Tables V–VII that C and D are highly evaluated by all clients. The rank of E is middle because it is highly evaluated by the clients except the most credible agent. A and B obtain relatively low evaluations from all clients so that they are ranked poorly.

V. Conclusions

In this paper, new fuzzy aggregation operators based on FDEA are proposed to rank the multi-attribute objects in ACE systems. Different from the other methods which predetermine the weight factors, the proposed model can find out the optimal weight factors by fuzzy linear programming to reflect the inherent preference of clients which give some evaluation on the attributes of objects evaluated. Because each evaluated object tries to find out its weight factors of attributes to its own advantage under the same constraints, the proposed aggregation methods are fair-competition ones with more objectivity for ranking the objects. Three different models are proposed based on fuzzy inequality relation, possibility, and necessity measures, respectively. Given a fuzzy constraint, there are always different explanations on it because a fuzzy constraint is a kind of human thinking featured by ambiguity. These three models are used to characterize the different viewpoints on fuzzy constraints in FDEA models. Generally speaking, the fuzzy constraint in the sense of necessity measure is stronger than that in the sense of possibility measure. Users can choose a suitable model based on his preference.

References

[1] G. Bordogna, M. Fedrizzi, and G. Pasi, "The evaluation of consensus in group decision making: A linguistic approach," *Proc. 7th IFSA World Congress*, vol. 3, pp. 83–88, 1997.

[2] A. Charnes, W. W. Cooper, and E. Rhodes, "Measuring the efficiency of decision making units," *Eur. J. Oper. Res.*, vol. 2, pp. 429–444, 1978.

[3] W. D. Cook and M. Kress, "A data envelopment model for aggregating preference rankings," *Manag. Sci.*, vol. 36, pp. 1302–1310, 1990.

[4] D. Dubois and H. Prade, "Operations on fuzzy numbers," *Int. J. Syst. Sci.*, vol. 9, pp. 613–626, 1978.

[5] ——, *Possibility Theory: An Approach to Computerized Processing of Uncertainty*. New York: Plenum, 1988.

[6] ——, "Possibility theory and data fusion in poorly informed environment," *Contr. Eng. Pract.*, vol. 2, pp. 811–823, 1994.

[7] ——, "An introductory survey of possibility theory and its recent developments," *J. Jpn. Soc. Fuzzy Theory Syst.*, vol. 10, pp. 21–42, 1998.

[8] H. Dyckhoff and W. Pedrycz, "Generalized means as a model of compensative connectives," *Fuzzy Sets Syst.*, vol. 14, pp. 143–154, 1984.

[9] P. Guo and H. Tanaka, "Extended fuzzy DEA," in *Proc. 3rd Asian Fuzzy Syst. Symp.*, 1998, pp. 517–521.

[10] ——, "Fuzzy DEA: A perceptual evaluation method," *Fuzzy Sets Syst.*, to be published.

[11] P. Guo, H. Tanaka, and M. Inuiguchi, "Fuzzy aggregation models for ranking the objects with multiple attributes," in *Proc. 8th Int. Fuzzy Syst. Assoc. World Congress*, 1999, pp. 367–371.

[12] A. Hashimoto, "A ranking voting system using a DEA/AR exclusion model: A note," *Eur. J. Oper. Res.*, vol. 97, pp. 600–604, 1997.

[13] M. Inuiguchi and J. Ramik, "Fuzzy linear programming: A brief survey of techniques and application to portfolio selection," *Fuzzy Sets Syst.*, vol. 111, pp. 3–28, 2000.

[14] K. Leung and Z. Wang, "A new nonlinear integral used for information fusion," in *Proc. IEEE—Fuzzy'98*, 1998, pp. 802–807.

[15] J. Marichal, "On sugeno integral as an aggregation function," *Proc. 6th Eur. Congr. Intelligent Techniques and Soft Computing (EUFIT'98)*, vol. 1, pp. 540–544, 1998.

[16] J. Ramik and J. Rimanek, "Inequality relation between fuzzy numbers and its use in fuzzy optimization," *Fuzzy Sets Syst.*, vol. 16, pp. 123–138, 1985.

[17] R. R. Yager, "On ordered weighted averaging aggregation operators in multi-criteria decision making," *IEEE Trans. Syst., Man, Cybern.*, vol. 18, pp. 183–190, 1988.

[18] ——, "Families of OWA operators," *Fuzzy Sets Syst.*, vol. 59, pp. 125–148, 1993.

Recurrent Learning Algorithms for Designing Optimal Controllers of Continuous Systems

Yi-Jen Wang and Chin-Teng Lin

***Abstract*—This paper proposes the recurrent learning algorithm for designing the controllers of continuous dynamical systems in the optimal control problems. The designed controllers are in the form of unfolded recurrent neural networks embedded with physical laws coming from the classical control techniques. The proposed learning algorithm is characterized by its *double-forward-recurrent-loops* structure for solving both the *temporal recurrent* and the *structure recurrent* problems. The first problem is resulted from the nature of general optimal control problems, where the objective functions are often related to (evaluated at) some specific (instead of all) time steps or system states only, causing missing learning signals at some time steps or system states. The second problem is due to the high-order discretization of the continuous systems by the Runge–Kutta method that we perform to increase the control accuracy. This discretization transforms the system into several identical subnetworks interconnected together, like a recurrent neural network expanded in the time axis. Two recurrent learning algorithms with different convergence properties are derived; the first- and second-order learning algorithms. The computations of both algorithms are local and performed efficiently as network signal propagation. We also propose two new nonlinear controller structures for two specific control problems:1) two-dimensional (2-D) guidance problem and 2) optimal PI control problem. Under the training of the proposed recurrent learning algorithms, these two controllers can be easily tuned to be suboptimal for given objective functions. Extensive computer simulations have shown the optimization and generalization abilities of the controllers designed by the proposed learning scheme.**

I. Introduction

In optimal control problems, the theory of calculus of variations is often used to find the necessary conditions for designing optimal controllers [1]. However, this approach requires precise system model with extra system states (called "costates") and relies heavily on the nonlinear programming techniques, resulting in high computational complexity. To attack these problems, neuro-controllers have recently become popular for suboptimal control of discrete-time dynamical systems [2]–[4]. Although the controller designed by the theory of calculus of variations is globally optimal, it requires precise modeling of the controlled system. This is difficult, if not impossible, to attain in the real world, and thus the suboptimal control is enough for most practical applications. The existing neuro-controllers are usually designed by applying the backpropagation (BP) learning algorithm [5] on feedforward neural networks. However, since the objective function of an optimal control problem is often related to (evaluated at) some specific (instead of all) time steps or system states (in words of neural learning, only the critics or desired values of some system states at some time steps are available for controller evaluation and learning), the normal supervised learning schemes such as BP cannot be used directly in such cases. We call this difficulty of learning as *time-recurrent* learning problem.

Several researchers applied the learning algorithms of recurrent neural networks for designing optimal controllers to solve the time-recurrent learning problem [6]–[9]. The concept is to expand the whole system [including the controller and plant model (see

Manuscript received March 10, 1999; revised May 1, 2000. This work was supported by the National Science Council, R.O.C., under Grant NSC 89-2622-E-009-003. This paper was recommended by Associate Editor J. Oommen.

The authors are with the Department of Electrical and Control Engineering, National Chiao-Tung University, Hsinchu, Taiwan, R.O.C. (email: ctlin@fnn.cn.nctu.edu.tw.)

Publisher Item Identifier S 1083-4427(00)07054-5.

