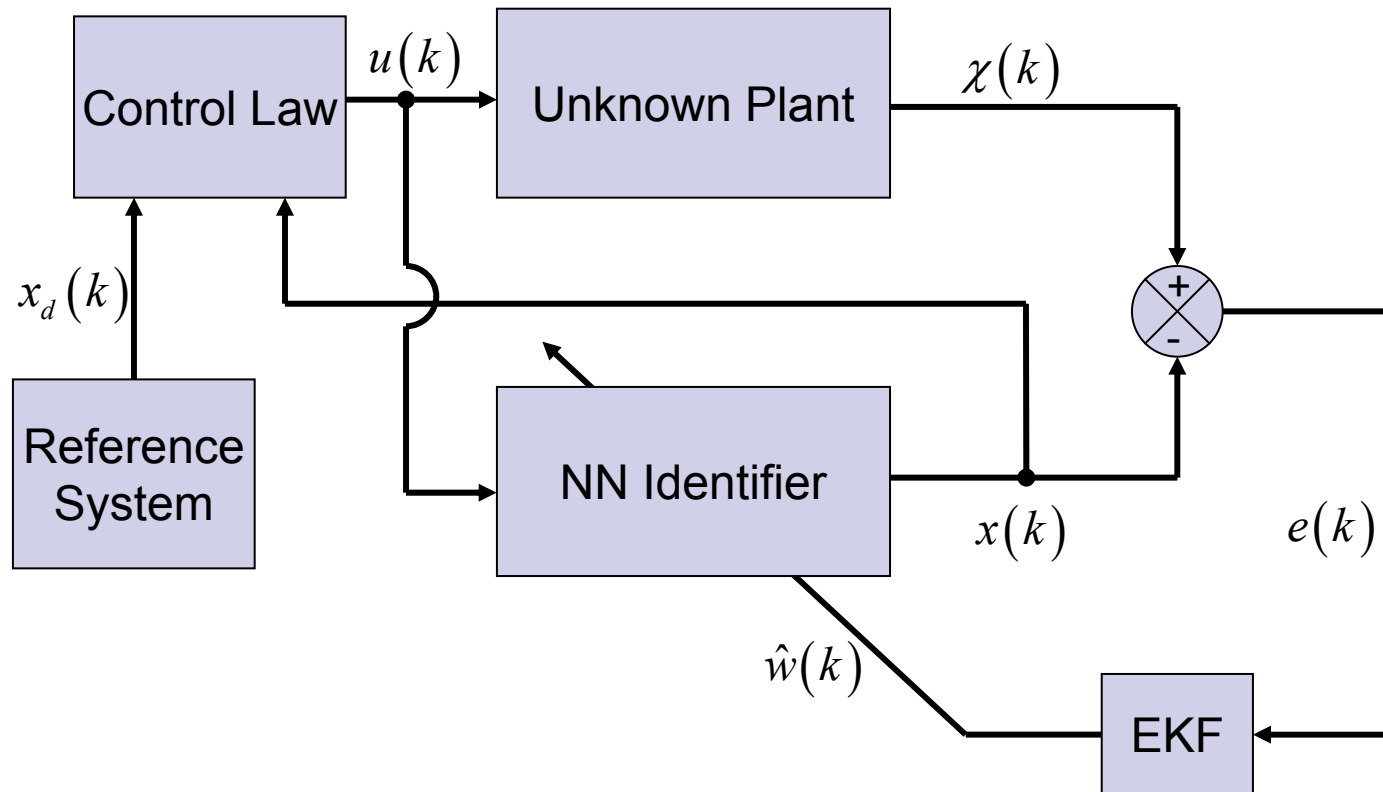
$x_d - \chi_1$ is the output tracking error of the neural network.



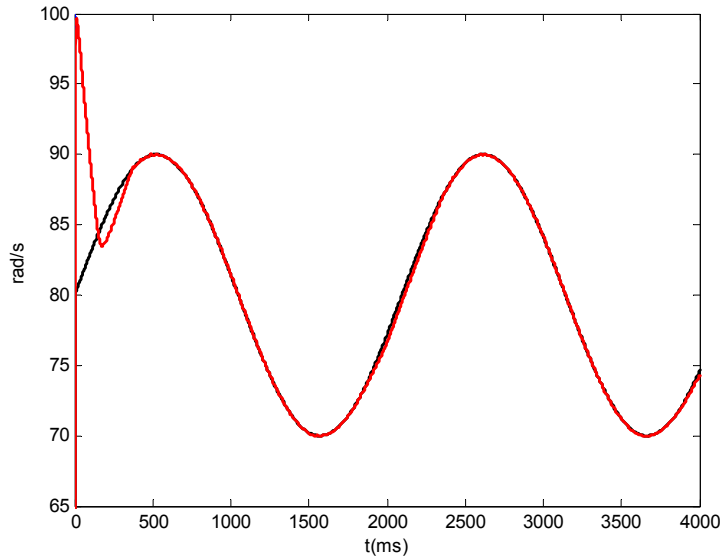
Based on this proposition, it is possible to divide the tracking objective into two parts:

1. Minimization of $x_1 - \chi_1$, which can be achieved by the proposed on-line RHONN identifier algorithm as established in *Theorem 2*.
2. Minimization of $x_d - x_1$. For this, a tracking algorithm is developed on the basis of the RHONN. This minimization is obtained by designing the control law (5.6), as stated in *Theorem 3*.

