A Real-Time Learning Control Approach for Nonlinear Continuous-Time System Using Recurrent Neural Networks

Tommy W. S. Chow, *Member, IEEE,* Xiao-Dong Li, and Yong Fang

***Abstract*—In this paper, a real-time iterative learning control (ILC) approach for a nonlinear continuous-time system using recurrent neural networks (RNN's) with time-varying weights is presented. Two RNN's are utilized in the ILC system. One is used to approximate the nonlinear system and another is used to mimic the desired system response. The ILC rule is obtained by combining the two RNN's to form a neural network control system. Also, a kind of iterative RNN's training algorithm is developed based on the two-dimensional (2-D) system theory. An RNN using the proposed 2-D training algorithm is able to approximate any trajectory to a very high degree of accuracy. Simulation results show that the proposed ILC approach is very efficient. The newly developed 2-D RNN's training algorithms provides a new dimension to the application of RNN's in a nonlinear continuous-time system.**

***Index Terms*—Approximation, continuous-time iterative learning control, real-time training algorithm, recurrent neural networks, two-dimensional system.**

I. Introduction

ITERATIVE learning control (ILC) is an approach to improve the transient response of a system operating repetitively over a fixed time interval. The objective is to determine a control input iteratively so that the tracking of a given reference signal or the output trajectory over a fixed time interval is possible. As a result, the output accuracy is progressively increased. This makes the application of the ILC approach increasingly important in many control applications, such as robot manipulators. Until now, the most widely used ILC algorithm is the proportional-plus-integral-plus-derivative (PID)-type approach because it essentially forms a PID-like system. Despite the immense popularity of the PID-type controller, all PID-type ILC techniques suffer from a tight restriction [5]. The understanding of the structure and parameters of an unknown system cannot be directly increased through the PID-type learning approach, as it is difficult to generalize the obtained results from a particular task to other similar tasks. Apart from the PID-type controller, there are many other types of controller derived for the ILC approach. However, all these proposed learning algorithms require many learning iterations to achieve the required accuracy. Recently, two-dimensional (2-D) system theory was introduced to the ILC approach [4]–[8]. In the application of 2-D system theory to the ILC technique, very promising results on linear system control have been obtained [4]–[6]. In [7], Chow and Fang extended the discrete-time 2-D ILC control technique to continuous-time 2-D ILC systems. More recently, in [8], Fang and Chow proposed a modified linear discrete-time ILC rule assuring the desired trajectory to be accurately tracked in only one learning iteration. In [9], Chow and Fang worked on the ILC of nonlinear discrete-time system using recurrent neural networks (RNN's) with time-varying weights. In their work, they derived a novel 2-D RNN's training algorithm on which the development of an efficient nonlinear ILC system was based. The developed RNN's based ILC approach for nonlinear discrete-time system can achieve the required accuracy with fewer learning iterations than common nonlinear ILC techniques. Despite its encouraging performance, the developed RNN's based ILC approach in [9] was only applicable to a discrete-time control system. It is the major objective of this paper in extending the RNN's-based ILC approach to nonlinear continuous-time system.

It is generally understood that the selection of the neural network (NN) training algorithm plays an important role for most NN's applications. This notion is especially essential for the implementation of the RNN's-based real-time ILC approach. The shortcomings of using conventional gradient-descent-type training algorithms were thoroughly discussed in [9]. In the conventional gradient-descent-type approach, the gradient is determined under the assumption that the weights do not vary with time. In fact, we must admit that weights are time varying during the training process. As a result, 2-D system theory was first introduced to RNN's-based ILC approach [9]. The 2-D system theory provides a new dimension for the study of NN's. Contributing by the two independent dynamic processes of the 2-D system, the 2-D model provides an excellent mathematical platform to describe the complicated system dynamics. In the training process of continuous-time RNN's, there are two independent dynamics, namely, the continuous-time variable of the RNN's and the iterations variable which is discrete in the training process of the RNN's. In this sense, the 2-D continuous discrete-time system theory, originally introduced by Kaczorek [1], [3] in 1994, is exploited for the development of the real-time RNN's-based continuous-time ILC approach. The 2-D RNN's training algorithms are always convergent and exhibit an excellent convergence rate. In this paper, we show that the

Manuscript received July 16, 1998; revised September 4, 1999. Abstract published on the Internet December 23, 1999.

T. W. S. Chow is with the Department of Electronic Engineering, City University of Hong Kong, Kowloon, Hong Kong.

X.-D. Li is with the Department of Physics and Electrical Information Engineering, Ningxia University, Yingchuan, China.

Y. Fang was with the Department of Electronic Engineering, City University of Hong Kong, Kowloon, Hong Kong. He is now with the School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore.

Publisher Item Identifier S 0278-0046(00)02494-1.

newly developed continuous-time RNN working together with the proposed 2-D training algorithm is capable of approximating any continuous-time trajectory to a very high degree of accuracy within a few iterations.

In this paper, our proposed RNN's-based ILC approach for a nonlinear continuous-time system utilizes two RNN's of the same network architecture. One RNN is used to approximate the nonlinear system, while another one is used to mimic the desired output. The learning rule of the ILC is implemented by combining the two RNN's to form the NN control system. In the ILC processing, the two RNN's are trained by the proposed 2-D training algorithm. Because the development of the ILC approach is based on the proposed 2-D RNN's training algorithms, we will first derive the 2-D continuous-time RNN's training algorithms with time-varying weights. The obtained results show that the performance of our proposed continuous-time ILC approach is very promising. A very low error level is achieved within a few learning iterations.

The organization of this paper is as follows. First, the 2-D representation of the training of a continuous-time RNN is described in Section II. In Section III, we derive two real-time iterative training algorithms for different forms of RNN's. The RNN's-based ILC approach together with its simulation results are presented inSection IV. Finally, Section V concludes this paper.

II. 2-D Representation of RNN's Training

In this paper, we consider the continuous-time RNN's. A general expression of these types of RNN's with L neural units is given by the following differential equation:

$$\frac{dx}{dt} = -\alpha x + f\,(W_1,\, x,\, W_2,\, u) \tag{1}$$

where $x = (x_1, x_2, \cdots, x_L) \in R^L$ is the neural state, and $u = (u_1, u_2, \cdots, u_m) \in R^m$ is the input vector, $L = n + N$, n is the number of output neurons, and N is the number of hidden neurons, $W_1 \in R^{L\times L}$ and $W_2 \in R^{L\times m}$ are the connection weight matrices associated with the neural state and the input vector, respectively, α is a time constant and is chosen as $\alpha > 0$, and f: $R^L \times R^m \to R^L$ is a vector-valued nonlinear function.The popular choices of the nonlinear function f are given as follows:

1) $f(W_1, x, W_2, u) = W_1\sigma(x) + W_2u$
2) $f(W_1, x, W_2, u) = \sigma(W_1x + W_2u)$
3) $f(W_1, x, W_2, u) = \sigma(W_1x) + W_2u$

where the neural activation function $\sigma(x)$ is usually chosen as a continuous and differentiable nonlinear sigmoidal function satisfying the following conditions: 1) $\sigma(x) \to \pm 1$ as $x \to \pm\infty$; 2) $\sigma(x)$ is bounded with the upper bound 1 and the lower bound -1; 3) $\sigma(x) = 0$ at a unique point $x = 0$; and 4) $0 < \sigma'(x)1$ and $\sigma'(x) \to 0$ as $x \to \pm\infty$.

