

Received December 24, 2020, accepted January 12, 2021, date of publication January 18, 2021, date of current version January 27, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3051984

A Nearer Optimal and Faster Trained Value Iteration ADP for Discrete-Time Nonlinear Systems

JUNPING HU¹, GEN YANG^{1,2}, ZHICHENG HOU², (Member, IEEE),
GONG ZHANG², (Member, IEEE), WENLIN YANG², (Member, IEEE),
AND WEIJUN WANG^{2,3}

¹College of Mechanical and Electrical Engineering, Central South University, Changsha 410083, China

²Guangzhou Institute of Advanced Technology, Chinese Academy of Sciences, Guangzhou 511458, China

³Shenzhen CAS Derui Intelligent Technology Company Ltd., Shenzhen 518000, China

Corresponding author: Zhicheng Hou (zc.hou@giat.ac.cn)


This work was supported in part by the China Postdoctoral Science Foundation under Grant 2019M662848, and in part by the Natural Science Foundation of Guangdong Province under Grant 2018A030310046.

ABSTRACT Adaptive dynamic programming (ADP) is generally implemented using three neural networks: model network, action network, and critic network. In the conventional works of the value iteration ADP, the model network is initialized randomly and trained by the backpropagation algorithm, whose results are easy to get trapped in a local minimum; both the critic network and action network are trained in each outer-loop, which is time-consuming. To approximate the optimal control policy more accurately and decrease the value iteration ADP training time, we propose a nearer optimal and faster trained value iteration ADP for discrete-time nonlinear systems in this study. First, before training the model network with a backpropagation algorithm, we use a global searching method, i.e., genetic algorithm, to evolve the weights and biases of the neural network for a few generations. Second, in the outer-loop training process, we propose a trigger mechanism to decide whether to train the action network or not, which can save much training time. Examples of both linear and nonlinear systems are induced to verify the superiority of the proposed method compared with the conventional value iteration ADP. The simulation results show that the proposed algorithm can provide a nearer optimal control policy and save more training time than the conventional value iteration ADP.

INDEX TERMS ADP, value iteration, genetic algorithm, trigger mechanism.

I. INTRODUCTION

Discrete-time systems are more suitable for computer processing than the continuous-time systems [1]. In practice, the continuous-time system control problems are mostly discretized into discrete-time system control [2]–[5]. Adaptive dynamic programming (ADP) proposed by Werbos [6] is an effective method for solving the optimal control problems for discrete-time systems [5], [18]–[24]. ADP can also be described as adaptive critic designs [7]–[9], neural dynamic programming [10], or reinforcement learning [11]–[14]. There are two principal schemes of ADP algorithms, which are the policy iteration algorithm [15]–[18]

The associate editor coordinating the review of this manuscript and approving it for publication was Hao Luo .

and the value iteration algorithm [19]–[22]. A policy iteration algorithm requires an initial stabilizing control action [18], whereas a value iteration algorithm does not [19]. In this research, we utilize the promising value iteration algorithm.

In the research of value iteration ADP (VI-ADP), Al-Tamimi *et al.* [19] prove the convergence of VI-ADP for discrete-time nonlinear systems. This work provides a theoretical foundation for VI-ADP research. Different from [19], where the initial cost function (also called value function [24] or performance index function [18]) is set as 0, Li *et al.* [22] propose a general VI-ADP that initializes the cost function with a Lyapunov function. Compared with VI-ADP, the general VI-ADP is able to guarantee the stability of the system with a finite iterative control policy. Still, it is required that $V_0(x_k) \geq V_1(x_k)$ holds for any

controllable x_k , which is hard to satisfy. Considering the existence of approximation errors of the neural networks, Heydari [4] analyzes the stability of VI-ADP. It is proved that if the initial iterative cost function $V_0(\cdot)$ belongs to an admissible control policy, the iterative control policies remain stable. As most previous works are operated on the whole state space in each iteration, Wei *et al.* [5], [23] develop a local VI-ADP that the iterative cost functions and control policy are both updated in a given subset of the state space in each iteration. Since the real-world systems generally operate in a subset of the state space in each iteration period, the local VI-ADP is more suitable for real-world applications. After that, Li *et al.* [24] propose an online VI-ADP algorithm that combines the local VI-ADP with a gaussian distribution that treats the current states as the expected value. This algorithm can update the cost function and control policy at each running step of the system.

However, most of the previous studies on VI-ADP algorithms update both the iterative cost function (critic network) and the iterative control policy (action network) in each outer-loop, i.e., train two neural networks in each outer-loop. High order control systems require a large amount of state data to build the approximation structures to approximate the iterative cost function and iterative control policy [5]. Even though VI-ADP is often used as an offline ADP scheme, it costs too much time when implementing VI-ADP in the high order systems. Furthermore, in the training process of the model network, the weights and biases are always initialized randomly. A genetic algorithm (GA) is a kind of evolutionary algorithm that works well in global optimization [28]. Compared with the BP neural network whose weights and biases are initialized randomly, the one whose weights and biases are initialized by GA (GA-BP) has a better approximation accuracy and fault tolerance ability [26]–[28]. However, it is uncommon in the literature to utilize GA-BP to train the model network. Moreover, in the conventional GA-BP algorithm, GA often evolves BP neural network parameters for large numbers of generations, i.e., 500 generations in [26] and 1000 generations in [28], which is also time-consuming.