NN Discrete-Time Sliding Mode Block Control



Simulation results for a discrete-time induction motor model

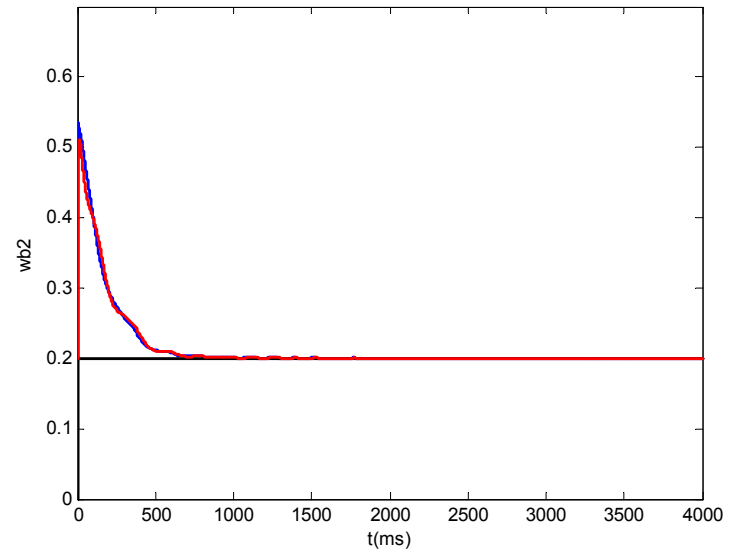


Tracking performance

$\omega(k)$ (blue line -Plant signal)

$x_2(k)$ (red line -RHONN signal) and

$\omega_d(k)$ (black line -Desired signal)



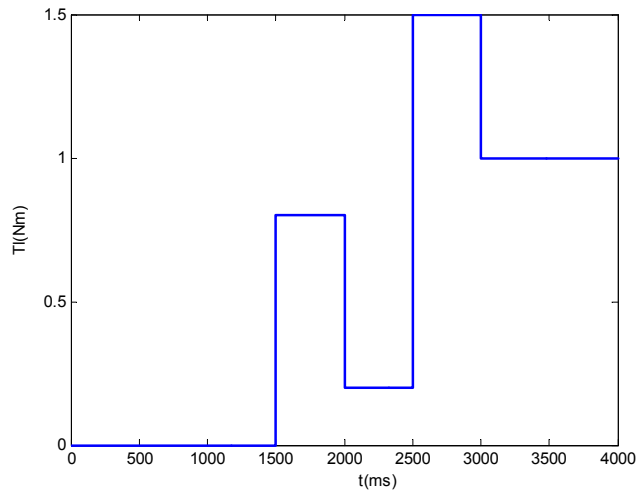
Tracking performance

$\psi(k)$ (blue line -Plant signal)

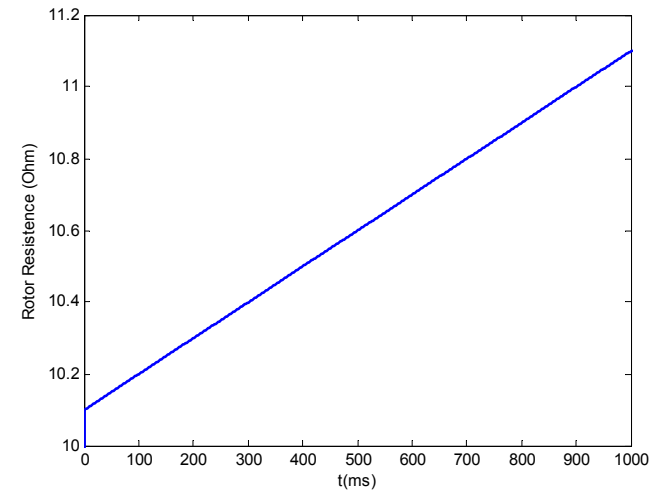
$x_1^2 + x_2^2$ (red line -RHONN signal) and

$\psi_d(k)$ (black line -Desired signal)