In Section III, our work will be mainly based on the two types of continuous-time RNN's (1) with the $f(W_1, x, W_2, u)$ described in 1) and 2). Fig. 1 shows a typical structure of these types of RNN's. Two different training algorithms for these two types of RNN's are developed. The derivations of these training algorithms are all based on the 2-D continuous discrete-time

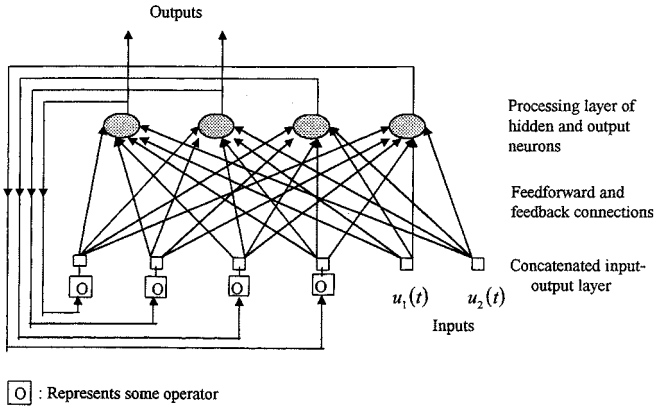

Fig. 1. Architecture graph of proposed RNN's.

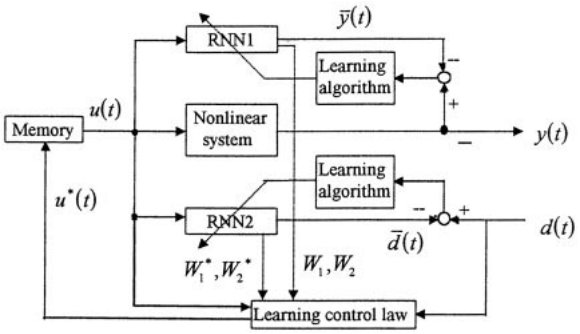

Fig. 2. General structure of the proposed learning control scheme with two RNN's ($u^*(t)$ denotes the new control input).

system theory. For other forms of $f(W_1, x, W_2, u)$, such as the one described in 3), the derivation processes of 2-D RNN's training algorithms are largely similar and are not included in this paper.

In this paper, we consider the training of an RNN with time-varying weights consisting of two dynamical processes in terms of its time variable t and its number of training iterations k. Each variable of an RNN depends upon the two independent dynamics. For example, $W_1(t, k)$ and $W_2(t, k)$ represent the RNN weights, and $x(t, k)$ represents the neural state vector in time t and kth training iteration. Using the 2-D notations, the RNN model (1) can be rewritten as

$$\frac{\partial x(t, k)}{\partial t} = -\alpha x(t, k) + f\,(W_1(t, k), x(t, k), W_2(t, k), u(t)). \tag{2}$$

Equation (2) is a 2-D dynamical system, which clearly describes the 2-D dynamical behavior of the real-time iterative training of an RNN. The objective of the real-time training is to match the states of output neurons to the desired values at each time t. For each time t, the error between the output state of RNN and the desired response, the current neural state $x(t, k)$, and the weights $W_1(t, k)$ and $W_2(t, k)$ at the kth execution of the training algorithm are recorded. Subsequently, the network weights are adjusted with the aim of reducing the error in the $k + 1$th training iteration. Theoretically, we should be able to drive the output of the RNN to the desired response at time t after a number of training iterations.

Let Ω denote a set of n output neurons, and let $d_i(t)$, which is independent of k, denote the desired response of output neuron i at time t. A 2-D L-by-1 error vector $e(t, k)$ is defined as follows:

$$e(t, k) = (e_1(t, k), e_2(t, k), \cdots, e_L(t, k))' \tag{3}$$

where

$$e_i(t, k) = \begin{cases} d_i(t) - x_i(t, k), & \text{if } i \in \Omega \\ 0, & \text{otherwise.} \end{cases}$$

To simplify the expression, we suppose all neurons are output neurons, namely, $n = L$, then we have

$$e(t, k) = d(t) - x(t, k) \tag{4}$$

where $d(t) = (d_1(t), d_2(t), \cdots, d_n(t))$. The extension to the general case is straightforward.

At every training iteration, the training process of updating weights can be expressed as

$$\begin{aligned} W_1(t, k+1) &= W_1(t, k) + \Delta W_1(t, k) \\ W_2(t, k+1) &= W_2(t, k) + \Delta W_2(t, k) \end{aligned} \tag{5}$$

where $\Delta W_1(t, k)$ and $\Delta W_2(t, k)$ are the training rules adjusting the network weights at time t from $W_1(t, k)$ to $W_1(t, k+1)$, and from $W_2(t, k)$ to $W_2(t, k+1)$.

For each k, let the state variable of the 2-D dynamical system (2) begin with the same nonzero initial value, namely,

$$x_i(0, k) = x_i(0) = d_i(0) \neq 0,\ i \in \Omega, \qquad k = 0, 1, 2, \cdots. \tag{6}$$

Let the initial weights at each time t be randomized from a uniform distribution

$$W_1(t, 0) = W_{10}(t) \quad W_2(t, 0) = W_{20}(t), \qquad t \in [0, \infty). \tag{7}$$

Equations (6) and (7) are the boundary conditions of the 2-D system (2). Under the initial conditions (6) and (7), it is essential to assure that the error $e(t, k)$ approaches zero when the number of training iterations increases. We have the following definition.

Definition 1: The training rule (5) is said to be convergent if

$$e(t, k) \to 0, \qquad \text{for } t \in [0, \infty), \text{ as } k \to \infty \tag{8}$$

for any initial boundary conditions (6) and (7).

III. 2-D Training Algorithms for RNN's

In this section, we first express the error equation and the neural state equation of a continuous-time RNN in the form of a 2-D continuous-discrete Roesser's model [3]. Sufficient condition of the convergence of the Roesser's model state is derived. Also, real-time iterative training rules for two typical forms of RNN's (2) are included.