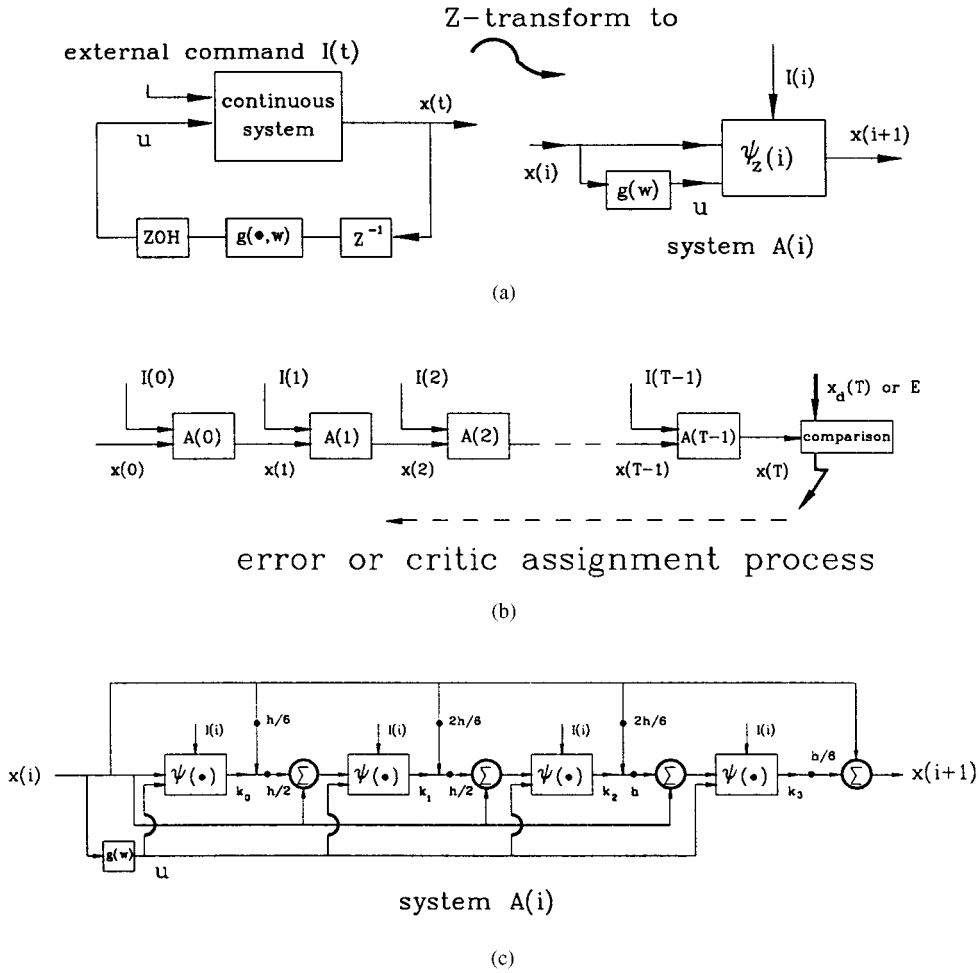

Fig. 1. (a, left) Block diagram of a continuous system with a zero-order holder (ZOH) and a controller [$g(\cdot)$]. (a, right) Block diagram of the system in (a, left) when the continuous system is discretized by Z-transform [denoted by $\psi_z(i)$]. (b) The whole system in (a, right) or in (c) [denoted by $A(i)$] expanded in time sequences in the temporally optimal control problem. (c) Block diagram of the system in (a, left) when the continuous system is discretized by the four-order Runge–Kutta method, where $\psi(\cdot)$ is the describing function of the continuous system.

Fig. 1(a)] in the time axis as shown in Fig. 1(b). Since the desired (critic) values of some system states are available only at specific time steps [for example, the desired value is available only at the final time step T in Fig. 1(b)], a recurrent learning algorithm is required to assign the critic value to each time step properly for tuning the parameters of the controller. In the existing recurrent neural learning approaches for optimal control, the controlled continuous systems are usually discretized by the Z-transform, and the gradients of error (objective) function are obtained by using the gradient descent method recurrently to tune the neuro-controllers. The shortcomings along with these approaches are the bad modeling precision of Z-transform discretization and low convergence speed of gradient descent methods. Although higher order system modeling and higher order learning scheme can enhance the control accuracy, the corresponding recurrent learning algorithm is not easy to derive. Another disadvantage of the normal neuro-controllers is their bad generalization ability for exterpolation [10] due to the "black-box" structure of normal neural networks without physical laws embedded. This makes the normal neuro-controllers less robust. These shortcomings need to be solved in practical applications, and they are the points we want to attack in this paper.

To design the optimal controller for a continuous system in this paper, we shall first apply the classical control techniques (such as the normal optimal control theory and PID control) to form the basic structure of the controller, in which the gains are viewed as unknown parameters. These gains are then tuned to become optimal parameters for a given objective function of the optimal control problem through a neural-network-type recurrent learning algorithm derived in this paper. Such kind of optimal controllers is quite robust, since they embed physical laws in their network structure [11]. For practical applications, we consider a controller with zero-order holder. To obtain higher control accuracy, we apply the Runge–Kutta method [12] to perform high-order discretization of the continuous plant model directly, and then view the discretized system as a special neural network [see Fig. (1)]. Since this discretization process results in several identical subnetworks interconnected in the way described by the Runge–Kutta algorithm, the formed network is like a recurrent neural network expanded in the time axis. The outputs of the subnetworks form the internal states of the designed optimal controller, which are analogous to the *costates* in the theory of calculus of variations approach from the computational complexity point of view. However, instead of being defined explicitly in the latter approach, the internal states are obtained from the system states directly through neural learning in our approach. The learning of the network formed by the Runge–Kutta method is identified as the *structure-recurrent* learning problem. Due to both the structure-recurrent and time-recurrent learning problems faced by our optimal controller, the tuning of the controller gains (parameters) is a recurrent learning problem. Hence the kernel of

this paper is to derive a recurrent learning algorithm for this learning task. With this learning algorithm, an optimal controller with good generalization (robustness) capability and high control accuracy can be obtained for a continuous system.

There are two approaches to deriving the recurrent learning rules; *forward* and *backward* recurrent methods [6]. We adopt the forward recurrent method in this paper for its less requirement of computer storage and easy implementation of software program. In deriving the forward recurrent learning rules, we first use the steepest descent method [5] for its simple arithmetic operations to get a first-order recurrent learning algorithm. Since the convergence speed of this learning algorithm is quite low, we further derive a second-order recurrent learning algorithm based on the order derivative method [13] to speed up the convergence. The basic concept of this second-order learning algorithm is to find the curvature of the error surface for tuning the learning constants in the algorithm adaptively. This is very similar to the Newton's method, but avoids the complexity of finding the inverse of Hessian matrix. Although similar neural learning algorithms had been proposed previously [5], few is applied to both the structure- and time-recurrent learning.

Another major contribution of this paper is in proposing two new nonlinear controller structures for two specific control problems. They are two-dimensional (2-D) guidance problem and optimal PI control problem. The new controller structure for the first problem need not compute or predict the "time-to-go" (T_{go}) value in a guidance problem. For the second problem, we propose a generalized PI controller which is a nonlinear controller. Under the training of the proposed recurrent learning algorithms, these two controllers can be easily tuned to become suboptimal controllers for given objective functions. Such suboptimal controllers cannot be designed by using the normal linear control theory or algorithms.

The rest of this paper is organized as follows. Section II details the derivation of the first-order and second-order recurrent learning algorithms for the controller of a Runge–Kutta-type discretized dynamical system with zero-order holder. Section III presents three simulation examples to demonstrate the applicability of the proposed learning scheme. These include a one-dimensional (1-D) boundary value problem, a 2-D guidance problem, and the optimal PI control of a nonminimal phase linear dynamical system. In this section, we also compare the convergence property of the two proposed recurrent learning algorithms. Finally, conclusions are made in Section IV.

II. Recurrent Learning Algorithms for Designing Optimal Controller

Consider a time-invariant continuous system with a controller and a zero-order holder described by

$$\dot{\boldsymbol{x}}(t) = \psi(\boldsymbol{x}(t), \boldsymbol{u}(i)), \quad \boldsymbol{u}(i) = g(\boldsymbol{x}(i), \boldsymbol{\omega}) \tag{1}$$

where

- $\boldsymbol{x} \in \Re^n$ system states;
- $\boldsymbol{u} \in \Re^m$ controller outputs that are functions of system states;
- $\boldsymbol{\omega} \in \Re^l$ controller gains;
- i index for sampling time h, i.e., $t \in [ih, (i+1)h]$, $i = 1, \cdots, N$.

Suppose E is the performance index of the system in (1) which is a function of $\boldsymbol{x}$ and $\boldsymbol{\omega}$. For example, E may represent the total system input (controller output) energy, or/and the difference between the final system states (e.g., velocity, position) and their corresponding desired (target) values. The optimal controller design problem for the system in (1) is to find $\boldsymbol{\omega}$ that minimizes E.

In this section, we shall find the order derivatives of E with respective to $\boldsymbol{\omega}$ and then derive recurrent update rules for parameters $\boldsymbol{\omega}$. The derivation is similar to that for recurrent learning algorithms in neural networks [8], [9], [14]. However, to achieve high control accuracy, we discretize the continuous system in (1) to become a four-order Runge–Kutta-type model. This makes the derivation of the recurrent learning rules less straightforward and some attentions and skills need to be paid and applied. Two recurrent learning algorithms with different convergence properties are proposed in the following two Sections II-A and II-B . They are first- and second-order recurrent learning algorithms, respectively.