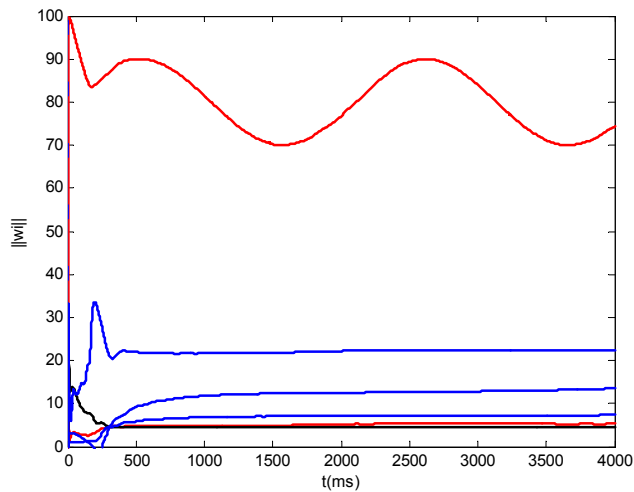
Load torque $T_L (k)$



Rotor resistance (R_r)



Weight evolution



5.3 Luenberger-Like neural observer

In this section, we consider a class of MIMO discrete-time nonlinear system, for which we develop a Luenberger-like observer.

This observer is based on a recurrent high order neural network (RHONN), which estimates the state vector of the unknown plant dynamics. It deals with the so-called mixed uncertainties (the presence of simultaneous external and internal uncertainties).

The learning algorithm for the RHONN is based on an extended Kalman filter.

Neural observer design

We consider to estimate the state of a system, which is assumed to be observable, given by

$$\begin{aligned}x(k+1) &= F(x(k), u(k)) + d(k) \\ y(k) &= Cx(k)\end{aligned}\tag{5.7}$$

where

$x \in \mathfrak{R}^n$ is the state vector of the system,

$u(k) \in \mathfrak{R}^m$ is the input vector,

$y(k) \in \mathfrak{R}^p$ is the output vector,

$C \in \mathfrak{R}^{p \times n}$ is a known output matrix,

$d(k) \in \mathfrak{R}^n$ is a disturbance vector and

$F(\cdot)$ is a smooth vector field, with $F_i(\cdot)$ entries.

Hence (5.7) can be rewritten as:

$$\begin{aligned}x(k) &= \begin{bmatrix} x_1(k) & \cdots & x_i(k) & \cdots & x_n(k) \end{bmatrix}^T \\d(k) &= \begin{bmatrix} d_1(k) & \cdots & d_i(k) & \cdots & d_n(k) \end{bmatrix}^T\end{aligned}\quad (5.8)$$

$$x_i(k+1) = F_i(x(k), u(k)) + d_i(k)$$


$$y(k) = Cx(k)$$

For this system, we propose a Luenberger-like neural observer (RHONO) with the following structure:

$$\hat{x}(k) = \begin{bmatrix} \hat{x}_1(k) & \cdots & \hat{x}_i(k) & \cdots & \hat{x}_n(k) \end{bmatrix}^T\quad (5.9)$$

$$\hat{x}_i(k+1) = w_i^T z(\hat{x}(k), u(k)) + L_i e(k)$$

$$y(k) = C\hat{x}(k)$$



Theorem 4: For system (5.8), the RHONN (5.9), trained with the EKF-based algorithm, ensures that the output error and the state estimation error are semiglobally uniformly ultimately bounded (SGUUB); additionally the RHONO weights remain bounded.

Simulation Results

In this section, the neural observer is applied to a modified Van der Pol oscillator, whose nonlinear dynamics is represented by the following equation