In this study, to approximate the optimal control policy more accurately and decrease the training time of VI-ADP, we propose a nearer optimal and faster trained VI-ADP (NoFt-VI-ADP) algorithm for discrete-time nonlinear systems. The main contributions of this research are summarized as follows:

- 1) Different from previous works that initialize the weights and biases of the model network randomly, we implement GA to evolve these parameters before being tuned by the BP algorithm.
- 2) Unlike previous GA-BP works that evolve the parameters of BP neural network for large numbers of generations, we only evolve these parameters for a few generations.
- 3) Different from the conventional works that train the critic network and action network in each outer-loop,

we propose a novel trigger mechanism that only trains the action network when there is a trigger signal.

- 4) The proposed NoFt-VI-ADP performs in a data-driven way that does not require the knowledge of system dynamics.

The organization of this article is as follows: In sect. 2, the less evolved GA-BP algorithm is implemented to train the model network. In sect. 3, the trigger mechanism is used to train the outer-loop. In sect. 4, the overall NoFt-VI-ADP algorithm is given. In sect. 5, linear and nonlinear examples are illustrated to show the superiority of the proposed NoFt-VI-ADP compared with VI-ADP. The conclusions are given in sect.6.

II. TRAIN THE MODEL NETWORK WITH A LESS EVOLVED GA-BP ALGORITHM

In most cases, since the real dynamics of a system are difficult to obtain, the neural network is used to approximate the dynamics of the system [19]–[22]. As shown in Fig. 1, a large number of (x_s, u_s, x'_s) are sampled from the system, where x_s represents the current state vector, u_s is the control vector, and x'_s is the corresponding successor state vector. The notation s stands for the sampled data. With these data, we train a three layers model network (as shown in Fig. 2) with the input $[x_s^T, u_s^T]^T$ and the target x'_s to approximate the dynamics of the system. The output of the model network is expressed as follows:

$$\hat{x}_s^{ij} = W^{jT} \sigma(Z^j), \tag{1}$$

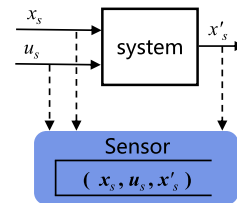


FIGURE 1. Flow chart of sampling offline data.

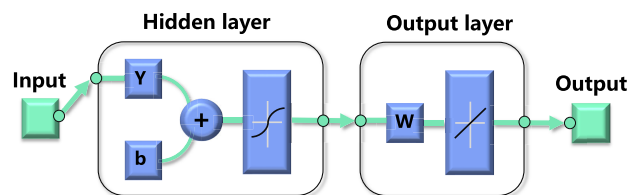


FIGURE 2. Neural network diagram.

where j is the training iteration index, \hat{x}_s^{ij} is the estimation of x'_s , W^j is the hidden-to-output-layer weights matrix, σ is a $\tanh(\cdot)$ function, i.e., $\sigma(x) = [(e^x - e^{-x}) / (e^x + e^{-x})]$, $Z^j = Y^jT [x_s^T, u_s^T]^T + b^j$, Y^j is the input-to-hidden-layer weights matrix, b^j is the input-to-hidden-layer biases matrix. Then we define the error function of the neural network as:

$$e^j = \hat{x}_s^{ij} - x'_s. \tag{2}$$

The objective function to be minimized in the neural network training process is expressed as:

$$E^j = \frac{1}{2}(e^j)^T(e^j). \quad (3)$$

Remark 1: In this article, all the matrix differentiation is in the denominator layout form.

Therefore, W^j is updated as follows:

$$\begin{aligned} W^{j+1} &= W^j + \Delta W^j \\ &= W^j - \alpha \left[\frac{\partial E^j}{\partial W^j} \right] \\ &= W^j - \alpha \left[\frac{\partial \hat{x}_s^{j'}}{\partial W^j} \frac{\partial e^j}{\partial \hat{x}_s^{j'}} \frac{\partial E^j}{\partial e^j} \right] \\ &= W^j - \alpha \sigma(Z^j) e^{jT}, \end{aligned} \quad (4)$$

where $\alpha > 0$ is the learning rate. Herein, we prove the convergence of the model network as follows:

A. CONVERGENCE ANALYSIS OF THE MODEL NETWORK

Theorem 1: Let the neural network target be expressed by

$$x'_s = W^{*T} \sigma(Z). \quad (5)$$

Let the model network be trained by (4). If the learning rate α is small enough, then the neural network weights W is asymptotically convergent to the optimal weights W^* . [18]