A. First-Order Recurrent Learning Algorithm

In this section, we apply the four-order Runge–Kutta method to discretize the system described by (1), and find the first-order order derivatives of the system to derive the first-order recurrent learning algorithm. By the gradient descent method based on order derivative (i.e., first-order order derivative), the forward recurrent learning rule for $\boldsymbol{\omega}$ in (1) is

$$\boldsymbol{\omega}_j(i) = \boldsymbol{\omega}_j(i-1) + \eta \frac{\partial^+ E}{\partial \boldsymbol{\omega}_j}, \quad j = 1, \cdots, l$$

where

$$\frac{\partial^+ E}{\partial \boldsymbol{\omega}_j} = \sum_{i=1}^{N} \left(\frac{\partial E}{\partial \boldsymbol{x}(i)} \frac{\partial^+ \boldsymbol{x}(i)}{\partial \boldsymbol{\omega}_j} + \frac{\partial E}{\partial \boldsymbol{u}(i)} \frac{\partial^+ \boldsymbol{u}(i)}{\partial \boldsymbol{\omega}_j} \right) \tag{2}$$

and

$$\frac{\partial^+ \boldsymbol{x}(i)}{\partial \boldsymbol{\omega}_j} = \frac{\partial \boldsymbol{x}(i)}{\partial \boldsymbol{\omega}_j} + \frac{\partial \boldsymbol{x}(i)}{\partial \boldsymbol{x}(i-1)} \frac{\partial^+ \boldsymbol{x}(i-1)}{\partial \boldsymbol{\omega}_j} + \frac{\partial \boldsymbol{x}(i)}{\partial \boldsymbol{u}(i-1)} \frac{\partial^+ \boldsymbol{u}(i-1)}{\partial \boldsymbol{\omega}_j}, \tag{3}$$

$$\frac{\partial^+ \boldsymbol{u}(i)}{\partial \boldsymbol{\omega}_j} = \frac{\partial \boldsymbol{u}(i)}{\partial \boldsymbol{\omega}_j} + \frac{\partial \boldsymbol{u}(i)}{\partial \boldsymbol{x}(i)} \frac{\partial^+ \boldsymbol{x}(i)}{\partial \boldsymbol{\omega}_j}. \tag{4}$$

We now use the four-order Runge–Kutta method to compute $(\partial \boldsymbol{x}(i)/\partial \boldsymbol{x}(i-1))$ and $(\partial \boldsymbol{x}(i)/\partial \boldsymbol{u}(i-1))$ in (3) to obtain the forward recurrent computation rule. From the four-order Runge–Kutta algorithm, we have

$$\boldsymbol{x}(i) = \boldsymbol{x}(i-1) + \tfrac{1}{6} h(\boldsymbol{k}_0 + 2\boldsymbol{k}_1 + 2\boldsymbol{k}_2 + \boldsymbol{k}_3) \tag{5}$$

where

$$\boldsymbol{k}_0 = \psi(\boldsymbol{x}(i); \boldsymbol{\omega}), \quad \boldsymbol{k}_1 = \psi\left(\boldsymbol{x}(i) + \tfrac{1}{2} h\boldsymbol{k}_0; \boldsymbol{\omega}\right),$$
$$\boldsymbol{k}_2 = \psi\left(\boldsymbol{x}(i) + \tfrac{1}{2} h\boldsymbol{k}_1; \boldsymbol{\omega}\right), \quad \boldsymbol{k}_3 = \psi(\boldsymbol{x}(i) + h\boldsymbol{k}_2; \boldsymbol{\omega}).$$

The partial derivative of $\boldsymbol{x}(i)$ with respective to $\boldsymbol{x}(i-1)$ is

$$\frac{\partial \boldsymbol{x}(i)}{\partial \boldsymbol{x}(i-1)} = I + \frac{h}{6} \frac{\partial}{\partial \boldsymbol{x}(i-1)} (\boldsymbol{k}_0 + 2\boldsymbol{k}_1 + 2\boldsymbol{k}_2 + \boldsymbol{k}_3) \tag{6}$$

where

$$\frac{\partial \boldsymbol{k}_0}{\partial \boldsymbol{x}(i-1)} = \psi_{,1}\big|_{(\boldsymbol{x}(i-1), \boldsymbol{u}(i-1))}$$

$$\frac{\partial \boldsymbol{k}_1}{\partial \boldsymbol{x}(i-1)} = \psi_{,1}\big|_{(d_1, \boldsymbol{u}(i-1))} \cdot \left(I + \frac{1}{2} h \frac{\partial \boldsymbol{k}_0}{\partial \boldsymbol{x}(i-1)} \right)$$

$$\frac{\partial \boldsymbol{k}_2}{\partial \boldsymbol{x}(i-1)} = \psi_{,1}\big|_{(d_2, \boldsymbol{u}(i-1))} \cdot \left(I + \frac{1}{2} h \frac{\partial \boldsymbol{k}_1}{\partial \boldsymbol{x}(i-1)} \right)$$

$$\frac{\partial \boldsymbol{k}_3}{\partial \boldsymbol{x}(i-1)} = \psi_{,1}\big|_{(d_3, \boldsymbol{u}(i-1))} \cdot \left(I + h \frac{\partial \boldsymbol{k}_2}{\partial \boldsymbol{x}(i-1)} \right)$$

where $\psi_{,i}$ is the partial derivative of ψ with respective to the ith argument of ψ (i.e., $\psi_{,1} \equiv (\partial\psi/\partial \boldsymbol{x})$, $\psi_{,2} \equiv (\partial\psi/\partial \boldsymbol{u})$),

$$d_j = \boldsymbol{x}(i-1) + \tfrac{1}{2}\, h\boldsymbol{k}_{j-1}, \quad j = 1,\, 2$$

and

$$d_3 = \boldsymbol{x}(i-1) + h\boldsymbol{k}_2.$$

Similarly, the partial derivative of $\boldsymbol{x}(i)$ with respective to $\boldsymbol{u}(i-1)$ is

$$\frac{\partial \boldsymbol{x}(i)}{\partial \boldsymbol{u}(i-1)} = I + \frac{h}{6}\,\frac{\partial}{\partial \boldsymbol{u}(i-1)}(\boldsymbol{k}_0 + 2\boldsymbol{k}_1 + 2\boldsymbol{k}_2 + \boldsymbol{k}_3) \tag{7}$$

where

$$\begin{aligned}
\frac{\partial \boldsymbol{k}_0}{\partial \boldsymbol{u}(i-1)} &= \psi_{,2}\big|_{(\boldsymbol{x}(i-1),\, \boldsymbol{u}(i-1))} \\
\frac{\partial \boldsymbol{k}_1}{\partial \boldsymbol{u}(i-1)} &= \psi_{,1}\big|_{(d_1,\, \boldsymbol{u}(i-1))} \cdot \left(\frac{h}{2}\,\frac{\partial \boldsymbol{k}_0}{\partial \boldsymbol{u}(i-1)}\right) \\
&\quad + \psi_{,2}\big|_{(d_1,\, \boldsymbol{u}(i-1))} \\
\frac{\partial \boldsymbol{k}_2}{\partial \boldsymbol{u}(i-1)} &= \psi_{,1}\big|_{(d_2,\, \boldsymbol{u}(i-1))} \cdot \left(\frac{h}{2}\,\frac{\partial \boldsymbol{k}_1}{\partial \boldsymbol{u}(i-1)}\right) \\
&\quad + \psi_{,2}\big|_{(d_2,\, \boldsymbol{u}(i-1))} \\
\frac{\partial \boldsymbol{k}_3}{\partial \boldsymbol{u}(i-1)} &= \psi_{,1}\big|_{(d_3,\, \boldsymbol{u}(i-1))} \cdot \left(h\,\frac{\partial \boldsymbol{k}_1}{\partial \boldsymbol{u}(i-1)}\right) \\
&\quad + \psi_{,2}\big|_{(d_3,\, \boldsymbol{u}(i-1))}.
\end{aligned}$$

The complete first-order recurrent learning algorithm is summarized as follows. First, for time index i from 1 to N, forward recurrently compute (6) and (7) and then forward recurrently compute (3) and (4) (*double forward recurrent loops*) to obtain $\partial^+\boldsymbol{x}(i)/\partial\omega_j$ and $\partial^+\boldsymbol{u}(i)/\partial\omega_j$ sequences for $j = 1, \cdots, l$. Batch-mode or sequentially collect all the terms $\partial^+\boldsymbol{u}(i)/\partial\omega_j)$, $(\partial^+\boldsymbol{x}(i)/\partial\omega_j)$ for $i = 1, \cdots, N$, and then substitute them into (2) to obtain the gradient directions of E with respective to ω parameter space. It worth pointing out that the computation of order derivatives in the above algorithm [i.e., (6) and (7)] is completed in the running computation cycle and thus they needs not be stored. This avoids the requirement of large computer memory and heavy computation loads accumulated at the final time step N as the cases in backward recurrent learning rules [6].

B. Second-Order Recurrent Learning Algorithm

Generally, the first-order recurrent learning algorithm derived in Section II-A shows low convergence speed. To speed up the convergence, we shall derive a second-order counterpart in this subsection. This second-order recurrent learning algorithm is also a forward recurrent rule. Suppose the objective function, E, is a function of $\boldsymbol{x}$ and $\boldsymbol{u}$. The second-order derivatives of E with respective to ω_j and ω_k are derived as follows, where ω_j and ω_k are any two adjustable parameters of the controller:

$$\begin{aligned}
\frac{\partial^+}{\partial \boldsymbol{\omega}_k}\,\frac{\partial^+ E}{\partial \boldsymbol{\omega}_j} = \frac{\partial^+}{\partial \boldsymbol{\omega}_k}\, F\Bigg(&\frac{\partial^+ \boldsymbol{x}(1)}{\partial \boldsymbol{\omega}_j}, \cdots, \frac{\partial^+ \boldsymbol{x}(N)}{\partial \boldsymbol{\omega}_j}, \frac{\partial^+ \boldsymbol{u}(1)}{\partial \boldsymbol{\omega}_j}, \\
&\cdots, \frac{\partial^+ \boldsymbol{u}(N)}{\partial \boldsymbol{\omega}_j}, \boldsymbol{x}(1), \cdots, \boldsymbol{x}(N) \\
&\boldsymbol{u}(1), \cdots, \boldsymbol{u}(N)\Bigg)
\end{aligned} \tag{8}$$

where $j,\, k = 1, \cdots, l$ and $F = (\partial^+ E/\partial\omega)$.