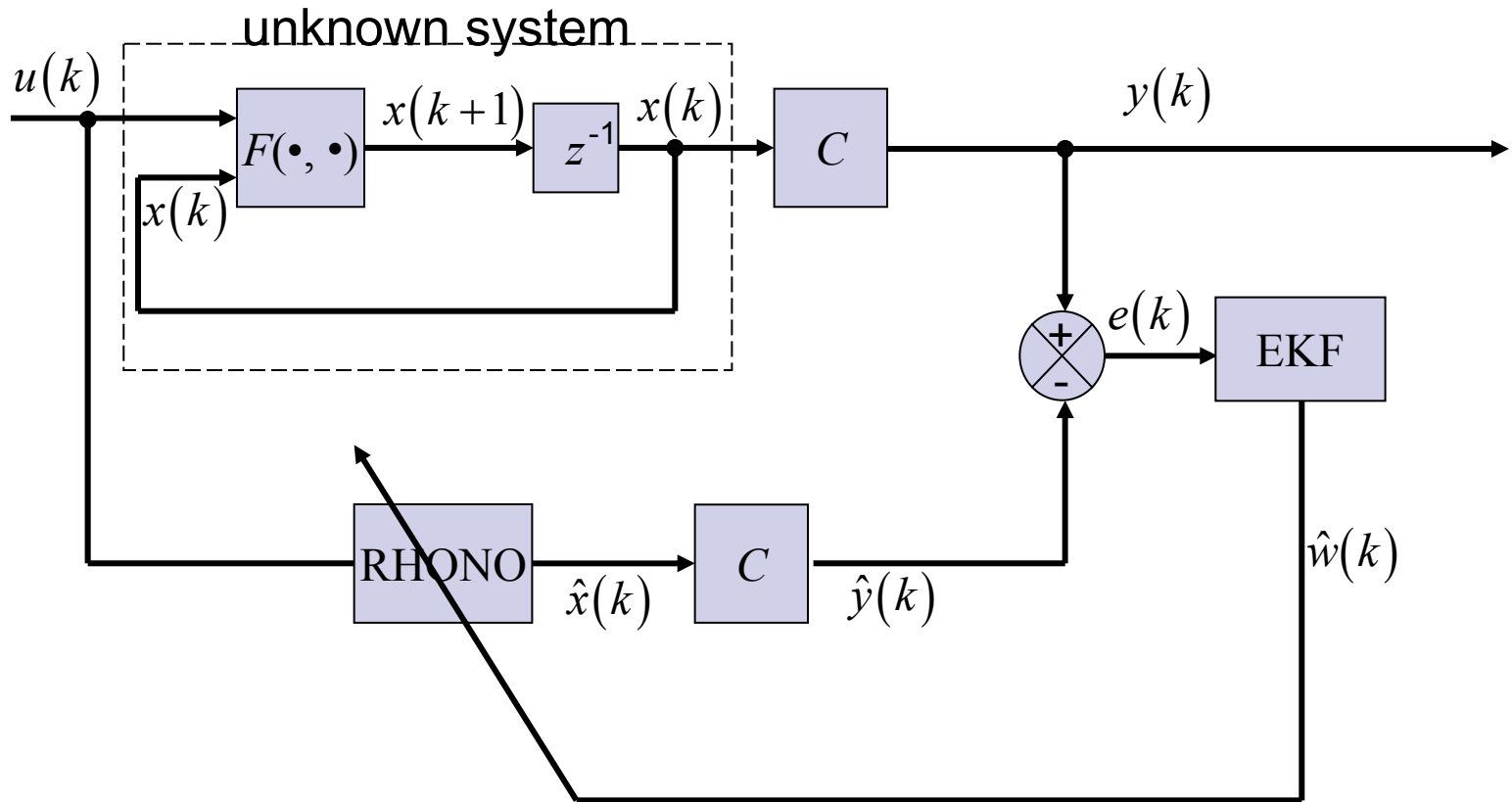
$$x_1(k+1) = x_1(k) + \Delta T x_2(k) + d_1(k)$$

$$x_2(k+1) = x_2(k) + \Delta T \left(-\xi \left(x_1^2(k) - 1 \right) x_2(k) \right) \\ + \Delta T \left(-x_1(k) + u(k) \right) + d_2(k)$$