Proof: Let $\bar{W}^j = W^j - W^*$. From (4) we have

$$\bar{W}^{j+1} = \bar{W}^j - \alpha \sigma(Z^j) e^{jT}. \quad (6)$$

Consider the following Lyapunov function candidate:

$$L(\bar{W}^j) = \text{tr} \left\{ \bar{W}^{jT} \bar{W}^j \right\}. \quad (7)$$

where tr means the trajectory of the matrix. When $W^j = W^*$, the term $L(\bar{W}^j)$ is 0. Then, the difference of the Lyapunov function candidate is given by:

$$\begin{aligned} \Delta L(\bar{W}^j) &= \text{tr} \left\{ \bar{W}^{(j+1)T} \bar{W}^{(j+1)} - \bar{W}^{(j)T} \bar{W}^{(j)} \right\} \\ &= \text{tr} \left\{ \bar{W}^{(j+1)T} \bar{W}^{(j+1)} \right\} - \text{tr} \left\{ \bar{W}^{(j)T} \bar{W}^{(j)} \right\} \\ &= \text{tr} \left\{ -2\bar{W}^{(j)T} \left[\alpha \sigma(Z^j) e^{jT} \right] \right. \\ &\quad \left. + \alpha^2 \left[\sigma(Z^j) e^{jT} \right]^T \sigma(Z^j) e^{jT} \right\} \\ &= \text{tr} \left\{ -2\alpha \left[e^j e^{jT} \right] + \alpha^2 \left[\sigma(Z^j) e^{jT} \right]^T \sigma(Z^j) e^{jT} \right\} \\ &= \alpha \left\| e^j \right\|^2 (-2) + \alpha^2 \sum_{t_1=1, t_2=1}^{\epsilon, l} \left[\sigma(Z^j)(t_1) \cdot e^j(t_2) \right]^2 \\ &= \alpha \left\| e^j \right\|^2 (-2) + \alpha^2 \left\| e^j \right\|^2 \left\| \sigma(Z^j) \right\|^2 \\ &= \alpha \left\| e^j \right\|^2 \left(-2 + \alpha \left\| \sigma(Z^j) \right\|^2 \right), \end{aligned} \quad (8)$$

where ϵ is the number of hidden layer neurons, l is the number of output layer neurons. According to the definition

of $\sigma(\cdot)$, we know that $\sigma(Z^j)$ is finite for $\forall Z^j$. Thus, if α is small enough that satisfy $\alpha \leq 2 / \left\| \sigma(Z^j) \right\|^2$, then we have $\Delta L(\bar{W}^j) \leq 0$, and \bar{W}^j will converge to 0, which means W is asymptotically convergent to the optimal weights W^* . The proof is completed.

So, the model network asymptotically converges to the dynamics of the system.

B. A LESS EVOLVED GA-BP ALGORITHM

GA can search points over the entire domain, whereas the BP algorithm can locally optimize the points. The procedure of GA-BP is mainly in two steps. Firstly, we use GA to generate the fittest chromosome to initialize the weights and biases of the neural network. Secondly, we train the neural network with gradient-based BP algorithms.

In the conventional GA-BP algorithms, GA often evolves for hundreds or thousands of generations to improve the performance of the neural network, whereas it is still time-consuming. Herein, we utilize a less evolved GA that only evolves the weights and biases of the model network for a few times, i.e., 20 to 50 generations. Although the less evolved GA will not obtain the globally optimal values, it can generate a group of parameters that performs much better than chosen randomly, which is illustrated in Fig. 3. The main elements of GA are explained as follows:

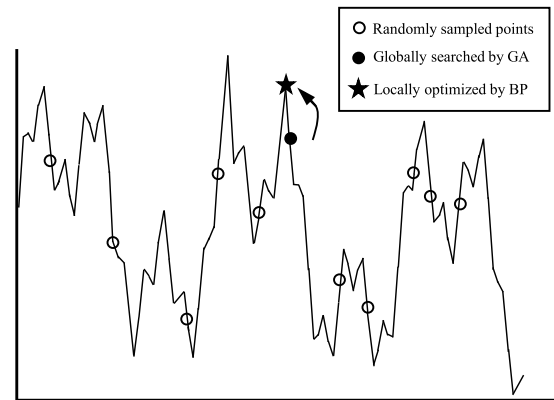


FIGURE 3. Global searching by GA and local optimization by BP.

1) ENCODING

The most crucial part of GA is to translate the parameters of a real problem into a chromosome. In GA-BP, a chromosome is a binary string composed of all the weights and biases of the BP neural network (illustrated in Fig. 4). Each chromosome represents a neural network, and all the chromosomes in the same generation constitute a population.

2) FITNESS FUNCTION

In GA, chromosomes in a population compete and exchange information with each other. The fitness function determines the survival probability of each chromosome. A chromosome with a higher fitness has a bigger probability to survive in the

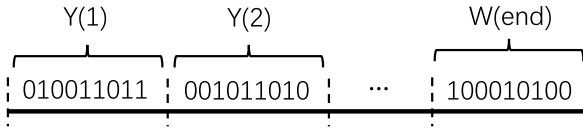


FIGURE 4. Illustration of a GA-BP chromosome.

next generation. In this study, we define the fitness function as follows:

$$f_h = \frac{1}{E_h^{MSE}}, \quad (9)$$

where f_h represents the fitness of the h -th chromosome, E_h^{MSE} represents the mean square error (MSE) of the neural network corresponding to the h -th chromosome expressed as:

$$E_h^{MSE} = \frac{1}{2N} \sum_{z=1}^N (\hat{x}_{sh}^{(z)} - x_s^{(z)})^T (\hat{x}_{sh}^{(z)} - x_s^{(z)}), \quad (10)$$

where N is the total groups of the sampled state vectors, $\hat{x}_{sh}^{(z)}$ is the output of the model network corresponding to the h -th chromosome.