To compute the curvature of E (i.e., $(\partial^+\partial^+ E/\partial\boldsymbol{\omega}_k\partial\boldsymbol{\omega}_j)$), we first compute $(\partial^+/\partial\omega_j)\boldsymbol{x}(i)$ and $(\partial^+/\partial\omega_j)\boldsymbol{u}(i)$ by (3) and (4). We then take order derivatives of $(\partial^+\boldsymbol{x}(i)/\partial\omega_j)$ and $(\partial^+\boldsymbol{u}(i)/\partial\boldsymbol{\omega}_j)$ with respective to ω_k to obtain

$$\begin{aligned}
&\frac{\partial^+}{\partial \boldsymbol{\omega}_k}\,\frac{\partial^+ \boldsymbol{x}(i)}{\partial \boldsymbol{\omega}_j} \\
&\quad = \frac{\partial^+}{\partial \boldsymbol{\omega}_k}\left(\frac{\partial \boldsymbol{x}(i)}{\partial \boldsymbol{\omega}_j}\right) + \frac{\partial^+}{\partial \boldsymbol{\omega}_k}\left(\frac{\partial \boldsymbol{x}(i)}{\partial \boldsymbol{x}(i-1)}\right)\frac{\partial^+ \boldsymbol{x}(i-1)}{\partial \boldsymbol{\omega}_j} \\
&\qquad + \frac{\partial^+}{\partial \boldsymbol{\omega}_k}\left(\frac{\partial \boldsymbol{x}(i)}{\partial \boldsymbol{u}(i-1)}\right)\frac{\partial^+ \boldsymbol{u}(i-1)}{\partial \boldsymbol{\omega}_j} \\
&\qquad + \frac{\partial \boldsymbol{x}(i)}{\partial \boldsymbol{x}(i-1)}\,\frac{\partial^+}{\partial \boldsymbol{\omega}_k}\left(\frac{\partial^+ \boldsymbol{x}(i-1)}{\partial \boldsymbol{\omega}_j}\right) \\
&\qquad + \frac{\partial \boldsymbol{x}(i)}{\partial \boldsymbol{u}(i-1)}\,\frac{\partial^+}{\partial \boldsymbol{\omega}_k}\left(\frac{\partial^+ \boldsymbol{u}(i-1)}{\partial \boldsymbol{\omega}_j}\right)
\end{aligned} \tag{9}$$

and

$$\begin{aligned}
&\frac{\partial^+}{\partial \boldsymbol{\omega}_k}\left(\frac{\partial^+ \boldsymbol{u}(i)}{\partial \boldsymbol{\omega}_j}\right) \\
&\quad = \frac{\partial^+}{\partial \boldsymbol{\omega}_k}\left(\frac{\partial \boldsymbol{u}(i)}{\partial \boldsymbol{\omega}_j}\right) + \frac{\partial^+}{\partial \boldsymbol{\omega}_k}\left(\frac{\partial \boldsymbol{u}(i)}{\partial \boldsymbol{x}(i)}\right)\frac{\partial^+ \boldsymbol{x}(i)}{\partial \boldsymbol{\omega}_j} \\
&\qquad + \frac{\partial \boldsymbol{u}(i)}{\partial \boldsymbol{x}(i)}\,\frac{\partial^+}{\partial \boldsymbol{\omega}_k}\left(\frac{\partial^+ \boldsymbol{x}(i)}{\partial \boldsymbol{\omega}_j}\right).
\end{aligned} \tag{10}$$

The first term in (9) is zero because $\psi(\boldsymbol{x},\, \boldsymbol{u})$ in (1) is not explicitly function of $\boldsymbol{\omega}$. The first term and first part of the second term in (10) are computed according to (3), (4), and function g of (1). The previous part of the second and third terms in (9) can be computed by finding the order derivatives of (6) and (7) as

$$\begin{aligned}
&\frac{\partial^+}{\partial \boldsymbol{\omega}_k}\left(\frac{\partial \boldsymbol{x}(i)}{\partial \boldsymbol{x}(i-1)}\right) \\
&\quad = \frac{\partial^+}{\partial \boldsymbol{\omega}_k}\left(\frac{h}{6}\,\frac{\partial}{\partial \boldsymbol{x}(i-1)}(\boldsymbol{k}_0 + 2\boldsymbol{k}_1 + 2\boldsymbol{k}_2 + \boldsymbol{k}_3)\right)
\end{aligned} \tag{11}$$

where

$$\begin{aligned}
&\frac{\partial^+}{\partial \boldsymbol{\omega}_k}\left(\frac{\partial \boldsymbol{k}_0}{\partial \boldsymbol{x}(i-1)}\right) \\
&\quad = \frac{\partial^+}{\partial \boldsymbol{\omega}_k}\left(\psi_{,1}\big|_{(\boldsymbol{x}(i-1),\, \boldsymbol{u}(i-1))}\right) \\
&\frac{\partial^+}{\partial \boldsymbol{\omega}_k}\left(\frac{\partial \boldsymbol{k}_1}{\partial \boldsymbol{x}(i-1)}\right) \\
&\quad = \frac{\partial^+}{\partial \boldsymbol{\omega}_k}\left(\psi_{,1}\big|_{(d_1,\, \boldsymbol{u}(i-1))}\right) \cdot \left(I + \frac{1}{2}h\,\frac{\partial \boldsymbol{k}_0}{\partial \boldsymbol{x}(i-1)}\right) \\
&\qquad + \frac{h}{2}\psi_{,1}\big|_{(d_1,\, \boldsymbol{u}(i-1))}\,\frac{\partial^+}{\partial \boldsymbol{\omega}_k}\left(\frac{\partial \boldsymbol{k}_0}{\partial \boldsymbol{x}(i-1)}\right) \\
&\frac{\partial^+}{\partial \boldsymbol{\omega}_k}\left(\frac{\partial \boldsymbol{k}_2}{\partial \boldsymbol{x}(i-1)}\right) \\
&\quad = \frac{\partial^+}{\partial \boldsymbol{\omega}_k}\left(\psi_{,1}\big|_{(d_2,\, \boldsymbol{u}(i-1))}\right) \cdot \left(I + \frac{1}{2}h\,\frac{\partial \boldsymbol{k}_1}{\partial \boldsymbol{x}(i-1)}\right) \\
&\qquad + \frac{h}{2}\psi_{,1}\big|_{(d_2,\, \boldsymbol{u}(i-1))}\,\frac{\partial^+}{\partial \boldsymbol{\omega}_k}\left(\frac{\partial \boldsymbol{k}_1}{\partial \boldsymbol{x}(i-1)}\right)
\end{aligned}$$

$$\frac{\partial^+}{\partial \omega_k}\left(\frac{\partial \boldsymbol{k}_3}{\partial \boldsymbol{x}(i-1)}\right) = \frac{\partial^+}{\partial \omega_k}\left(\psi_{,1}|_{(d_3,\,\boldsymbol{u}(i-1))}\right)\cdot\left(I + h\frac{\partial \boldsymbol{k}_2}{\partial \boldsymbol{x}(i-1)}\right) + h\psi_{,1}|_{(d_3,\,\boldsymbol{u}(i-1))}\frac{\partial^+}{\partial \omega_k}\left(\frac{\partial \boldsymbol{k}_2}{\partial \boldsymbol{x}(i-1)}\right). \tag{12}$$

Similarly, we have

$$\frac{\partial^+}{\partial \omega_k}\frac{\partial \boldsymbol{x}(i)}{\partial \boldsymbol{u}(i-1)} = \frac{\partial^+}{\partial \omega_k}\left(\frac{h}{6}\frac{\partial}{\partial \boldsymbol{u}(i-1)}(\boldsymbol{k}_0 + 2\boldsymbol{k}_1 + 2\boldsymbol{k}_2 + \boldsymbol{k}_3)\right) \tag{13}$$

where

$$\frac{\partial^+}{\partial \omega_k}\left(\frac{\partial \boldsymbol{k}_0}{\partial \boldsymbol{u}(i-1)}\right) = \frac{\partial^+}{\partial \boldsymbol{\omega}_k}\psi_{,2}|_{(\boldsymbol{x}(i-1),\,\boldsymbol{u}(i-1))}$$

$$\frac{\partial^+}{\partial \omega_k}\left(\frac{\partial \boldsymbol{k}_1}{\partial \boldsymbol{u}(i-1)}\right) = \frac{\partial^+}{\partial \boldsymbol{\omega}_k}\psi_{,1}|_{(d_1,\,\boldsymbol{u}(i-1))}\cdot\left(\frac{h}{2}\frac{\partial \boldsymbol{k}_0}{\partial \boldsymbol{u}(i-1)}\right) + \psi_{,1}|_{(d_1,\,\boldsymbol{u}(i-1))}\cdot\frac{h}{2}\frac{\partial^+}{\partial \omega_k}\left(\frac{\partial \boldsymbol{k}_0}{\partial \boldsymbol{u}(i-1)}\right) + \frac{\partial^+}{\partial \boldsymbol{\omega}_k}\psi_{,2}|_{(d_1,\,\boldsymbol{u}(i-1))}$$