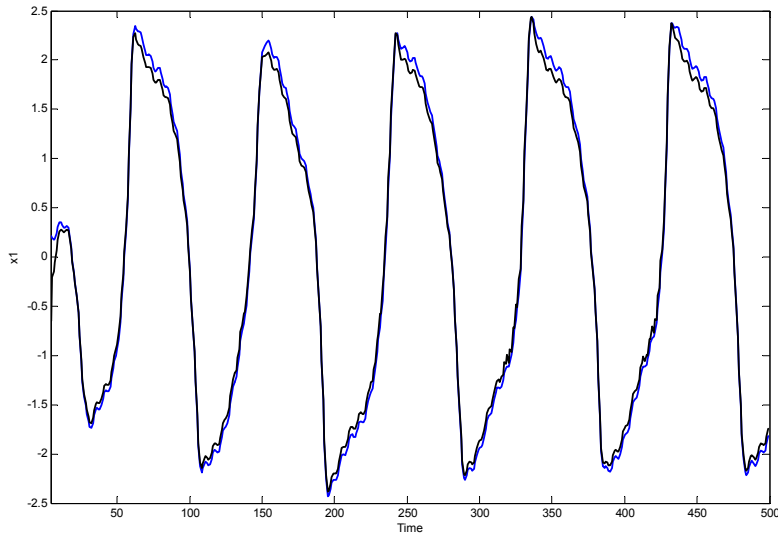
$$y(k) = x_1(k)$$

$$d_1(k) = 0.1 \sin(k)$$

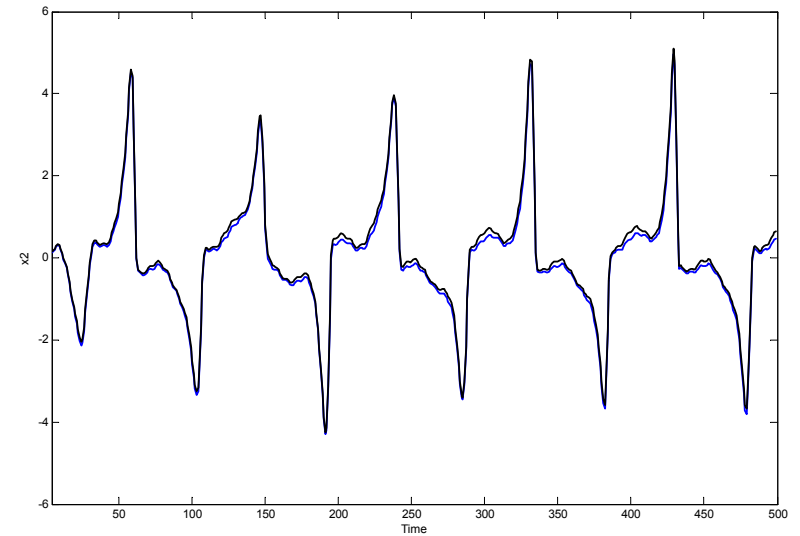
$$d_2(k) = 0.1 \cos(k)$$



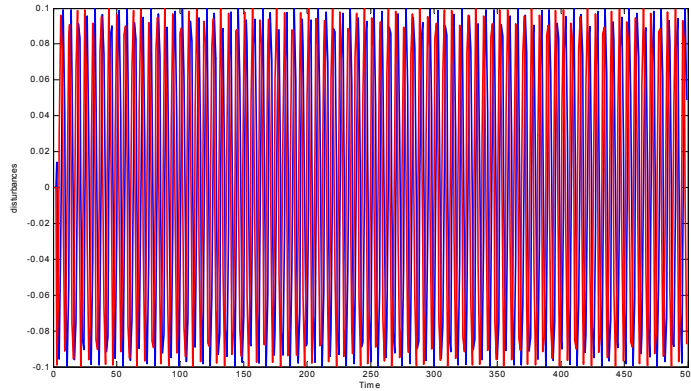
Observation scheme



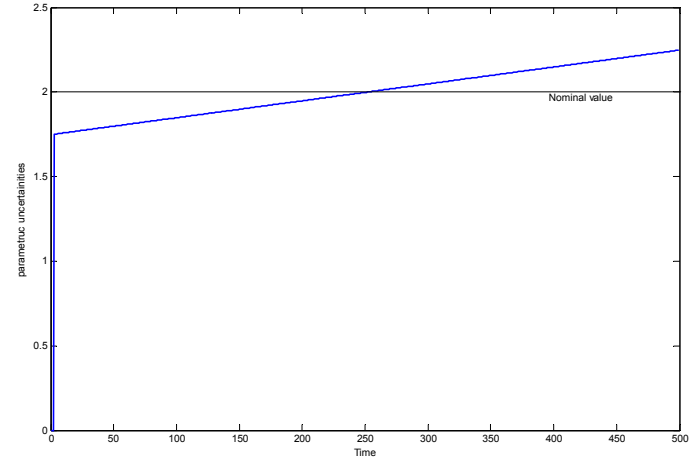
Time evolution of the state $x_1(k)$ (blue line) and its estimated $\hat{x}_1(k)$ (black line)



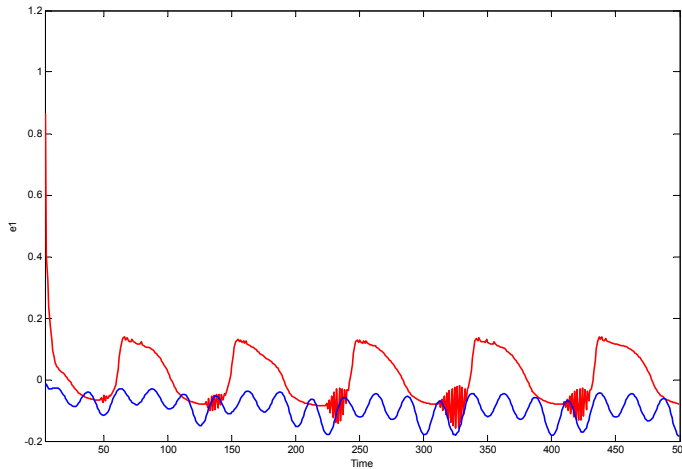
Time evolution of the state $x_2(k)$ (blue line) and its estimated $\hat{x}_2(k)$ (black line)