3) SELECTION

Select K chromosomes from the previous generation population. During the selection, the chromosome with the largest fitness is retained, and the others are obtained by roulette wheel selection.

4) CROSSOVER

Randomly choose a pair of chromosomes from the population with the probability of P_c for K times. Each group of chromosome constitutes a couple of parents. Through single-point crossover, these parents generate offspring. After that, all the offspring are added to the population.

5) MUTATION

Mutate genes of every chromosome with the probability of P_m . After that, add the mutation results to the population.

At last, a summary of the less evolved GA-BP is given in algorithm 1. In this process, the model network is trained by a less evolved GA-BP algorithm.

III. TRAIN THE OUTER-LOOP WITH TRIGGER MECHANISM

In this section, firstly, we review the value iteration scheme for solving the Bellman equation of discrete-time systems. Then, we introduce the trigger mechanism in the outer-loop training process.

A. THE VALUE ITERATION SCHEME

Consider the following discrete-time system

$$x(k+1) = f(x(k)) + g(x(k))u(k), \quad k = 0, 1, 2, \dots, \quad (11)$$

Algorithm 1 Train the Model Network With the Less Evolved GA-BP

Step 1. Determine the topology of the model network. Provide the maximum generation G (20 to 50), the crossover probability P_c , the mutation probability P_m and the population size Q . Set $g = 0$.

Step 2. Randomly generate K groups of real number weights and biases as individuals of the population.

Step 3. Calculate the fitness of all the individuals, select K individuals from the population. During the selection, the individual with the largest fitness is retained, and the others are obtained by roulette wheel selection.

Step 4. Code the selected individuals into binary string chromosomes, crossover with a probability of P_c , mutate with a probability of P_m , add the offspring to the population. Then decode all the chromosomes into real number individuals.

Step 5. If $g = G$, go to step 6; else $g = g + 1$, and go back to Step 3.

Step 6. Calculate all the individuals' fitness, initialize the weights and biases of the model network with the fittest individual.

Step 8. Train the model network with gradient-based BP algorithms until meeting the termination conditions.

Step 7. Finish the algorithm.

where $x(k) \in \mathbb{R}^l$ is the state vector at time step k , $u(k) \in \mathbb{R}^r$ is the control vector, $f(x(k)) \in \mathbb{R}^l$ represents the drift dynamics, and $g(x(k)) \in \mathbb{R}^{l \times r}$ represents the input coupling function. Both $f(\cdot)$ and $g(\cdot)$ are unknown. Assume that the system is Lipschitz continuous on a compact set $\Omega \in \mathbb{R}^l$ containing the origin.

The goal is to find a state feedback control policy $\pi^*(x(k))$ that to any $x(k) \in \Omega$, it can generate a control sequence $\{u^*(k), u^*(k+1), u^*(k+2), \dots\}$ to stabilize the system (11) and simultaneously minimize the infinite-horizon cost function given by:

$$V(x(k), u(k)) = \sum_{j=0}^{\infty} U(x(k+j), u(k+j)), \quad (12)$$

where U is the utility function expressed as follows:

$$U(x(k), u(k)) = x^T(k)Qx(k) + u^T(k)Ru(k), \quad (13)$$

$Q \in \mathbb{R}^{l \times l}$ and $R \in \mathbb{R}^{r \times r}$ are both positive-definite matrices.

Definition 1: A control policy $\pi(\cdot)$ is admissible [19], [21] with respect to system (11) on a compact set Ω if $\pi(\cdot)$ is continuous on Ω , $\pi(0) = 0$, $\pi(x)$ stabilizes (11) on Ω , and $V(x, \pi(x))$ is finite, $\forall x \in \Omega$.

Let Ω_π be the set of admissible control policies associated with the state set Ω . We define the optimal cost function as follows:

$$V^*(x(k)) = \inf_{\pi} \{V(x(k), \pi(x(k))) : \pi \in \Omega_\pi\}. \quad (14)$$

According to [31], the optimal cost function $V^*(x(k))$ satisfies the Bellman equation:

$$V^*(x(k)) = \min_{u(k)} \{U(x(k), u(k)) + V^*(x(k+1))\}. \quad (15)$$

The optimal control vector $u^*(k)$ should satisfy

$$u^*(k) = \arg \min_{u(k)} \{U(x(k), u(k)) + V^*(x(k+1))\}. \quad (16)$$

Note that $u^*(k)$ satisfies the first-order necessary condition, which is given by the gradient of the right-hand side of (16) to $u(k)$ as follows:

$$\frac{\partial(x^T(k)Qx(k) + u^T(k)Ru(k))}{\partial u(k)} + \left(\frac{\partial(x(k+1))}{\partial u(k)}\right)^T \frac{\partial V^*(x(k+1))}{\partial x(k+1)} = 0. \quad (17)$$

That is

$$2Ru(k) + g^T(x(k)) \frac{\partial V^*(x(k+1))}{\partial x(k+1)} = 0. \quad (18)$$

Then we can obtain the optimal control vector at time step k as follows:

$$u^*(k) = -\frac{1}{2}R^{-1}g^T(x(k)) \frac{\partial V^*(x(k+1))}{\partial x(k+1)}. \quad (19)$$