$$\frac{\partial^+}{\partial \omega_k}\left(\frac{\partial \boldsymbol{k}_2}{\partial \boldsymbol{u}(i-1)}\right) = \frac{\partial^+}{\partial \boldsymbol{\omega}_k}\psi_{,1}|_{(d_2,\,\boldsymbol{u}(i-1))}\cdot\left(\frac{h}{2}\frac{\partial \boldsymbol{k}_1}{\partial \boldsymbol{u}(i-1)}\right) + \psi_{,1}|_{(d_2,\,\boldsymbol{u}(i-1))}\cdot\frac{h}{2}\frac{\partial^+}{\partial \omega_k}\left(\frac{\partial \boldsymbol{k}_1}{\partial \boldsymbol{u}(i-1)}\right) + \frac{\partial^+}{\partial \boldsymbol{\omega}_k}\psi_{,2}|_{(d_2,\,\boldsymbol{u}(i-1))}$$

$$\frac{\partial^+}{\partial \omega_k}\left(\frac{\partial \boldsymbol{k}_3}{\partial \boldsymbol{u}(i-1)}\right) = \frac{\partial^+}{\partial \boldsymbol{\omega}_k}\psi_{,1}|_{(d_3,\,\boldsymbol{u}(i-1))}\cdot\left(h\frac{\partial \boldsymbol{k}_2}{\partial \boldsymbol{u}(i-1)}\right) + \psi_{,1}|_{(d_3,\,\boldsymbol{u}(i-1))}\cdot h\frac{\partial^+}{\partial \boldsymbol{\omega}_k}\left(\frac{\partial \boldsymbol{k}_2}{\partial \boldsymbol{u}(i-1)}\right) + \frac{\partial^+}{\partial \boldsymbol{\omega}_k}\psi_{,2}|_{(d_3,\,\boldsymbol{u}(i-1))}. \tag{14}$$

In the second-order recurrent learning algorithm derived in the above, the computations are the same as those in the first-order counterpart derived in Section II-B , except the computations of (11), (13), and the first and second terms of (10). The results obtained by (8) indicate the curvature of the E-surface with respective to $\boldsymbol{\omega}$. This curvature information can be used to adapt the learning constant η. The second-order recurrent learning algorithm is a Newton-type algorithm; the update rule for parameters $\boldsymbol{\omega}$ is $\Delta\omega \approx -H^{-1}(\partial E/\partial\omega)$, where the update-stepsize factor H is the Hessian matrix corresponding to the second-order derivative of E [15]. Notice that (9) and (10) not only can be applied to continuous systems described by (1), but it can also be used for discrete-time systems. In the latter case, since we need not discretize the system using the Runge–Kutta method, we only need to change the computations of the Runge–Kutta parts in the proposed algorithms [i.e., (6), (7), (11), and (13)] to the normal (first-order) difference relations. That's, the original double forward recurrent loops is reduced to the single forward recurrent loop. In other words, we only need to change the formed network structure from Fig. 1(c)–(a) and then apply the network outputs to (9) and (10). The rest of computations are the same as the original.

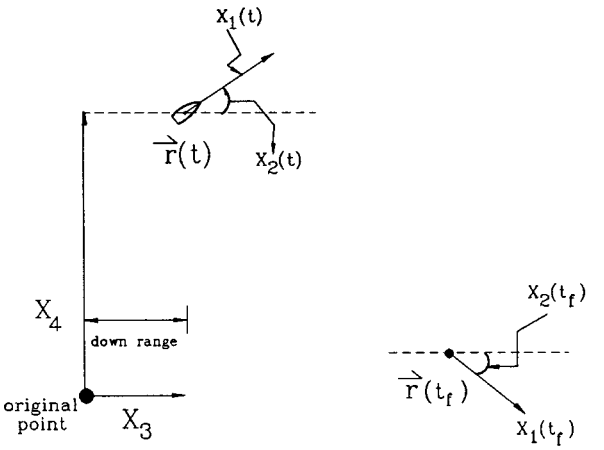

Fig. 2. Geometric relations and definitions of the states of the parabolic trajectory problem and the flying vehicle problem in Examples A and B of Section III.

III. Applications

In this section, we shall apply the recurrent learning algorithms developed in the last section to design optimal controllers for three application problems. They are 1-D maximum-rang parabolic trajectory problem, 2-D guidance problem, and the optimal PI control of a nonminimal phase linear dynamical system.

Example A (One-Dimensional Boundary Value Problem : Maximum–Rang Parabolic Trajectory Problem): Consider a particle with parabolic motion. Let x_1 be the velocity of the particle, x_2 the flight path angle (angle of velocity vector), x_3 the down range, and x_4 the altitude. Geometric relations of these states are shown in Fig. 2. By Newton's law, the parabolic motion equations of the particle are

$$\dot{x}_1 = -g\sin(x_2), \quad \dot{x}_2 = -g\cos(x_2)/x_1$$
$$\dot{x}_3 = x_1\cos(x_2), \quad \dot{x}_4 = x_1\sin(x_2) \tag{15}$$

where g is the gravitational force.

This example is to find the initial flight path angle of the particle such that the particle can reach the furthest place for a given particle initial velocity. More clearly, we want to find the optimal initial flight path angle $[x_2(0)]$ that results in the largest down range (x_3) for the given particle initial velocity, $x_1 = 100$ (ft/s). Although this example can be solved by the analytic approach to obtain the closed form solution; the optimal initial flight path angle is 45°, we shall use the proposed learning algorithms to solve this problem to demonstrate their learning ability.

Let the performance index of the above problem be

$$E \equiv (\boldsymbol{x}_3(T) - 3000)^2 \tag{16}$$

where T is a free time value indicating the time step at which the particle falls to the ground. In the above performance index, the constant 3000 (ft) is an arbitrarily chosen large number. Since the down range of the particle [described by (15)] cannot reach such a big value, minimizing the performance index E in (16) can make the particle reach the

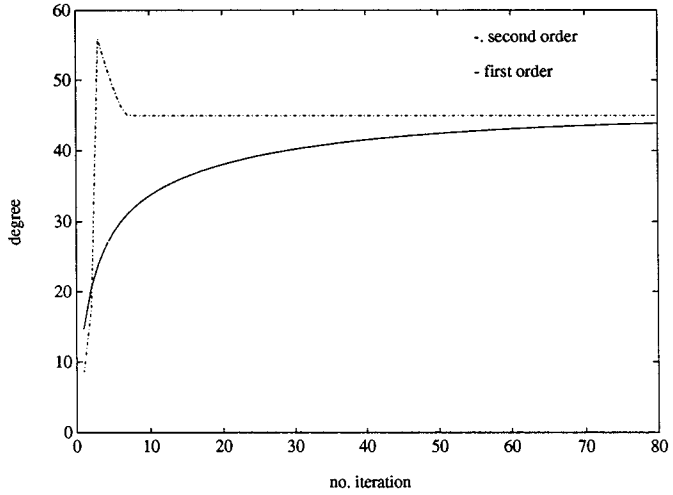

Fig. 3. Learning curves of the first-order (—) and second-order (.-) recurrent learning algorithms for finding the optimal initial flight path angle in Example A.

furthest place. When apply the proposed first- and second-order recurrent learning algorithms to find the optimal initial flight path angle, we obtain the two learning curves shown in Fig. 3, where the initial flight path angle before learning is randomly set as 5°. The real computation time per iteration of the first-order learning algorithm is about half of that of the second-order one. As comparing the real convergence time, the first-order learning algorithm took 55 s to get the initial flight path angle as 37° and 220 s to 43°, whereas the second-order learning algorithm took only 55 s to obtain the optimal initial flight path angle, 45°.

Example B (Two-Dimensional Boundary Value Problem: Guidance Problem): Consider a vehicle flying in the air. Assume that only the lateral direction of the flying vehicle is controllable by means of the air dynamic force caused by relative motion or by the dynamic force of the lateral engine in the vehicle. The axis direction of the flying vehicle is uncontrollable and has only inertial motion. We further assume that the motion drag of the flying vehicle is proportional to the square of velocity, and the other drag forces can be ignored. Since the gravity acceleration is assumed to be constant, its effect can be compensated directly by the controller according to the vehicle flight angle, and thus the gravity acceleration is not considered in the following analysis. The problem now is to design a controller that can let the vehicle move from point A in a 2-D space to the circumference of 500 feet of point B (i.e., a circle centered at point B with radius of 500 feet, called *boundary surface*). The additional requirements for the controller in achieving the above task are that it should make the energy of the total lateral forces smallest, and let the flying vehicle meet the desired boundary flight path angle and reach the maximum flight speed at the final point (i.e., at the boundary surface around point B). This problem belongs to the final state lying on surface problem and free final time problem in classical optimal control [1], [16].

By Newton's law and the above assumptions, the motion equations of the flying vehicle are simplified as

$$\begin{aligned} &\dot{x}_1 = c_1 \exp\{-c_2 x_4\} x_1^2, \quad x_1 \dot{x}_2 = -N_{VM\perp} \\ &\dot{x}_3 = x_1 \cos x_2, \quad \dot{x}_4 = x_1 \sin x_2 \end{aligned} \tag{17}$$

where x_1 is the velocity of the vehicle, x_2 is the flight path angle, x_3 represents the down range, x_4 is the height of the vehicle (altitude), $N_{VM\perp}$ is the lateral acceleration command from the controller, c_1 is the drag coefficient, and $\exp\{-c_2 x_4\}$ represents the air density. The geometric relations and definitions of these states are shown in Fig. 2.