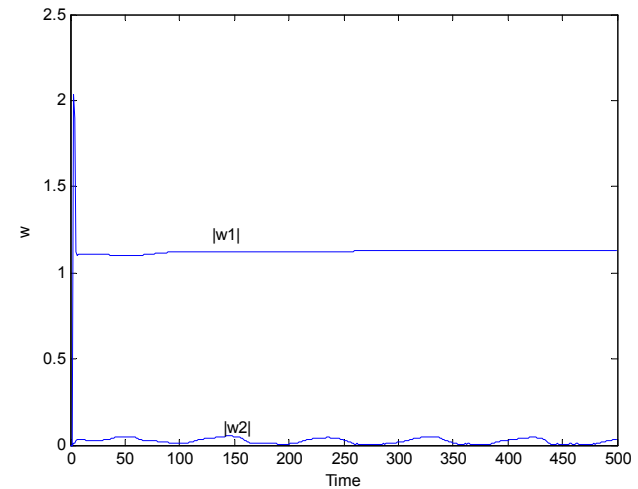
Disturbances $d_1(k)$ (blue line)
and $d_2(k)$ (black line)



Uncertainties in parameter ξ



$\tilde{x}_1(k)$ (blue line) and
 $\tilde{x}_2(k)$ (black line)



Weight evolution

5.4 Sliding mode control using Discrete-Time RHONO

Above we develop an algorithm to control a Discrete-Time MIMO nonlinear system based on the sliding mode technique assuming the state measurement; now we develop the same technique but using the discrete-time RHONO, and on the basis of *Theorem 3*, *Theorem 4* and *Proposition 2*.

Proposition 2. Given a desired output trajectory y_d , a dynamic system with output y , and a neural observer with output \hat{y} , the following inequality holds:


$$\|y_d - y\| \leq \|\hat{y} - y\| + \|y_d - \hat{y}\|$$

where:

$y_d - y$ is the system output tracking error,

$\hat{y} - y$ is the output estimation error and

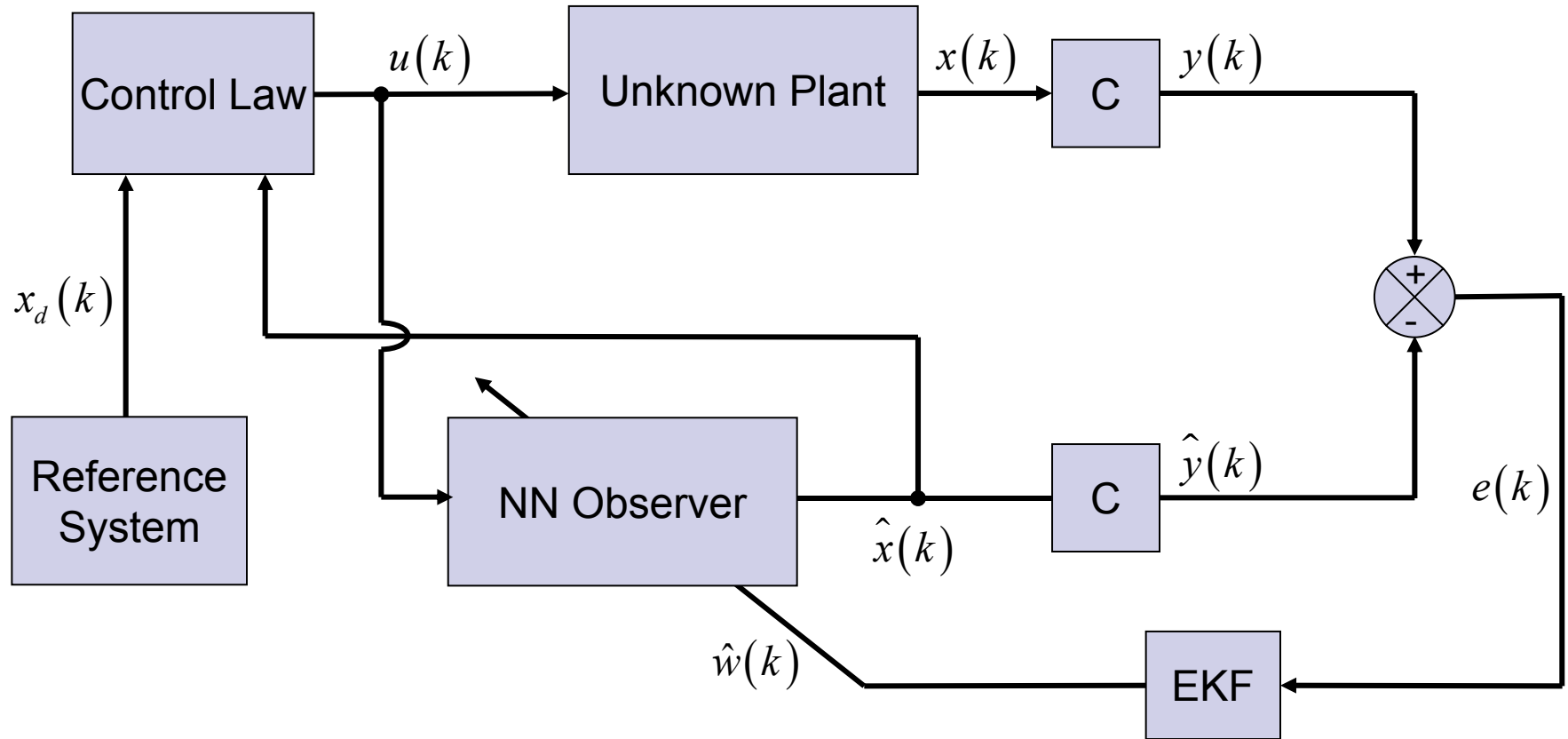
$y_d - \hat{y}$ is the output tracking error of the neural observer.