For the RNN in (2), let

$$\eta(t, k) = \int_0^t [x(t, k+1) - x(t, k)]\, dt \tag{9}$$

then $\eta(0, k) = 0$, and

$$\begin{aligned} \frac{\partial \eta(t, k)}{\partial t} &= x(t, k+1) - x(t, k) \\ &= x(t, k+1) - x(0, k+1) - x(t, k) + x(0, k) \\ &= \int_0^t \frac{\partial x(\tau, k+1)}{\partial \tau}\, d\tau - \int_0^t \frac{\partial x(\tau, k)}{\partial \tau}\, d\tau \\ &= -\alpha \int_0^t x(\tau, k+1)\, d\tau + \int_0^t f\,(W_1(\tau, k+1), \\ &\quad x(\tau, k+1), W_2(\tau, k+1), u(\tau))\, d\tau \\ &\quad + \alpha \int_0^t x(\tau, k)\, d\tau - \int_0^t f\,(W_1(\tau, k), x(\tau, k), \\ &\quad W_2(\tau, k), u(\tau))\, d\tau \\ &= -\alpha\eta(t, k) + \int_0^t [f\,(W_1(\tau, k+1), x(\tau, k+1), \\ &\quad W_2(\tau, k+1), u(\tau)) - f\,(W_1(\tau, k), x(\tau, k), \\ &\quad W_2(\tau, k), u(\tau))]\, d\tau. \end{aligned} \tag{10}$$

From (3) and (4), we obtain

$$\begin{aligned} &e(t, k+1) - e(t, k) \\ &\quad = x(t, k) - x(t, k+1) \\ &\quad = -\frac{\partial \eta(t, k)}{\partial t} \\ &\quad = \alpha\eta(t, k) - \int_0^t [f\,(W_1(\tau, k+1), x(\tau, k+1), \\ &\qquad W_2(\tau, k+1), u(\tau)) - f\,(W_1(\tau, k), x(\tau, k), \\ &\qquad W_2(\tau, k), u(\tau))]\, d\tau. \end{aligned} \tag{11}$$

If we let

$$\begin{aligned} &f\,(W_1(t, k+1), x(t, k+1), W_2(t, k+1), u(t)) \\ &\qquad - f\,(W_1(t, k), x(t, k), W_2(t, k), u(t)) \\ &\quad = K_1 \frac{\partial \eta(t, k)}{\partial t} + K_2 \frac{\partial e(t, k)}{\partial t} \end{aligned} \tag{12}$$

by combining (10) with (11), and using (12), we can obtain a 2-D continuous discrete Roessor's model

$$\begin{aligned} \begin{pmatrix} \dfrac{\partial \eta(t, k)}{\partial t} \\ e(t, k+1) \end{pmatrix} &= \begin{pmatrix} -\alpha I + K_1 & K_2 \\ \alpha I - K_1 & I - K_2 \end{pmatrix} \begin{pmatrix} \eta(t, k) \\ e(t, k) \end{pmatrix} \\ &\quad + \begin{pmatrix} -K_1 & -K_2 \\ K_1 & K_2 \end{pmatrix} \begin{pmatrix} \eta(0, k) \\ e(0, k) \end{pmatrix} \end{aligned} \tag{13}$$

where I is identity matrix with appropriate dimension, and $\eta(0, k) = 0$, for $k = 0, 1, 2, 3, \cdots$. From (3), (4), and (6), we also know $e(0, k) = 0$, for $k = 0, 1, 2, 3, \cdots$. Thus, we have

$$\begin{pmatrix} \dfrac{\partial \eta(t, k)}{\partial t} \\ e(t, k+1) \end{pmatrix} = \begin{pmatrix} -\alpha I + K_1 & K_2 \\ \alpha I - K_1 & I - K_2 \end{pmatrix} \begin{pmatrix} \eta(t, k) \\ e(t, k) \end{pmatrix} \tag{14}$$

where the boundary conditions are $\eta(0, k) = 0$ for $k = 0, 1, 2, 3\cdots$ and $e(t, 0) = d(t) - x(t, 0)$ for $t \in [0, \infty)$.

Theorem 1: For a 2-D continuous discrete-time system

$$\begin{pmatrix} \dfrac{\partial x(t,k)}{\partial t} \\ y(t,k+1) \end{pmatrix} = \begin{pmatrix} A_1 & A_2 \\ A_3 & A_4 \end{pmatrix} \begin{pmatrix} x(t,k) \\ y(t,k) \end{pmatrix} \tag{15}$$

where $x(t,k) \in R^{n_1}$, $y(t,k) \in R^{n_2}$, $A_1 \in R^{n_1 \times n_1}$, $A_2 \in R^{n_1 \times n_2}$, $A_3 \in R^{n_2 \times n_1}$ and $A_4 \in R^{n_2 \times n_2}$, and $x(0,k) = 0$ are the boundary conditions for $k = 1, 2, 3, \cdots$ and $y(t,0)$ for $t > 0$. If the matrix A_4 is stable, then, for all $t > 0$,

$$\begin{pmatrix} x(t,k) \\ y(t,k) \end{pmatrix} \to 0 \quad \text{as } k \to \infty.$$

The proof can be referred to [7, Proof of Theorem 1].

It is now required to derive the training algorithms for two different forms of $f(W_1(t), x(t), W_2(t), u(t))$.

1):

$$\begin{aligned} &f\left(W_1(t), x(t), W_2(t), u(t)\right) \\ &\quad = W_1(t)\sigma(x(t)) + W_2(t)u(t). \end{aligned} \tag{16}$$

In (12), substituting function f with (16), and using (5), we have

$$\begin{aligned} &W_1(t,k)\sigma\left(x(t,k+1)\right) + \Delta W_1(t,k)\sigma\left(x(t,k+1)\right) \\ &\quad - W_1(t,k)\sigma(x(t,k)) + \Delta W_2(t,k)u(t) \\ &\quad = K_1 \frac{\partial \eta(t,k)}{\partial t} + K_2 \frac{\partial e(t,k)}{\partial t}. \end{aligned} \tag{17}$$

It is necessary to select matrices K_1 and K_2 in order that (14) satisfies Theorem 1. Subsequently, using (17), the training rule ΔW_1 and ΔW_2 can be determined. From (14) and Theorem 1, it is noticed that the convergence of $e(t,k)$ is independent of the matrix K_1. For simplicity, we select $K_1 = 0$ and $K_2 = I$. Subsequently, from (17), we are able to obtain the following real-time training algorithm.