The control objectives of this problem mentioned above can be described by the following performance index:

$$E = \int_{t_0}^{t_f} N_{VM\perp}(t)\, dt + g_1 (x_2 - r_f)^2 + g_2 (x_1 - v_f)^2 \tag{18}$$

where r_f is the desired final flight path angle, v_f is an arbitrary big velocity for achieving the maximum final velocity x_1, and g_1 and g_2 are weighting factors of E. In (18), t_0 is the initial time, and t_f is the time at which the vehicle arrives on the boundary surface. Hence the value t_f is unknown in designing the controller.

To design a *robust* optimal controller for this guidance problem, the structure of our controller is based on that designed by the classical analysis approach, the theory of calculus of variations [16], [17]. The classical analysis approach aims at letting the vehicle reach the final point B exactly (instead of reaching the boundary surface around point B) with an expected flight path angle. It thus minimizes the following performance index

$$E = \int_{t_0}^{t_f} |\vec{\boldsymbol{N}}(t)|^2\, dt, \tag{19}$$

where $\vec{\boldsymbol{N}}(t)$ represents the lateral acceleration command from the controller. The obtained optimal controller by the classical analysis approach (called "classical optimal controller") is [17]

$$\vec{\boldsymbol{N}}(t) = \frac{\boldsymbol{\omega}_1}{T_{go}} [\vec{\boldsymbol{V}}_1 - \vec{\boldsymbol{V}}(t_0)] + \frac{\boldsymbol{\omega}_2}{T_{go}^2} \{\vec{\boldsymbol{r}}_1 - \vec{\boldsymbol{r}}(t) - \vec{\boldsymbol{V}}(t_0) T_{go}\} \tag{20}$$

for each time interal, $t_0 \le t \le t_0 + ht$, where $\vec{\boldsymbol{N}}(t)$ is the acceleration command from the controller, $\vec{\boldsymbol{V}}(t)$ is the velocity vector of the vehicle, $\vec{\boldsymbol{r}}(t)$ is the position vector of the vehicle, $\vec{\boldsymbol{r}}_1$ is the final position of the vehicle, $\vec{\boldsymbol{V}}_1$ is the final velocity of the vehicle, and T_{go} is the flight time from position $\vec{\boldsymbol{r}}(t)$ to $\vec{\boldsymbol{r}}_1$ (time-to-go). There are some difficulties in using this classical optimal controller; it requires that the flying vehicle is controllable in all directions (instead of the lateral direction only), the air density must be constant (i.e., not function of x_4), and more importantly, we need to perform a precise estimation of the time-to-go value (T_{go}) at each time step [18], [19]. These requirements are relaxed in the optimal controller design in this paper. Also, (20) is singular when T_{go} approaches zero (i.e., when the vehicle is close to the final point B). Hence the classical optimal controller cannot really lead the vehicle to arrive in point B exactly.

Although the assumptions made in the classical analysis approach in solving the guidance problem are not all the same as those made in this paper, we shall properly adopt the form of the controller in (20) to build the structure of our controller. In our approach, we first formulate the variable, T_{go} (time-to-go), in (20) as a function of states (see the Appendix). This avoids the need to estimate T_{go} and thus simplifies the structure of our controller. We then formulate (17) and the controller as a neural network as that shown in Fig. 1(b), where the external command $I(t)$ is neglected. The parameters $\boldsymbol{\omega}_1$ and $\boldsymbol{\omega}_2$ in (20) are viewed as the weights of the network. Then the recurrent learning algorithms derived in Section II are used to find the optimal weights that minimizes (18).

More clearly, according to the derivations in the Appendix, the structure of our controller is

$$N_{VM\perp}(t) = \boldsymbol{\omega}_1 \frac{x_1^2 (x_2 - r_f)}{R} + \boldsymbol{\omega}_2 \frac{x_1^2 \sin \delta}{R} \tag{21}$$

where $\delta = x_2 + \tan^{-1}((x_4 - z_d)/(x_d - x_3))$, $R = \sqrt{(x_3 - x_d)^2 + (x_4 - z_d)^2}$, and $(x_d, z_d) = \vec{\boldsymbol{r}}_1$ is the desired position. The controller in (20) is then substituted by (21), and then the whole system is discretized by the four-order Runge–Kutta method to form a network as the one in Fig. 1(c). We then apply the proposed first- and second-order recurrent learning algorithms to find the

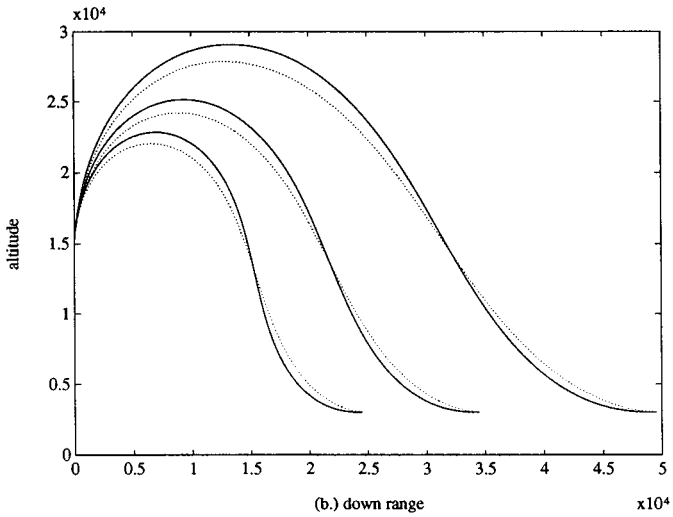

Fig. 4. Vehicle flying trajectories guided by the classical optimal controller $(\cdots)$ ($\omega_1 = -2$, $\omega_2 = 6$), and by the proposed recurrently learned optimal controller (—) ($\omega_1 = -2.247$, $\omega_2 = 5.919$) for three different testing points with the desired final flight path angle being $r_f = 0^\circ$ and the desired final flight height being $x_4 = 3000$ feet in Example B.

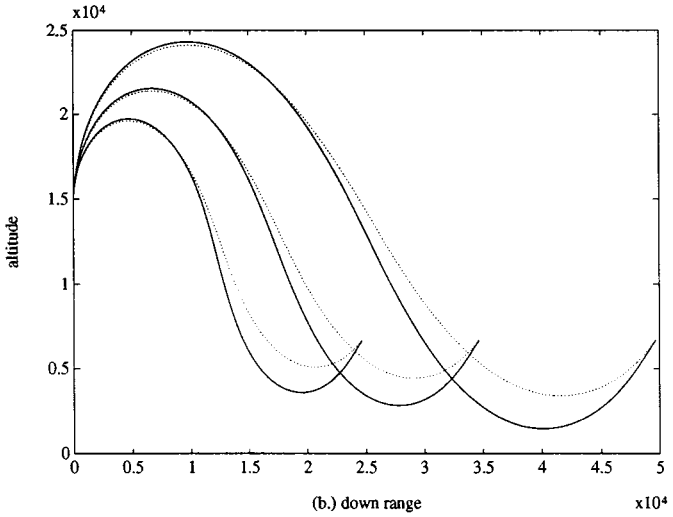

Fig. 5. Vehicle flying trajectories guided by the classical optimal controller $(\cdots)$ ($\omega_1 = -2$, $\omega_2 = 6$), and by the proposed recurrently learned optimal controller (—) ($\omega_1 = -2.247$, $\omega_2 = 5.919$) for three different testing points with the desired final flight path angle being $r_f = 46^\circ$ and the desired final flight height being $x_4 = 7000$ feet in Example B.

optimal controller gains, ω_1 and ω_2. To simplify the computations of the second-order learning algorithm, only the second-order terms, $(\partial^+\partial^+E/\partial\omega_1^2)$ and $(\partial^+\partial^+E/\partial\omega_2^2)$ (instead of the whole Hessian matrix of E), are calculated and used to change the learning constant adaptively.