Based on this proposition, it is possible to divide the tracking objective into two parts:

1. Minimization of $\hat{y} - y$, which can be achieved by the proposed on-line RHONN observer algorithm as established in *Theorem 5*.
2. Minimization of $y_d - \hat{y}$. For this, a tracking algorithm is developed on the basis of the RHONO. This minimization is obtained by designing the control law (5.6), as stated in *Theorem 3*.

Discrete-Time Sliding Mode Block Control with Neural Observer



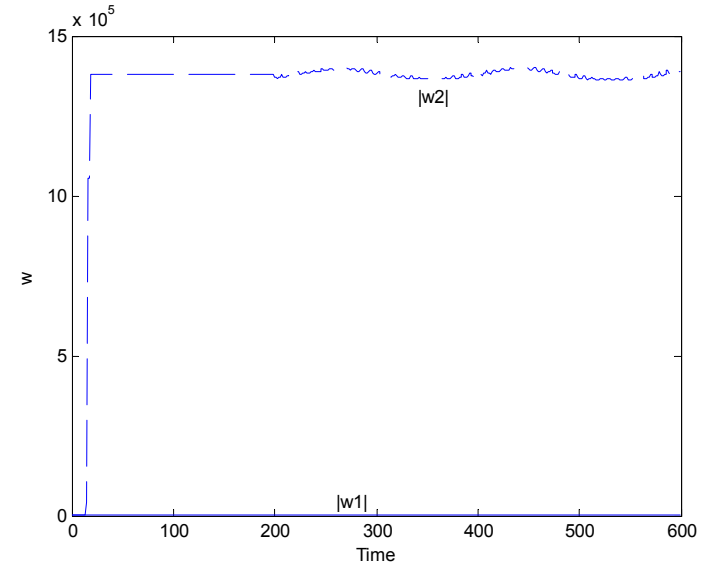
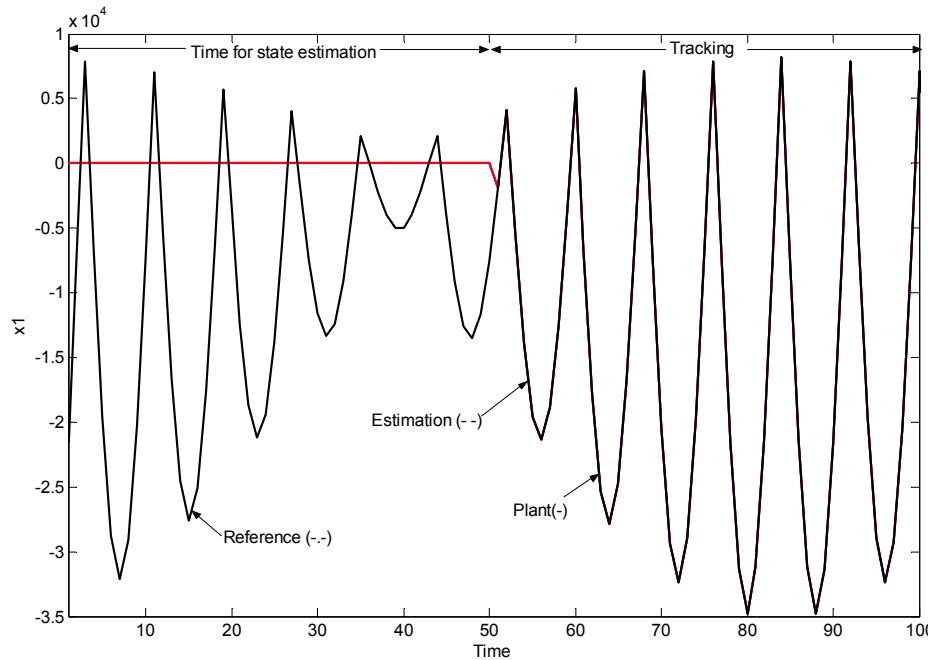


To illustrate the applicability of the proposed approach, we present the synchronization of two representative chaotic systems.