Training Algorithm 1:

$$\begin{aligned} &(\Delta W_1(t,k)\Delta W_2(t,k)) \\ &= \left(\frac{\partial e(t,k)}{\partial t} + W_1(t,k)\left(\sigma\left(x(t,k)\right) - \sigma\left(x(t,k+1)\right)\right)\right) \\ &\quad \cdot \left(\begin{pmatrix} \sigma\left(x(t,k+1)\right) \\ u(t) \end{pmatrix}^T \begin{pmatrix} \sigma\left(x(t,k+1)\right) \\ u(t) \end{pmatrix}\right)^{-1} \\ &\quad \cdot \begin{pmatrix} \sigma\left(x(t,k+1)\right) \\ u(t) \end{pmatrix}^T \end{aligned} \tag{18}$$

2:

$$\begin{aligned} &f\left(W_1(t), x(t), W_2(t), u(t)\right) \\ &\quad = \sigma\left(W_1(t)x(t) + W_2(t)u(t)\right). \end{aligned} \tag{19}$$

In (12), substituting function f with (19), and using (5), we have

$$\begin{aligned} &\sigma\left(W_1(t,k+1)x(t,k+1) + W_2(t,k+1)u(t)\right) \\ &\quad - \sigma\left(W_1(t,k)x(t,k) + W_2(t,k)u(t)\right) \\ &\quad = F(t,k)\left[W_1(t,k+1)x(t,k+1) + W_2(t,k+1)u(t)\right. \\ &\quad\quad \left. - W_1(t,k)x(t,k) - W_2(t,k)u(t)\right] \\ &\quad = F(t,k)\left[W_1(t,k)\left(x(t,k+1) - x(t,k)\right)\right. \\ &\quad\quad \left. + \Delta W_1(t,k)x(t,k+1) + \Delta W_2(t,k)u(t)\right] \\ &\quad = K_1 \frac{\partial \eta(t,k)}{\partial t} + K_2 \frac{\partial e(t,k)}{\partial t} \end{aligned} \tag{20}$$

where $F(t,k) = \operatorname{diag}(\sigma'(\xi_1), \sigma'(\xi_2), \cdots, \sigma'(\xi_L))$, and $\xi = (\xi_1, \xi_2, \cdots, \xi_L)^T$ is between

$$W_1(t,k+1)x(t,k+1) + W_2(t,k+1)u(t)$$

and

$$W_1(t,k)x(t,k) + W_2(t,k)u(t).$$

$F(t,k)$ is a nonsingular matrix because $\sigma(\,.\,)$ is a sigmoid function with its derivative, $\sigma'(\,.\,)$, satisfies $0 < \sigma'(\,.\,) \leq 1$. Similarly, we select $K_1 = 0$. We are then able to derive Training Algorithm 2 from (20).

Training Algorithm 2:

$$\begin{aligned} &(\Delta W_1(t,k)\Delta W_2(t,k)) \\ &\quad = \left(F^{-1}(t,k)K_2\frac{\partial e(t,k)}{\partial t} + W_1(t,k)\left(x(t,k)\right.\right. \\ &\quad\quad \left.\left. - x(t,k+1)\right)\right)\left(\begin{pmatrix} x(t,k+1) \\ u(t) \end{pmatrix}^T\right. \\ &\quad\quad \left.\cdot \begin{pmatrix} x(t,k+1) \\ u(t) \end{pmatrix}\right)^{-1} \begin{pmatrix} x(t,k+1) \\ u(t) \end{pmatrix}^T. \end{aligned} \tag{21}$$

In (21), despite the fact that $F(t,k)$ has not been determined, we can select $K_2 = F(t,k)$ in order to make Training Algorithm 2 computable. Because $F(t,k) = \operatorname{diag}(\sigma'(\xi_1), \sigma'(\xi_2), \cdots, \sigma'(\xi_L))$ and $0 < \sigma'(\,.\,) \leq 1$, the eigenvalues of matrix $I - F(t,k)$ are all less than 1 and nonnegative. From (14) and Theorem 1, we know that the convergence of Training Algorithm 2 is assured in general cases by putting $K_2 = F(t,k)$.

Also, it is noted that the derived Training Algorithms 1 and 2 are always convergent and are independent of the initial state and the initial weight matrices for any $t \in [0, \infty)$.

Until now, we have derived the 2-D real-time training algorithms for the RNN's with the two most popular forms of $f(W_1(t), x(t), W_2(t), u(t))$. The 2-D RNN's training algorithms for other forms of $f(W_1(t), x(t), W_2(t), u(t))$ can be derived in a similar fashion and, thus, will not be included in this paper.

Now, we can describe clearly our iterative training process: for an RNN represented by (2) and (16) or (2) and (19) with a initial state $x(0) = x_0$, randomized initial weight matrices $W_1(t,0)$ and $W_2(t,0)$, and a given input $u(t)$, the desired output state $d(t)$ at a required tolerance $||e(t)|| < \varepsilon$ can be obtained by updating the weight matrices $W_1(t)$ and $W_2(t)$ by using the appropriate training algorithm. The above process repeats in the next time point t, which can be up to infinity when operating in a real-time mode. Also, the convergence analysis of the real-time training algorithms indicate that the proposed training algorithms enable the RNN to track the desired trajectory to an arbitrary degree of accuracy. The

proposed algorithms are then applied to a nonlinear dynamical system to demonstrate its performance.

Example 1: Consider the problem of approximating a nonlinear dynamical system represented by

$$\frac{d\,p(t)}{dt} = g\,(p(t),\, u(t)) \tag{22}$$

where $u(t)$ is the system input and $p(t)$ is the system output. In this example, the following nonlinear system is selected:

$$\begin{cases} \dfrac{dp_1(t)}{dt} = \sin p_2(t) \\ \dfrac{dp_2(t)}{dt} = p_1^4(t)\cos p_2(t) + u(t) \end{cases}$$

with the initial value $p_1(0) = 0.5$ and $p_2(0) = 1$. The input $u(t)$ is generated by a random function with a magnitude ranged between 0–10. An RNN in the form of

$$\frac{dx(t)}{dt} = -\alpha x(t) + W_1(t)\sigma(x(t)) + W_2(t)u(t)$$

is used for approximating the nonlinear system, where $x(t) = (x_1(t) \;\; x_2(t))^T \in R^2$, $W_1(t) \in R^{2\times 2}$, $W_2(t) \in R^{2\times 1}$. The initial weights of the RNN are randomized between -1 and 1 at each time point t, and the sigmoid nonlinear function is $\sigma(\,.\,) = \tanh(\,.\,)$. We let $\alpha = 0.01$. Using the real-time Training Algorithm 1 represented in (18), the weights of the RNN are adjusted in a real-time form. Figs. 3 and 4 show how the output neuron states $x_1(t)$ and $x_2(t)$, approximating $p_1(t)$ and $p_2(t)$ at the interval $t \in [0,\, 1]$ after only the first iteration. Figs. 5 and 6 show the total squared errors of approximating $p_1(t)$ and $p_2(t)$, respectively, at the interval $t \in [0,\, 1]$ from the first iteration to the sixth iteration. The total squared errors of approximating $p_1(t)$ and $p_2(t)$ are 1.27×10^{-5} and 3.12×10^{-4}, respectively, after the first iteration, while they are only 2.06×10^{-33} and 5.7×10^{-32}, respectively, after the sixth iteration. The result indicates that, at each time point, a very high level of approximation accuracy is obtained within only few iterations. Generally, in handling the approximation of a nonlinear continuous-time system with the proposed 2-D training algorithms, the approximation accuracy can be raised by simply increasing the number of iterations at each time point.

In this example, no hidden neuron is used in the network architecture. Simulations with different numbers of hidden neurons was also performed, but the obtained results indicate that the inclusion of hidden neurons does not have a noticeable effect on the system performance. In terms of the simplest network architecture for practical applications, the number of neurons required can be considered as the same as that of the dimension of the trajectory.