As the optimal cost function (15) is nonanalytic, it is nearly impossible to obtain $V^*(x(k))$ by solving this function directly [5]. Therefore, we use the VI algorithm [22], [30] to obtain the optimal cost function and the optimal control policy as follows:

The initial cost function $V_0(\cdot)$ is set 0, and the initial control policy $\pi_0(x(k))$ is solved by:

$$\begin{aligned} \pi_0(x(k)) &= u_0(k) \\ &= \arg \min_{u(k)} \{x^T(k)Qx(k) + u^T(k)Ru(k) + V_0(x(k+1))\}, \end{aligned} \quad (20)$$

Then for the iteration index $i = 1, 2, 3, \dots$, the value iteration algorithm iterates between the cost function update

$$\begin{aligned} V_i(x(k)) &= \min_{u(k)} \{x^T(k)Qx(k) + u^T(k)Ru(k) \\ &\quad + V_{i-1}(x(k+1))\} \\ &= x^T(k)Qx(k) + u_{i-1}^T(k)Ru_{i-1}(k) \\ &\quad + V_{i-1}(x(k+1)) \\ &= x^T(k)Qx(k) + \pi_{i-1}^T(x(k))R\pi_{i-1}(x(k)) \\ &\quad + V_{i-1}(x(k+1)), \end{aligned} \quad (21)$$

and the control policy improvement

$$\pi_i(x(k)) = u_i(k) = -\frac{1}{2}R^{-1}g^T(x(k)) \frac{\partial V_i(x(k+1))}{\partial x(k+1)}. \quad (22)$$

Remark 2: k is the time step, whereas i stands for the iteration index of the outer-loop.

According to [19], as the outer-loop runs to infinite iterations, the cost function and the control policy will approach their optimal value. i.e., $V_i \rightarrow V^*$ and $\pi_i \rightarrow \pi^*$, as $i \rightarrow \infty$.

B. THE TRIGGER MECHANISM

Since V_i is a mapping between the state vector and the cost function, π_i is a mapping between the state vector and the control vector, ADP uses a critic network and an action network to approximate (21) and (22), respectively. In this way, as the outer-loop runs to infinite iterations, the critic network can approximate to the optimal cost function V^* , and the action network can approximate to the optimal control policy π^* .

Remark 3: Training the outer-loop means training the critic network to approximate (21) and the action network to approximate (22) alternately.

First, we train the model network to approximate the dynamics function (11) as follows:

$$\hat{x}(k+1) = W^T \sigma(Y^T [x(k)^T, u^T]^T + b), \quad (23)$$

where $\hat{x}(k+1)$ is an estimate of $x(k+1)$. Since $g(x(k))$ in (11) is only a function of $x(k)$, we can simply set

$$u(k) = \bar{u} = [0, 0, \dots, 0], \quad (24)$$

and calculate $g(x(k))$ as follows:

$$g(x(k)) = \left. \frac{\partial x(k+1)}{\partial u(k)} \right|_{u(k)=\bar{u}} \quad (25)$$

So, we can use $\hat{g}(x(k))$ to approximate $g(x(k))$ as follows:

$$\begin{aligned} \hat{g}(x(k)) &= \left. \frac{\partial \hat{x}(k+1)}{\partial u(k)} \right|_{u(k)=\bar{u}} \\ &= W^T \left[1 - \tanh^2(Y^T [x(k)^T, \bar{u}^T]^T + b) \right] \\ &\quad \cdot Y^T \left. \frac{\partial [x(k)^T, u(k)^T]^T}{\partial u(k)} \right|_{u(k)=\bar{u}}, \end{aligned} \quad (27)$$

where $\left. \frac{\partial [x(k)^T, u(k)^T]^T}{\partial u(k)} \right|_{u(k)=\bar{u}} = \begin{bmatrix} 0_{l \times r} \\ I_r \end{bmatrix}$, the term $I_r \in \mathbb{R}^{r \times r}$ represents an identity matrix.

After that, we train the outer-loop. As VI is an offline algorithm, we use tuples to represent a group of state vectors or control vectors. We use $\hat{v}_{i-1}, \chi_i, v_i$, and $\hat{\chi}'_i$ to represent $\{\hat{u}_{i-1}^{(1)}, \hat{u}_{i-1}^{(2)}, \dots, \hat{u}_{i-1}^{(p)}\}, \{x_i^{(1)}, x_i^{(2)}, \dots, x_i^{(p)}\}, \{u_i^{(1)}, u_i^{(2)}, \dots, u_i^{(p)}\}$, and $\{\hat{x}'_i^{(1)}, \hat{x}'_i^{(2)}, \dots, \hat{x}'_i^{(p)}\}$, respectively, where p is the number of samples.

When $i = 0$, the cost function is set as $V_0(\cdot) = 0$, and the initial updated control vector tuple v_0 is calculated by (20). For $i = 1, 2, 3, \dots$, train the critic network with the input χ_i , and the target

$$V_i(\chi_i) = \chi_i^T Q \chi_i + \hat{v}_{i-1}^T R \hat{v}_{i-1} + \hat{V}_{i-1}(\hat{\chi}'_i), \quad (28)$$

The output of the critic network is expressed as:

$$\hat{V}_i(\chi_i) = W_i^{cT} \tanh(Y_i^{cT} \chi_i + b_i^c), \quad (29)$$

where W_i^c, Y_i^c, b_i^c are the weights and biases of the critic network at i -th iteration.