In simulations, a total of ten training points are selected for the network learning; $(x_d;\ z_d;\ r_f)$ = (20 000; 5000 or 0; 0.2 or -0.4), (35 000; 5000 or 0; 0.3 or -0.5), (50 000; 5000; 0.4 or -0.6). The training data are chosen from the boundaries of a vehicle's flight envelop, which is decided by the average speed of the vehicle. For testing the generalization ability after network learning, we choose six testing points; $x_d = 25\,000$ or $35\,000$ or $50\,000$, $z_d = 3000$ or 7000, $r_f = 0$ or 0.8. The final learned optimal controller gains by the second-order learning algorithm are $\omega_1 = -2.247$ and $\omega_2 = 5.919$. For comparison, we also use the classical analysis method (the theory of calculus of variations) to minimize (19) to find the optimal controller gains as $\omega_1 = -2$ and $\omega_2 = 6$. We then compare the guidance ability of the learned controller [i.e., the controller in (21) with $\omega_1 = -2.247$ and $\omega_2 = 5.919$] to that of the classical optimal controller [i.e., the controller in (21) with $\omega_1 = -2$ and $\omega_2 = 6$]. The vehicle flight trajectories under the control of these two controllers for the six testing points are plotted in Figs. 4 and 5. These two figures show that the learned optimal controller produces better final flight path angle (see the solid curves) than the classical optimal controller (see the dotted curves), when the vehicle reaches the boundary surface around point B. Notice that the desired final flight path angle in Fig. 4 is $r_f = 0^\circ$, and that in Fig. 5 is $r_f = 46^\circ$; the desired final flight height in Fig. 4 is $x_4 = 3000$ feet, and that in Fig. 5 is $x_4 = 7000$ feet. The performance index (error) values are listed in Table I. From Table I, it is clear that the learned optimal controller is much better than the classical optimal controller in guidancing the vehicle. Since some testing points are exterpolation of the training points, the learned optimal controller shows superior generalization ability. Fig. 6 shows the convergence curves on the weight space for the first- and second-order learning algorithms with different initial weights (controller gains), where the first-order learning algorithm is run for 250 iterations, and the second-order learning algorithm is run for 150 iterations. We find the second-order learning algorithm has much higher learning speed than the first-order one. In Fig. 6, there appears

TABLE I
PERFORMANCE INDEX (ERROR) VALUES FOR THREE DIFFERENT CONTROLLERS UNDER SIX TESTING CASES IN THE GUIDANCE PROBLEM OF FLYING VEHICLE IN EXAMPLE B OF SECTION III, WHERE CASE-1: ($x_d = 25\,000$, $z_d = 3000$, $r_f = 0$), CASE-2: ($x_d = 35\,000$, $z_d = 3000$, $r_f = 0$), CASE-3: ($x_d = 50\,000$, $z_d = 3000$, $r_f = 0$), CASE-4: ($x_d = 25\,000$, $z_d = 7000$, $r_f = 0.8$), CASE-5: ($x_d = 35\,000$, $z_d = 7000$, $r_f = 0.8$), CASE-6: ($x_d = 50\,000$, $z_d = 7000$, $r_f = 0.8$)

Controller Types	case-1	case-2	case-3	case-4	case-5	case-6
$\omega_1 = -2, \omega_2 = 6$	1802	1350	891.9	1995	1482	1098
$\omega_1 = -2.247, \omega_2 = 5.919$	192.6	219.6	265.9	642.9	443.9	396.6
optimal d_1, d_2 of Eq. (22)	192.2	220.0	264.1	654.9	425.12	375.9

a long flat valley in the error surface (E) on the weight (ω_1 – ω_2) space (the long straight line in the figure). This greatly limits the convergence speed of the first-order recurrent learning algorithm.In another simulation, we try to add more adjustable parameters (gains) to the controller described by (21) to see if the controller can be further improved. The new controller is designed as

$$N_{VM\perp}(t) = d_1\frac{x_1^2(x_2 - r_f)}{R} + d_2\frac{x_1^2\sin\,\delta}{R} \tag{22}$$

where

$$d_1 = \omega_1\left(1 + \frac{\omega_2}{R}\right), \quad d_2 = \omega_3\left(1 + \frac{\omega_4}{R}\right) \tag{23}$$

where the gains d_1 and d_2 are functions of range R in (21). By using the first-order recurrent learning algorithm, we perform the same learning process as that mentioned in the above to minimize (18) to find the optimal controller gains as $\omega_1 = -2.243$, $\omega_2 = 8.213$, $\omega_3 = 5.988$, $\omega_4 = -42.10$. The guidance results of this newly learned controller are also shown in Table I. The results indicate that the generalization ability of the controller in (22) is not better than that of the controller in (21). Once again, this shows the good robustness of the controller in (21), since the controller gains need not be tuned for different R values.

Example C (Optimal PI Control Problem: Generalized PI Controller): Consider a nonminimal phase linear dynamical system, $P(s) = ((s-3))/((s-2)(s-1))$, controlled by a PI controller. The states feedback from the linear system to the controller are the error

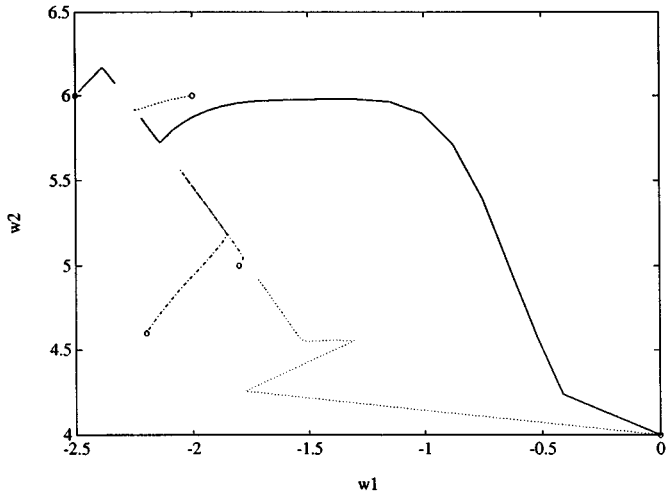

Fig. 6. Convergence curves on the weight space for the first-order ($\cdots$) and second-order (the curves other than the dotted curves) recurrent learning algorithms with different initial weights (controller gains) in Example B, where the first-order learning algorithm is run for 250 iterations, and the second-order learning algorithm is run for 150 iterations. (The circle indicates the starting point of a curve.)

signal $e(t)$ and the integral error signal $\int_0^t e(t)$. Assume the system external input command is in the form of pulses, whose magnitudes and pulse widths are adaptable. The problem now is to design an optimal PI controller to minimize the overshoot of the system step response. Our design procedure is first using the PI control law as the basic structure of the optimal controller. The gains of the controller are then tuned by the proposed recurrent learning algorithms. We use the unit step commands to the system as training data, and set the performance index as

$$E = \int_{t_0}^{t_f} (1 - y(t))^2 \, dt, \tag{24}$$

where $y(t)$ is the output of $P(s)$. This performance index takes care of both the overshoot and tracking errors, where t_0 is set as $15h$ ($h = 0.1$ s) and t_f as 15 s. As did in the previous examples, we use the four-order Runge–Kutta method to model the overall system, and then apply the recurrent learning algorithms derived in Section II to search the optimal PI gains. The performance of the learned classical PI controller in the generalization tests is shown in Fig. 7(b), which appears about 8.2% overshoot on the system step response.

We now propose a new PI controller, called generalized PI controller, which is a classical PI controller with adaptive PI gains. This controller allows its PI gains to be changed adaptively according to the system states, e and $\int_0^t e \, dt$, in the transient period. The adaptive PI gains in the generalized PI controller are given by

$$\boldsymbol{k}_p = \boldsymbol{k}_{p0} \exp\left\{\boldsymbol{\omega}_1 \left[\int_0^t e(t)\, dt\right]^2\right\}.$$

$$\boldsymbol{k}_I = \boldsymbol{k}_{I0} \exp\{\boldsymbol{\omega}_2 e(t)^2\}.$$

Hence, the P gain, $\boldsymbol{k}_p$, is a function of $\int_0^t e(t)\, dt$, and the I gain, $\boldsymbol{k}_I$, is a function of $e(t)$. Notice that when the system arrives in steady states (i.e., $e \to 0$, $\int_0^t e \, dt \to constant$) or when $\boldsymbol{\omega}_1$ and $\boldsymbol{\omega}_2$ are set as zero, the PI gains of the controller are fixed and the generalized PI controller is reduced to a classical one.

To train the generalized PI controller, we again use the four-order Runge–Kutta method to discretize the whole system [including the plant $P(s)$ and the generalized PI controller], and then apply the proposed recurrent learning algorithms to minimize the performance index E in (24). The learning speed of the first order and second order learning algorithms are illustrated in Fig. 7(a), showing again the higher convergence speed of the second-order learning algorithm. The obtained optimal gains are $\boldsymbol{k}_{p_0} = -0.9265$, $\omega_1 = -2.366$, $\boldsymbol{k}_{i_0} = -0.5859$, and $\omega_2 = 0.9158$ by the second-order learning algorithm. Fig. 7(b) shows the step responses of the system under the control of the learned classical PI controller and the learned generalized PI controller for comparison. The figure shows that the generalized PI controller causes the system to have only 1.5% overshoot, and have a faster response than the classical PI controller. It is noted that since the generalized PI controller is a nonlinear controller, state transformation should be performed properly by viewing the current state as the equilibrium state for each step command. Although the generalized PI controller is a nonlinear controller, its optimal controller gains can be easily obtained by the recurrent learning algorithms proposed in this paper. The learned generalized PI controller dose reduce the system overshoot efficiently as compared to the learned classical PI controller in this example.