In addition, our simulations have also shown that the convergence of the proposed training algorithm of RNN is not sensitive to the selection of parameter α. The condition of $\alpha > 0$ is to ensure the existence of a solution to (1).

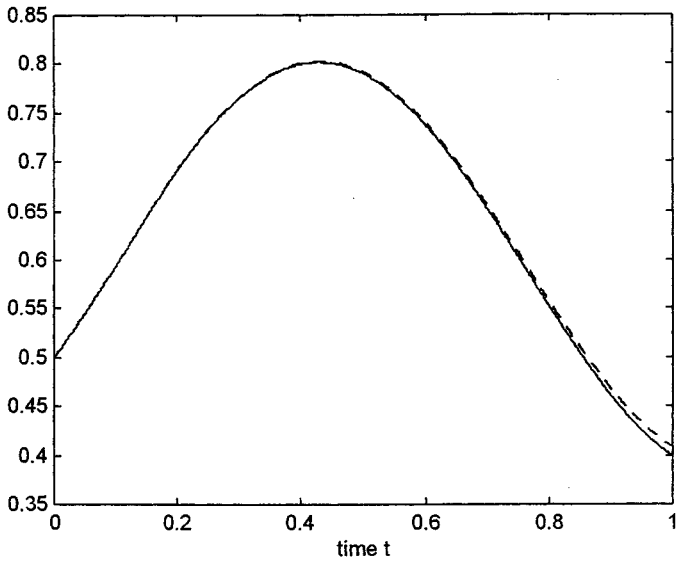

Fig. 3. Example 1: The curve of approximating $p_1(t)$ using output neuron states $x_1(t)$ at the interval $t \in [0,\, 1]$ after only the first iteration. The dashed line represents $x_1(t)$ and the solid line represents $p_1(t)$.

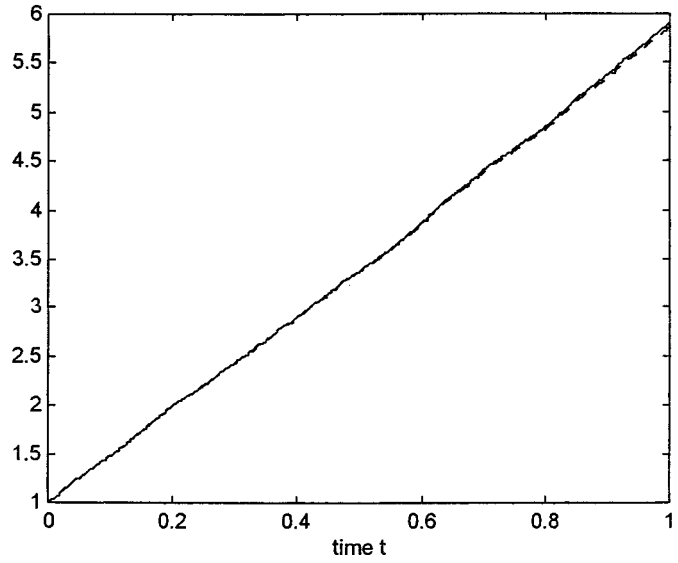

Fig. 4. Example 1: The curve of approximating $p_2(t)$ using output neuron states $x_2(t)$ at the interval $t \in [0,\, 1]$ after only the first iteration. The dashed line represents $x_2(t)$ and the solid line represents $p_2(t)$.

IV. ILC Approach Using RNN's with 2-D Training Algorithms

We have demonstrated that an RNN with time-varying weights is capable of approximating a nonlinear continuous-time system to any degree of accuracy using the proposed 2-D iterative training algorithm. In fact, the application of RNN's in control, identification, and filtering is e mainly based on this approximating capability. In this section, we apply RNN's combined with the proposed 2-D training algorithms to nonlinear ILC systems.

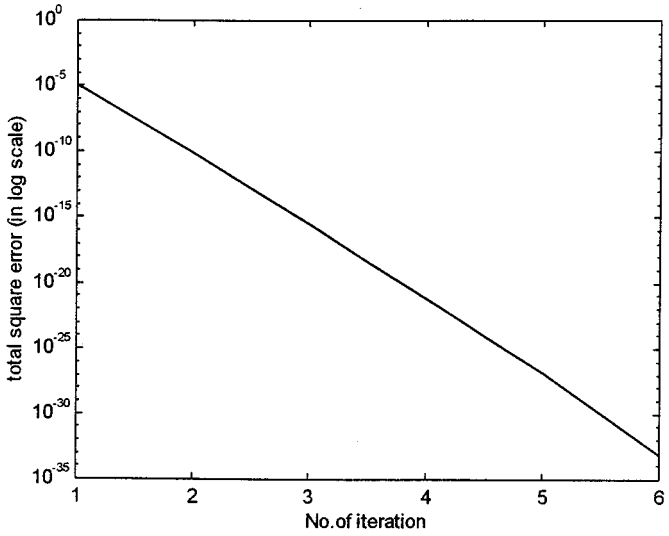

Fig. 5. Example 1: The total square errors of approximating $p_1(t)$ using output neuron states $x_1(t)$ at the interval $t \in [0,\ 1]$ at different number of iteration with the proposed 2-D Training Algorithm 1.

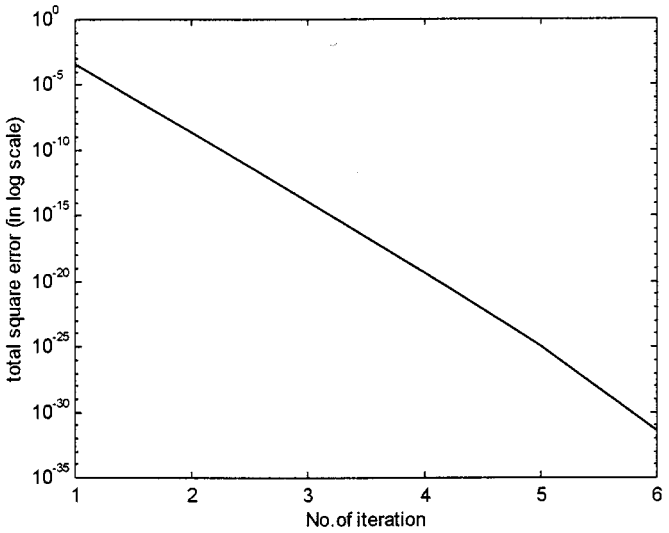

Fig. 6. Example 1: The total square errors of approximating $p_2(t)$ using output neuron states $x_2(t)$ at the interval $t \in [0,\ 1]$ at different number of iteration with the proposed 2-D Training Algorithm 1.