Then it comes to the core problem of how to obtain \hat{v}_{i-1} in (28). There are mainly two methods, i.e., a policy-based method and an algebra method.

1) A POLICY-BASED METHOD

In this method, the action network works as a control policy at each outer-loop. For $i = 1, 2, 3, \dots$, the control vector tuple at i -th iteration \hat{v}_{i-1} is obtained as follows:

$$\hat{v}_{i-1} = \hat{\pi}_{i-1}(\chi_i). \tag{30}$$

The control policy $\hat{\pi}_{i-1}(\cdot)$ is the action network trained at $(i - 1)$ -th iteration, which is trained with the input χ_{i-1} , and the target expressed as:

$$v_{i-1} = -\frac{1}{2}R^{-1}\hat{g}^T(\chi_{i-1})\frac{\partial \hat{V}_{i-1}(\hat{\chi}'_{i-1})}{\partial \hat{\chi}'_{i-1}}, \tag{31}$$

where $\hat{g}^T(\chi_{i-1})$ is calculated by (27) with the input of χ_{i-1} , the term $\frac{\partial \hat{V}_{i-1}(\hat{\chi}'_{i-1})}{\partial \hat{\chi}'_{i-1}}$ can be calculated by the chain rule of the critic network as follows:

$$\frac{\partial \hat{V}_{i-1}(\hat{\chi}'_{i-1})}{\partial \hat{\chi}'_{i-1}} = W_{i-1}^{cT} \left(1 - \tanh^2(Y_{i-1}^{cT}\hat{\chi}'_{i-1} + b_{i-1}^c) \right) Y_{i-1}^{cT}. \tag{32}$$

Then the term \hat{v}_{i-1} can be expressed as follows:

$$\begin{aligned} \hat{v}_{i-1} &= \hat{\pi}_{i-1}(\chi_i) \\ &= W_{i-1}^{aT} \tanh(Y_{i-1}^{aT}\chi_i + b_{i-1}^a), \end{aligned} \tag{33}$$

where W_{i-1}^a , Y_{i-1}^a and b_{i-1}^a are the weights and biases of the action network at $(i - 1)$ -th iteration.

The scheme is illustrated in Table 1, where the letters in bold are obtained by training the neural network. In this scheme, the action network works as a control policy that maps the state vector tuples to the control vector tuples. Even though the input state vector tuple χ_i is changed, the control policy can generate the corresponding control vector tuple \hat{v}_{i-1} to update the critic network at the i -th iteration. i.e., the state vector tuples can be different in each outer-loop.

The policy-based method can explore enough different state vectors in the state space, so the control policy obtained in this way has a good generalization ability. But in this method, we train both the critic network and action network in each outer-loop, which is time-consuming. Furthermore, there are action network approximation errors in each outer-loop training process [4].

2) AN ALGEBRA METHOD

The other method is to utilize the $(i - 1)$ -th updated control vector tuple v_{i-1} as the control vector tuple at the i -th iteration, i.e., $\hat{v}_{i-1} = v_{i-1}$. v_{i-1} can be calculated directly by the algebra function:

$$v_{i-1} = -\frac{1}{2}R^{-1}\hat{g}^T(\chi_{i-1})\frac{\partial \hat{V}_{i-1}(\hat{\chi}'_{i-1})}{\partial \hat{\chi}'_{i-1}}, \tag{34}$$

The scheme is illustrated in Table 2. Since the $(i - 1)$ -th updated control vector tuple v_{i-1} is corresponding to χ_{i-1} , if we want to utilize v_{i-1} as the control vector tuple of i -th iteration \hat{v}_{i-1} , χ_i must be the same with χ_{i-1} .

TABLE 1. Scheme of training the outer-loop with the policy-based method.

Outer-loop step	State vector tuple	Process
0	χ_0	$\hat{V}_0 \rightarrow \nu_0 \rightarrow \hat{\pi}_0$
1	χ_1	$\hat{\nu}_0 = \hat{\pi}_0(\chi_1) \rightarrow \hat{V}_1 \rightarrow \nu_1 \rightarrow \hat{\pi}_1$
2	χ_2	$\hat{\nu}_1 = \hat{\pi}_1(\chi_2) \rightarrow \hat{V}_2 \rightarrow \nu_2 \rightarrow \hat{\pi}_2$
\vdots	\vdots	\vdots
$i - 1$	χ_{i-1}	$\hat{\nu}_{i-2} = \hat{\pi}_{i-2}(\chi_{i-1}) \rightarrow \hat{V}_{i-1} \rightarrow \nu_{i-1} \rightarrow \hat{\pi}_{i-1}$
i	χ_i	$\hat{\nu}_{i-1} = \hat{\pi}_{i-1}(\chi_i) \rightarrow \hat{V}_i \rightarrow \nu_i \rightarrow \hat{\pi}_i$
$i + 1$	χ_{i+1}	$\hat{\nu}_i = \hat{\pi}_i(\chi_{i+1}) \rightarrow \hat{V}_{i+1} \rightarrow \nu_{i+1} \rightarrow \hat{\pi}_{i+1}$
\vdots	\vdots	\vdots
i_{max}	$\chi_{i_{max}}$	$\hat{\nu}_{i_{max}-1} = \hat{\pi}_{i_{max}-1}(\chi_{i_{max}}) \rightarrow \hat{V}_{i_{max}} \rightarrow \nu_{i_{max}} \rightarrow \hat{\pi}_{i_{max}}$

TABLE 2. Scheme of training the outer-loop with the algebra method.