Nonlinear systems have very rich dynamical behaviors including chaos, characterized by high sensitivity to parameter variation, external disturbance, and particularly tiny changes of initial conditions.

Chaotic attractors are globally bounded but locally unstable, which presents some important properties that have technological applications. Two typical discrete chaotic systems are used for simulation here, on their synchronization under the control of the proposed scheme: The Hénon System and the Lozi one.

Simulation results for discrete-time chaos synchronization



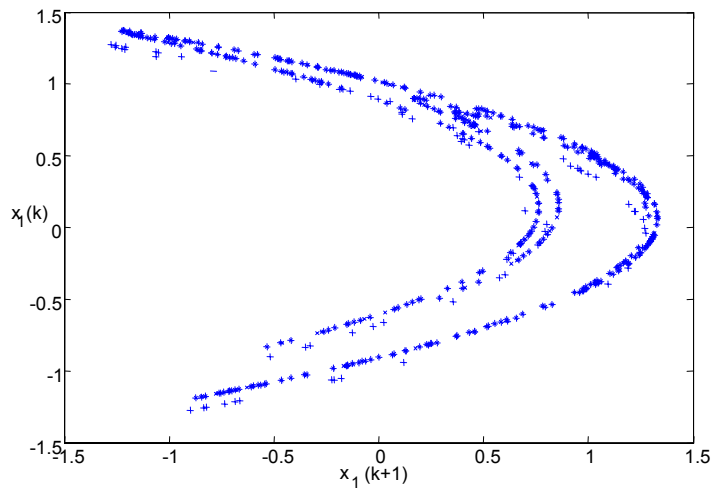
Tracking performance

$x_1(k)$ (blue line -Plant signal)

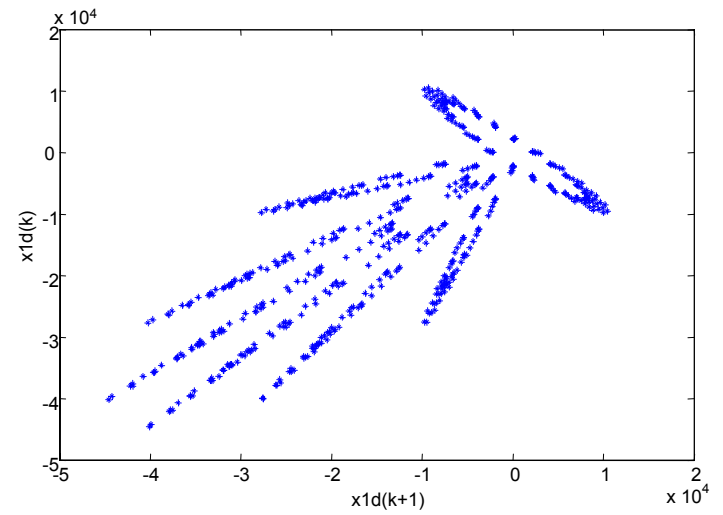
$\hat{x}_1(k)$ (red line -RHONO signal) and

$x_{1,d}(k)$ (black line -Desired signal)

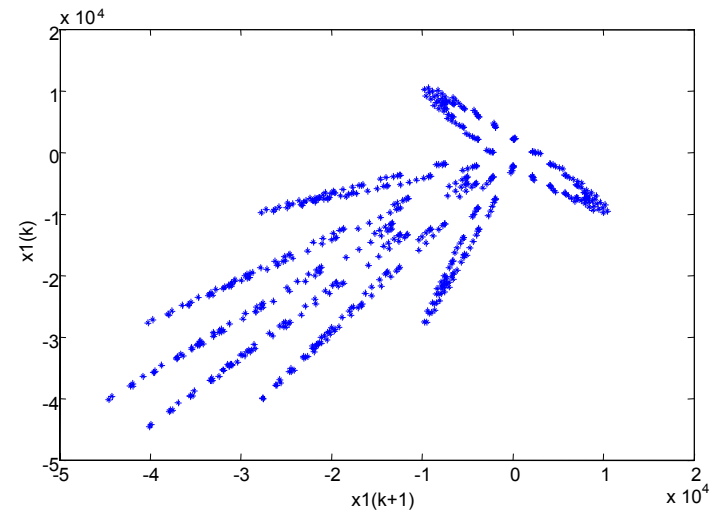
Weight evolution



Original Hénon Attractor



Reference Lozi Attractor



Synchronized Attractor



Acknowledgement:

The authors thank the support of CONACYT Mexico and CINVESTAV Guadalajara.



e-mail address

sanchez@gdl.cinvestav.mx

aalanis@gdl.cinvestav.mx