For a general class of nonlinear continuous-time system described as follows:

$$\frac{dy(t)}{dt} = F\left(y(t),\, u(t)\right) \tag{23}$$

where $y(t) \in R^n$ and $u(t) \in R^m$are the output vector and the input vector, respectively, and the desired output $d(t) \in R^n$, $t \in [0, T]$, which is assumed to be differentiable. The initial condition is $y(0) = d(0)$. The objective of ILC is to determine a control input sequence such that the system output tracks the desired output, i.e., $\sup_{t\in[0,T]} |d(t) - y(t)| < \varepsilon$, where ε is a required tolerance. Suppose the RNN's are in the following form:

$$\frac{dx(t)}{dt} = -\alpha x(t) + W_1(t)\sigma(x(t)) + W_2(t)u(t)$$

where $x(t) \in R^n$. In the proposed ILC scheme, we use two RNN's of the same network architecture to approximate the nonlinear system output $y(t)$, and the desired output $d(t)$. In (23), given an initial input $u(t)$, we are able to use one RNN to approximate the nonlinear system (23). Using the real-time Training Algorithm 1, the relationship between the input and the output in (23) at each time point t can be approximately represented as

$$\frac{d\overline{y}(t)}{dt} = -\alpha\overline{y}(t) + W_1(t)\sigma\left(\overline{y}(t)\right) + W_2(t)u(t) \tag{24}$$

where $\overline{y}(t)$ represents the approximation of $y(t)$. Similarly, based on the same input $u(t)$, the approximation of the desired output $d(t)$ is achieved by using another RNN. Suppose the relationship between $u(t)$ and $d(t)$ is approximately represented as

$$\frac{d\,\overline{d}(t)}{dt} = -\alpha\,\overline{d}(t) + W_1^*(t)\sigma\left(\overline{d}(t)\right) + W_2^*(t)u(t) \tag{25}$$

where $\overline{d}(t)$ represents the approximation of $d(t)$. When the number of training iteration is sufficient, the equations

$$\overline{y}(t) = y(t) \quad \text{and} \quad \overline{d}(t) = d(t) \tag{26}$$

holds approximately. At any time point t, once the previous input enables the equation $y(t) = d(t)$, we need to determine an input $u(t)$ such that the variation of output $y(t)$, or $(d\overline{y}(t)/dt)$, equals to the variation of the desired output $d(t)$, or $(d\overline{d}(t)/dt)$. Comparing (24) with (25), and combining (26), if the matrix $W_2(t)$ has the generalized inverse $W_2^g(t)$, we can derive the following ILC rule.

ILC Rule 1:

$$u(t) \Leftarrow W_2^g(t)\left[\left(W_1^*(t) - W_1(t)\right)\sigma(d(t)) + W_2^*(t)u(t)\right]. \tag{27}$$

This ILC rule enables the output $y(t)$ to track the desired output $d(t)$ at the subsequent time point. Because the time step of both the 2-D training algorithm and ILC Rule 1 are in a continuous mode, which have an initial condition of $y(0) = d(0)$, ILC Rule 1 can be executed over the whole fixed time interval. The general structure of the proposed learning control scheme with two RNN's is illustrated in Fig. 2.

Remark: If $y(0) \neq d(0)$, then let $y_1(t) = y(t) - y(0) + d(0)$, we can obtain a nonlinear dynamical system

$$\frac{dy_1(t)}{dt} = F_1\left(y_1(t),\, u(t)\right). \tag{23'}$$

Now, the objective of ILC is to determine a control input sequence such that

$$\sup_{t\in(0,T]} |d(t) - y_1(t)| = \sup_{t\in(0,T]} |d(t) - y(t) + y(0) - d(0)| < \varepsilon.$$

This is similar to the considered problem.

Based on ILC Rule 1, the following algorithm is proposed.

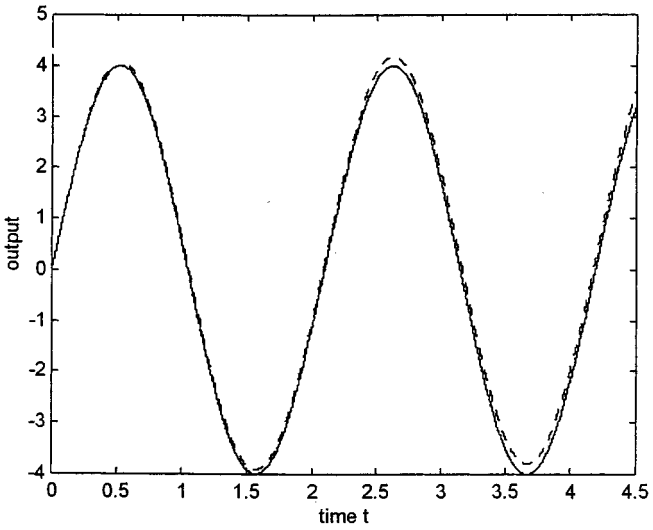

Fig. 7. Example 2: The outputs of the ILC of a nonlinear system using ILC Rule 1. The dashed line represents the control outputs after the fourth iteration and the solid line represents the desired output.

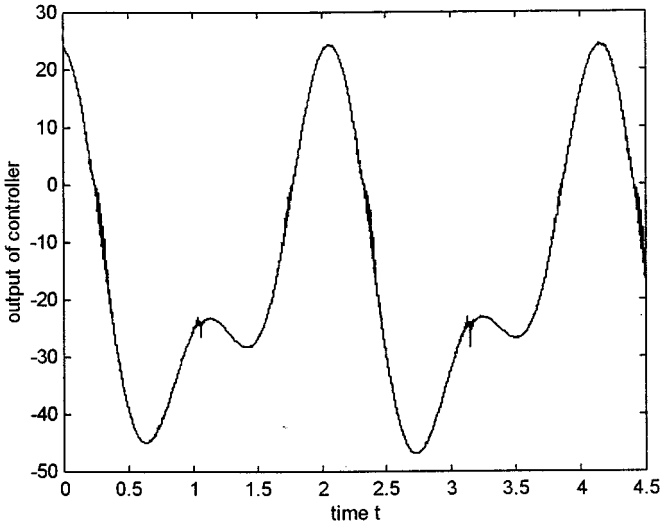

Fig. 8. Example 2: The controller output $u(t)$ using ILC Rule 1 after the fourth iteration.

ILC Algorithm

0) The approximation tolerance ε_1 of RNN's; the learning control tolerance ε; the desired output $d(t)$, $t \in [0, T]$; the initial weights $W_1(t), W_2(t), W_1^*(t)$ and $W_2^*(t)$, $t \in [0, T]$; and the initial condition $y(0) = d(0)$ are given.

1) Let $t = 0$.

2) Initialize input $u(t)$ and evaluate corresponding output $y(t)$.

3) For the input $u(t)$, the corresponding output $y(t)$, and desired output $d(t)$ update the weights $W_1(t), W_2(t), W_1^*(t)$, and $W_2^*(t)$ by the 2-D training algorithm (18).