Outer-loop step	State vector tuple	Process
0	χ_0	$\hat{V}_0 \rightarrow \nu_0$
1	$\chi_1 (= \chi_0)$	$\hat{\nu}_0 = \nu_0 \rightarrow \hat{V}_1 \rightarrow \nu_1$
2	$\chi_2 (= \chi_0)$	$\hat{\nu}_1 = \nu_1 \rightarrow \hat{V}_2 \rightarrow \nu_2$
\vdots	\vdots	\vdots
$i - 1$	$\chi_{i-1} (= \chi_0)$	$\hat{\nu}_{i-2} = \nu_{i-2} \rightarrow \hat{V}_{i-1} \rightarrow \nu_{i-1}$
i	$\chi_i (= \chi_0)$	$\hat{\nu}_{i-1} = \nu_{i-1} \rightarrow \hat{V}_i \rightarrow \nu_i$
$i + 1$	$\chi_{i+1} (= \chi_0)$	$\hat{\nu}_i = \nu_i \rightarrow \hat{V}_{i+1} \rightarrow \nu_{i+1}$
\vdots	\vdots	\vdots
i_{max}	$\chi_{i_{max}} (= \chi_0)$	$\hat{\nu}_{i_{max}-1} = \nu_{i_{max}-1} \rightarrow \hat{V}_{i_{max}} \rightarrow \nu_{i_{max}} \rightarrow \hat{\pi}_{i_{max}}$

i.e., $\chi_i = \chi_{i-1} = \dots = \chi_2 = \chi_1 = \chi_0$. This method cannot explore enough state vectors in the state space, so the control policy's generalization ability is limited. But in this method, only the critic network is trained in each outer-loop. The outer-loop can be trained faster than the policy-based way. Furthermore, since we calculate the control vector tuples in an algebra way, there are no action network approximation errors in the outer-loop training process.

3) THE PROPOSED TRIGGER MECHANISM

To keep a tradeoff between the generalization ability of the obtained control policy and the time consumption of the outer-loop training process, we propose a trigger mechanism. The training scheme is shown in Table 3, where there is a trigger signal at the i -th outer-loop. Generally, we keep the state vector tuples fixed and train the outer-loop with the algebra method. If the trigger signal is generated at the i -th outer-loop, we train the action network with the input χ_{i-1} and the target v_{i-1} to obtain the control policy $\hat{\pi}_{i-1}$. Then we feed the newly chosen χ_i into the action network to obtain \hat{v}_{i-1} , i.e., $\hat{v}_{i-1} = \hat{\pi}_{i-1}(\chi_i)$. It is paramount to note that, the trigger signal can be in many forms, i.e., a fixed interval outer-loop steps, or a pre-specified accuracy.

TABLE 3. Scheme of training the outer-loop with the trigger mechanism.

Outer-loop step	State vector tuple	Process
0	$\chi_0 (= \chi_0)$	$\hat{V}_0 \rightarrow \nu_0$
1	$\chi_1 (= \chi_0)$	$\hat{\nu}_0 = \nu_0 \rightarrow \hat{V}_1 \rightarrow \nu_1$
2	$\chi_2 (= \chi_0)$	$\hat{\nu}_1 = \nu_1 \rightarrow \hat{V}_2 \rightarrow \nu_2$
⋮	⋮	⋮
$i - 1$	$\chi_{i-1} (= \chi_0)$	$\hat{\nu}_{i-2} = \nu_{i-2} \rightarrow \hat{V}_{i-1} \rightarrow \nu_{i-1}$
i (Trigger signal)	$\chi_i (= \chi_i)$	$\hat{\pi}_{i-1} \rightarrow \hat{\nu}_{i-1} = \hat{\pi}_{i-1}(\chi_i) \rightarrow \hat{V}_i \rightarrow \nu_i$
$i + 1$	$\chi_{i+1} (= \chi_i)$	$\hat{\nu}_i = \nu_i \rightarrow \hat{V}_{i+1} \rightarrow \nu_{i+1}$
⋮	⋮	⋮
i_{max}	$\chi_{i_{max}}$	$\hat{\nu}_{i_{max}-1} = \nu_{i_{max}-1} \rightarrow \hat{V}_{i_{max}} \rightarrow \nu_{i_{max}}$

The trigger signal can also be a variable that changes along with the outer-loop’s training process. Herein, we consider a fixed interval outer-loop step. i.e., the trigger interval is a fixed number σ .

Since we have implemented the trigger mechanism to the outer-loop training process, the outer-loop can be trained faster than the policy-based method and the obtained control policy has a better generalization ability than the algebra method. Note that both the policy-based method and the algebra method are special cases of the proposed trigger mechanism that the trigger interval is 1 and infinity, respectively.