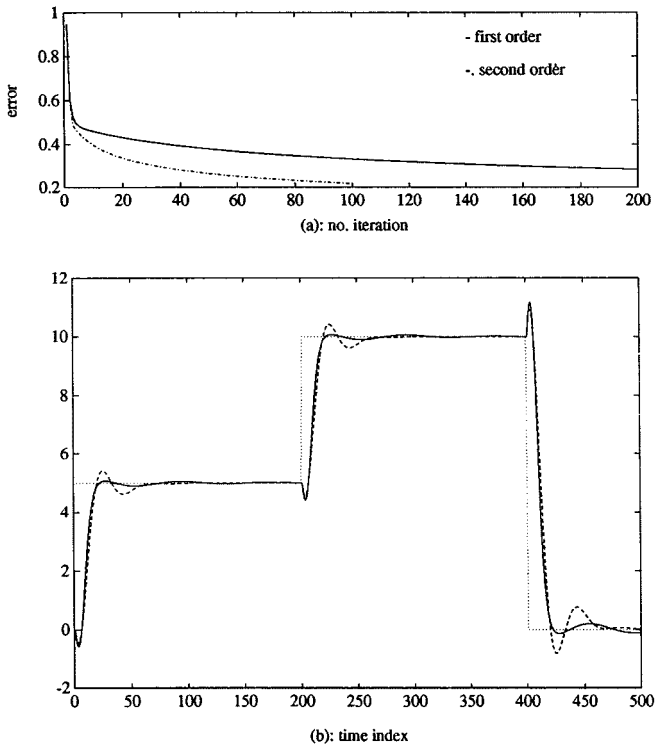

Fig. 7. Performance of the PI controllers in Example C. (a) Convergence curves of the first-order (—) and second-order ($\cdots$) recurrent learning algorithms in tuning the classical PI controller. (b) Step responses of the system controlled by the learned classical PI controller (- - -) and the learned generalized PI controller (—), where the desired step response is indicated by the dotted ($\cdots$) line.

IV. Conclusion

This paper proposes neural-network-type recurrent learning algorithms for designing optimal controllers of continuous systems, which are modeled (discretized) by the high-order Runge–Kutta method. As shown in the paper, the derived learning algorithms are in forward recurrent form, and are easy to implement. Especially, the proposed second-order recurrent learning algorithm shows superior convergence speed and accuracy. To achieve good generalization capability, the structures of the controllers designed in this paper are all based on the form of classical controllers. It is straightforward to extend the proposed recurrent learning algorithms to train the neuro-controllers that are general neural networks (such as multilayer perceptrons). We

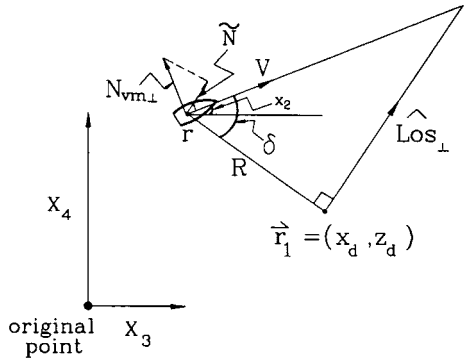

Fig. 8. Geometric relations of parameters δ, R, and $\hat{Los}_\perp$ in the Appendix for the guidance problem of Example B.

also propose two new controllers in this paper, the guidance controller of the flying vehicle without the need of estimating the time-to-go parameter, and the generalized PI controller with adaptive gains. These two controllers show better performance than their traditional counterparts. However, their properties such as stability and robustness need further studies.

APPENDIX

This Appendix is to formulate the variable, T_{go} (time-to-go), in (20) as a function of system states to avoid the estimation of T_{go} and thus simplifies the structure of our controller. Consider a vehicle at position $\boldsymbol{r}$ with velocity $\boldsymbol{V}$ in Fig. 8. Let

$$\delta = \boldsymbol{x}_2 + \theta, \quad \text{where } \theta = \tan^{-1}\left(\frac{\boldsymbol{x}_4 - \boldsymbol{z}_d}{\boldsymbol{x}_d - \boldsymbol{x}_3}\right) \tag{25}$$

and

$$R = \sqrt{(\boldsymbol{x}_3 - \boldsymbol{x}_d)^2 - (\boldsymbol{x}_4 - \boldsymbol{z}_d)^2}. \tag{26}$$

From the figure, the time-to-go value, T_{go}, can be estimated by

$$T_{go} \approx \frac{R}{|\boldsymbol{V}| \cos \delta}. \tag{27}$$

Then the second term of (20) can be approximated by

$$\frac{\boldsymbol{r}_1 - \boldsymbol{r}(t) - \boldsymbol{V}(t)T_{go}}{T_{go}^2} \approx \frac{(|\boldsymbol{V}(t)| \sin \delta)T_{go}}{T_{go}^2} \cdot \hat{Los}_\perp = \frac{|\boldsymbol{V}(t)|^2 \cdot \sin \delta(t)}{R} \cos \delta \cdot \hat{Los}_\perp \tag{28}$$

where $\hat{Los}_\perp$ is the unit vector perpendicular to $\boldsymbol{r}_1 - \boldsymbol{r}$. Also, the first term of (20) can be approximated by

$$\frac{\boldsymbol{V}_1(t) - \boldsymbol{V}(t)}{T_{go}} \approx \frac{|\boldsymbol{V}(t)|^2(x_2 - r_f)}{R} \cos \delta \cdot \hat{Los}_\perp. \tag{29}$$

From (28) and (29), we can obtain a guidance controller which need not explicitly compute T_{go} as

$$\tilde{N}(t) = \omega_1 \frac{|\boldsymbol{V}(t)|^2(x_2 - r_f)}{R} \cos \delta + \omega_2 \frac{|\boldsymbol{V}(t)|^2 \cdot \sin \delta(t)}{R} \cos \delta. \tag{30}$$

The direction of the control output, $\tilde{N}(t)$, is parallel to $\hat{Los}_\perp$. Since we assume that only the lateral direction of the flying vehicle is controllable, the control command to the vehicle at its lateral direction should be

$$N_{VM\perp}(t) = \omega_1 \frac{|\boldsymbol{V}(t)|^2(x_2 - r_f)}{R} + \omega_2 \frac{|\boldsymbol{V}(t)|^2 \sin \delta}{R}$$

which is (21). It is noted that the projection of the above control command in the direction of $\hat{Los}_\perp$ is equal to (30). This is exactly what we want.

REFERENCES

[1] D. E. Kirk, *Optimal Control Theory: An Introduction*. Englewood Cliffs, NJ: Prentice-Hall, 1970.
[2] W. T. Miller, R. S. Sutton, and P. J. Werbos, *Neural Networks for Control*. Cambridge, MA: MIT Press, 1990.
[3] K. S. Narendra and K. Parthasarathy, "Identification and control of dynamical system using neural networks," *IEEE Trans. Neural Networks*, vol. 1, no. 1, pp. 4–27, 1990.
[4] ——, "Gradient methods for the optimization of dynamical systems containing neural network," *IEEE Trans. Neural Networks*, vol. 2, no. 2, pp. 252–262, 1991.
[5] C. T. Lin and C. S. G. Lee, *Neural Fuzzy Systems*, ser. NJ. Englewood Cliffs: Prentice-Hall, 1996.
[6] S. W. Piche, "Steepest descent algorithms for neural network controllers and filters," *IEEE Trans. Neural Networks*, vol. 5, no. 2, pp. 198–212, 1994.
[7] G. V. Puskorius and L. A. Feldkamp, "Neurocontrol of nonlinear dynamical system with Kalman filter trained recurrent networks," *IEEE Trans. Neural Networks*, vol. 5, no. 2, pp. 279–297, 1994.
[8] R. J. Williams and D. Zipser, "A learning algorithm for continually running fully recurrent neural networks," *Neural Comput.*, vol. 1, no. 2, pp. 270–280, 1989.
[9] ——, "Experimental analysis of the real-time recurrent learning algorithm," *Connect. Scitation*, vol. 1, no. 1, pp. 87–111, 1989.
[10] H. Drucker and Y. LeCun, "Improving generalization performance using double backpropagation," *IEEE Trans. Neural Networks*, vol. 3, no. 6, pp. 991–997, 1994.
[11] A. Weinmann, *Uncertain Models and Robust Control*. New York: Spring-Verlag, 1992.
[12] E. D. Sontag, *Mathematical Control Theory: Deterministic Finite-Dimensional Systems*. New York: Spring-Verlag, 1990.
[13] P. J. Werbos, "Backpropagation through time: What it does and how to do it," *Proc. IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.
[14] R. J. Williams and J. Pieng, "An efficient gradient-based algorithm for on-line training of recurrent network trajectories," *Neural Comput.*, vol. 2, pp. 490–501, 1990.
[15] Y. J. Wang and C. T. Lin, "A second-order learning algorithm for multilayer networks based on block Hessian matrix," *Neural Networks*, vol. 11, no. 9, pp. 1607–1622, 1998.
[16] A. E. Bryson and Y. C. Ho, *Applied Optimal Control*. Bristol, PA: Hemisphere, 1975.
[17] R. H. Battin, *An Introduction to the Mathematics and Methods of Astrodynamics*, ser. AIAA Education Series, 1987, ch. 11.
[18] T. Riggs, "Linear optimal guidance for short range air-to-air missiles," in *AIAA Nat. Meet.*, August 1978, (also available as 1979 NAECON, May 1979).
[19] R. J. York, "Improved time-to-go estimation for an optimal control law," *Adv. Anal. Future Missiles*, pp. 323–384, November 21, 1979.