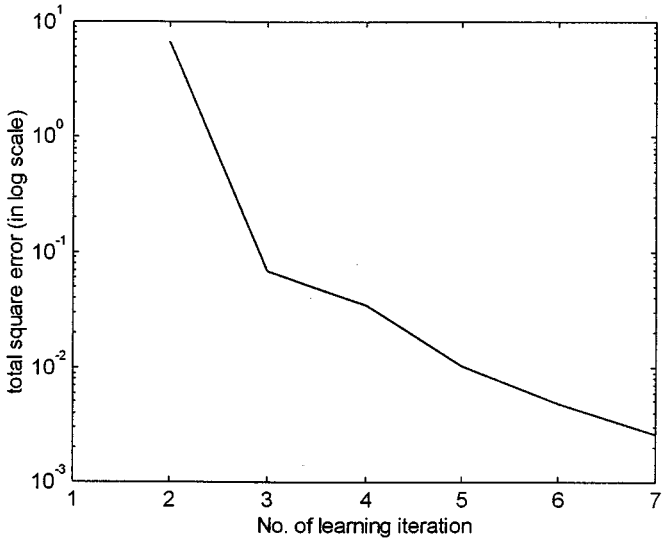

Fig. 9. Example 2: The total square errors at the interval $t \in [0, 3]$ at different numbers of iterations with the proposed ILC Rule 1.

4) If $\|y(t) - \overline{y}(t)\| > \varepsilon_1$ or $\|d(t) - \overline{d}(t)\| > \varepsilon_1$, return to step 3),

5) Using (27) to calculate the new input $u(t)$.

6) For the updated input $u(t)$, evaluate $y(t)$.

7) If $\|y(t) - d(t)\| > \varepsilon$, return to step 3),

8) Let $t = t + \Delta t$.

9) If $t \leq T$, return to step 2), else stop.

Example 2: Consider a nonlinear control system described by the following differential equation

$$\frac{dy(t)}{dt} = y(t) + y^2(t) + 0.5u(t) \tag{28}$$

where the desired output $d(t)$ is described by the equation

$$d(t) = 4\sin(3t).$$

The initial value is $y(0) = 0$. Two RNN's with only one output neuron and one external input connection are used to approximate the nonlinear system (28) and the desired output $d(t)$. In the approximations, 2-D Training Algorithm 1 is uniformly executed for four iterations at each time point. The sigmoid nonlinear function is chosen as $\sigma(.) = \tanh(.)$, and the constant α equals to 0.01. At any time point $t \in [0, 4.5]$, the initial input $u(t)$ and the initial weights $W_1(t)$, $W_2(t)$, $W_1^*(t)$ and $W_2^*(t)$ are randomized between 0 and 1. Using the above ILC algorithm, the control input at each time point is adaptively determined. Fig. 7 shows the tracking performance of the ILC system output, and Fig. 8 shows the controller output $u(t)$ when ILC Rule 1 is iteratively executed for 4 times at each time $t \in [0, 4.5]$. Also, Fig. 9 shows the total squared error of learning control at the interval $t \in [0, 3]$ when ILC Rule 1 is iteratively executed from the second to the seventh times.

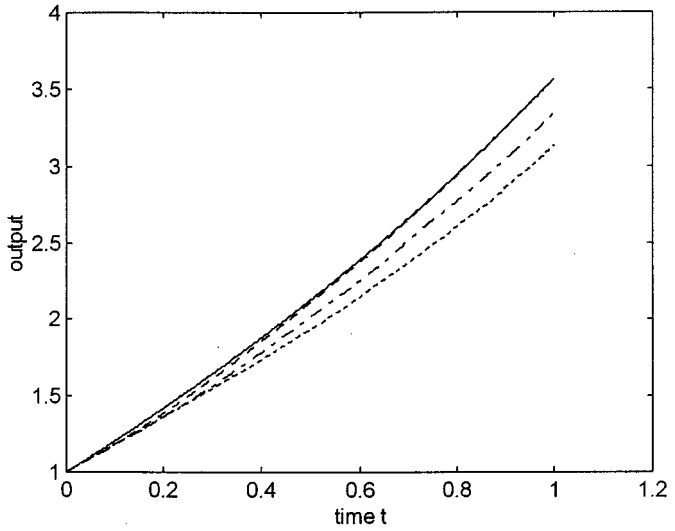

Fig. 10. Example 3: The outputs of the ILC of a nonlinear system using ILC Rule 2. (The dotted, dashed–dotted, and dashed lines represent the control outputs at the third, fourth, and fifth iteration, respectively, and the solid line represents the desired output.)

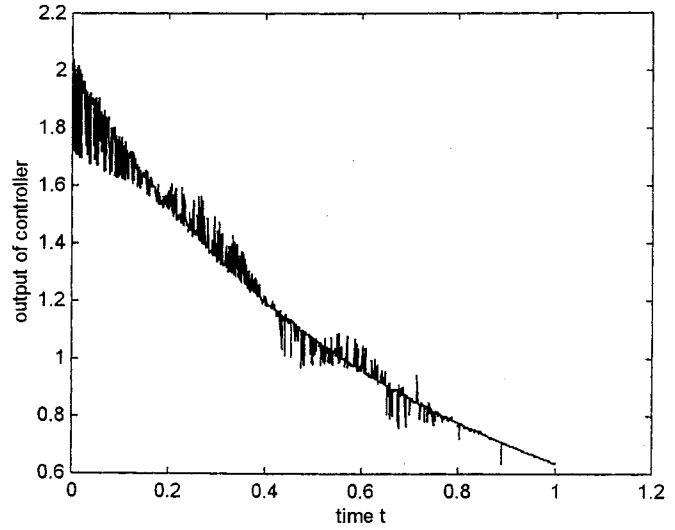

Fig. 11. Example 3: The controller output $u(t)$ using ILC Rule 2 after the fifth iteration.

In addition, if we use the following form of RNN's:

$$\frac{dx(t)}{dt} = -\alpha x(t) + \sigma\left(W_1(t)x(t) + W_2(t)u(t)\right)$$

with 2-D Training Algorithm 2 in the proposed ILC scheme, another ILC rule can be similarly derived as follows.

ILC Rule 2:

$$u(t) \Leftarrow W_2^g(t)\left[\left(W_1^*(t) - W_1(t)\right)d(t) + W_2^*(t)u(t)\right]. \quad (29)$$

Similarly, the ILC algorithm can be derived.

Example 3: Assume the nonlinear ILC system can be described as follows:

$$\frac{dy(t)}{dt} = y(t)u(t) + t^2 \quad (30)$$

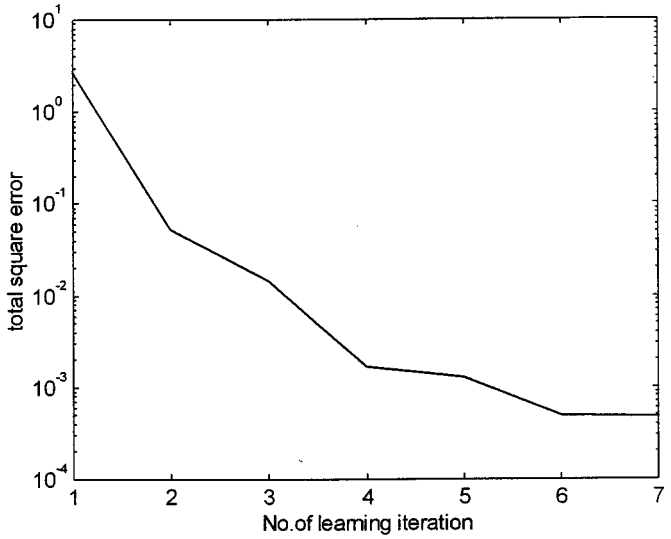

Fig. 12. Example 3: The total square errors at the interval $1 \in [0, 1]$ at different numbers of iterations with the proposed ILC Rule 2.

and the desired output $d(t)$ can be

$$d(t) = e^t + \sin t.$$

The initial value is $y(0) = 1$. Two RNN's with only one output neuron and one external input connection are used to approximate the nonlinear system (30) and the desired output $d(t)$, respectively. In the approximation process, the 2-D Training Algorithm 2 is uniformly executed for six iterations at each time point. The sigmoid nonlinear function is chosen as $\sigma(.) = \tanh(.)$, and the constant α equals to 0.01. At any time point $t \in [0, 1]$, the initial input $u(t)$ and the initial weights $W_1(t)$, $W_2(t)$, $W_1^*(t)$ and $W_2^*(t)$ are randomized between 0 and 1. Using ILC Rule 2, Fig. 10 shows the ILC performance when ILC Rule 2 is executed for three, four, and five times at each time $t \in [0, 1]$, and Fig. 11 shows the controller output $u(t)$ when ILC Rule 1 is iteratively executed for five times at each time $t \in [0, 1]$. Also, Fig. 12 shows the total squared error of learning control at the interval $t \in [0, 1]$ when ILC Rule 2 is iteratively executed from the first times to the eighth times.

Remark: It can be observed from Figs. 8 and 11 that the controller output $u(t)$ sometimes oscillates, which is mainly attributed to the time-variant weights of the RNN's.

Both Examples 2 and 3 show that the proposed ILC Rules 1 and 2 are able to reduce the control error to a very low level within only a few execution cycles. The number of iterations required for the training of the RNN is also a very few. Hidden neurons of RNN's are not required. Compared with the other nonlinear learning control strategies, which usually require many learning iterations to achieve the required accuracy, the convergence rate of the proposed ILC rules is excellent.

V. Conclusion

In this paper, we have successfully extended the discrete-time RNN's-based real-time ILC approach to continuous-time systems. In the proposed continuous-time ILC approach, two

RNN's of identical architecture with time-varying weights are utilized in the ILC system. One is used to approximate the nonlinear system and the other is used to mimic the desired system output. The ILC approach is obtained by combining the two RNN's to form an NN control system. The development of the ILC approach is highly dependent upon RNN's with time-varying weights and its associated 2-D training algorithm.

Conventionally, the weights of RNN's are treated as time invariant. In this paper, we first studied the continuous-time RNN's with time-varying weights. Two real-time training algorithms based on 2-D system theory were then described. The obtained results indicate that the proposed 2-D training algorithms require very few iterations, and the convergence is always assured. The RNN's-based 2-D training algorithm together with our proposed ILC approach forms a very efficient continuous-time nonlinear control methodology which can obtain very high control accuracy with only a few iterations. The obtained results show that the proposed 2-D RNN's training algorithm together with the ILC approach form a very promising control methodology for nonlinear continuous-time systems. Also, the algorithm is very computation efficient, which makes real-time implementation possible.

REFERENCES

[1] T. Kaczorek, "Reachability and controllability of 2-D continuous-discrete linear systems," in *Proc. 1st Int. Symp. Mathematical Models in Automation and Robotics*, Miedzyzdroje, Poland, Sept. 1–3, 1994, pp. 24–28.

[2] ——, "Local controllability and minimum energy control of continuous 2-D linear systems with variable coefficients," in *Multidimensional System and Signal Processing*. Boston, MA: Kluwer, 1995, vol. 6, pp. 69–75.

[3] T. Kaczorek and A. Stajniak, "Local reachability and minimum energy control of 2-D continuous-discrete linear systems," in *Proc. 10th Int. Conf. Systems Engineering*, Coventry, U.K., Sept. 6–8, 1994, pp. 558–565.

[4] Z. Geng and M. Jamshidi, "Learning control system analysis and design based on 2-D system theory," *J. Intell. Robot. Syst.*, vol. 3, pp. 17–26, 1990.

[5] Z. Geng, R. Carroll, and J. Xies, "Two-dimensional model and algorithm analysis for a class of iterative learning control systems," *Int. J. Contr.*, vol. 52, pp. 833–862, 1990.

[6] J. E. Kurek and M. B. Zaremba, "Iterative learning control synthesis based on 2-D system theory," *IEEE Trans. Automat. Contr.*, vol. 38, pp. 121–125, Jan. 1993.

[7] T. W. S. Chow and Y. Fang, "An iterative learning control method for continuous-time systems based on 2-D system theory," *IEEE Trans. Circuits Syst. I*, vol. 45, pp. 683–689, June 1998.

[8] Y. Fang and T. W. S. Chow, "Iterative learning control of linear discrete-time multivariable systems," *Automatica*, to be published.

[9] T. W. S. Chow and Y. Fang, "A recurrent neural network based real-time learning control strategy applying to nonlinear systems with unknown dynamics," *IEEE Trans. Ind. Electron.*, vol. 45, pp. 151–161, Feb. 1998.

Tommy W. S. Chow (M'93) received the B.Sc. (first Hons.) and Ph.D. degrees from the University of Sunderland, Sunderland, U.K.

In 1988, he joined City University of Hong Kong, Kowloon, Hong Kong, as a Lecturer. He is currently an Associate Professor in the Department of Electronic Engineering . His research interests include machine fault diagnosis, HOS analysis, system identification, neural networks learning algorithms, and applications.

Xiao-Dong Li received the B.S. degree from the Department of Mathematics, Shanxi Normal University, Xian, China, in 1987, and the M.Phil. degree from Eastern China Institute of Technology, Nanjing, China, in 1990.

He is currently an Associate Professor in the Department of Physics and Electrical Information Engineering, Ningxia University, Yingchuan, China.

Yong Fang received the B.S. degree in mathematics in 1984 from Sichuan Normal University, Chengdu, China, the M.S. degree from Nanjing University of Science and Technology, Nanjing, China, and the Ph.D. degree in 1999 from the Department of Electronics Engineering, City University of Hong Kong, Kowloon, Hong Kong.

He is currently with the School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore. From 1990 to 1995, he was a Lecturer in the Department of Mathematics, Neijang College, China. His research interests include 2-D system theory, adaptive control, neural networks, and signal processing.