IV. THE OVERALL NoFt-VI-ADP ALGORITHM

In the process of NoFt-VI-ADP, we first train the model network with the less evolved GA-BP algorithm and then train the outer-loop with the proposed trigger mechanism. After that, we obtain a near-optimal control policy and use it to control the system directly. To a specific system, if the dimensions of the state vector and control vector are n_1 and n_3 , respectively, and the number of the hidden layer neurons is selected as n_2 , then the action network is an $n_1 - n_2 - n_3$ structure. The system decides both n_1 and n_3 , and the researchers only choose n_2 , then the time complexity of the action network is illustrated as $O(n_1 \times n_2 + n_2 + n_2 \times n_3) = O(n_2)$ [33], [34]. The NoFt-VI-ADP algorithm scheme is shown in Fig. 5, and a summary of the procedure is shown in Algorithm 2.

V. CASE STUDY

In this section, to verify the superiority of the proposed NoFt-VI-ADP algorithm compared with the conventional VI-ADP, we apply both of them to a linear system and a nonlinear system.

A. EXAMPLE 1: DISCRETE-TIME LINEAR SYSTEM

Considering a linear system [22]

$$x(k + 1) = \begin{bmatrix} 0 & 0.4 \\ 0.3 & 1 \end{bmatrix} x(k) + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u(k), \quad (35)$$

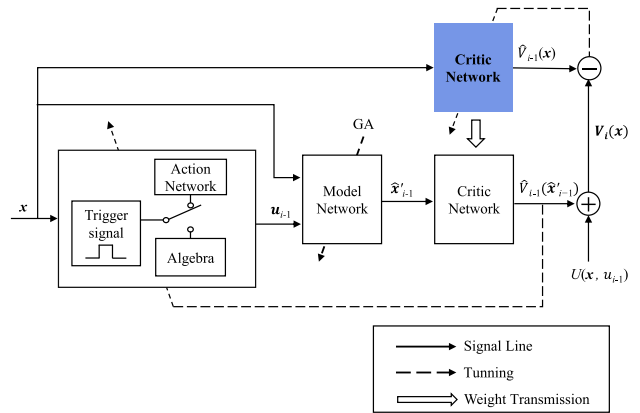


FIGURE 5. Scheme of the NoFt-VI-ADP algorithm. The model network parameters are initialized by GA. Whether the action network is trained or not in each outer-loop depends on the trigger signal.

where $x(k) = [x_1(k), x_2(k)]^T$. The utility function keeps a tradeoff between the state trajectory errors and the control energy consumptions. In this example, the weight matrices are given as follows:

$$Q = \begin{bmatrix} 0.2 & 0 \\ 0 & 0.2 \end{bmatrix}, \quad R = 1. \quad (36)$$

Then, the utility function is expressed as follows:

$$U(x(k), u(k)) = 0.2x_1(k)^2 + 0.2x_2(k)^2 + u(k)^2. \quad (37)$$

The state space of the system is selected as $\Omega = \{(x_1, x_2) : -1 \leq x_1 \leq 1, -1 \leq x_2 \leq 1\}$. Hyper-parameters required in this example are shown in Table 4.

Firstly, we train the model network. We randomly sample N groups of (x_s, u_s, x'_s) from the system and train the model network with both the less evolved GA-BP and the randomly initialized BP method. The convergence trajectories of the first column of the less evolved GA-BP model network output layer’s weights are shown in Fig. 6. It shows that the model

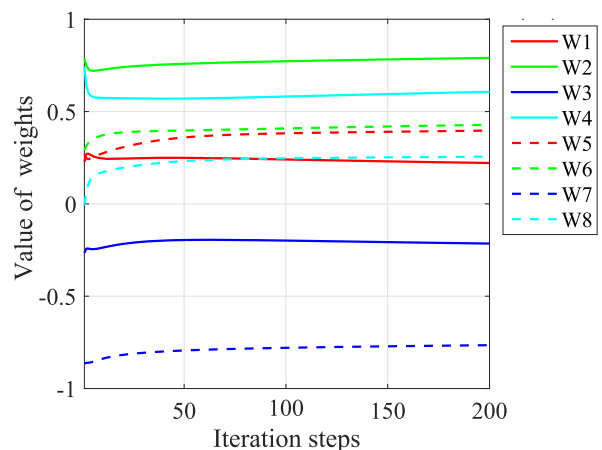


FIGURE 6. The model network weights convergence trajectories of Example 1.

Algorithm 2 The NoFt-VI-ADP Algorithm

- Step 1. Initialization: Construct the model network, the critic network, and the action network. Initialize the parameters N, p, Q, R, ξ, i_{max} .
- Step 2. Dataset of the model network: Sample N groups of (x_s, u_s, x'_s) , from the system to constitute the dataset.
- Step 3. Train the model network with GA initialization: Use GA to initialize the weights and biases of the model network with Algorithm 1. Then, train the model network with the input (x, u) and the target x' .
- Step 4. Initialize the outer-loop: Randomly choose a vector tuple of p state vectors $\chi_0 = \{x_0^{(1)}, x_0^{(2)}, \dots, x_0^{(p)}\}$ from the state space. Set the outer-loop iteration index $i = 0$, the initial cost $V_0(\chi_0) = 0$ and calculate the initial control vector tuple $v_0 = \{u_0^{(1)}, u_0^{(2)}, \dots, u_0^{(p)}\}$ by (20). Train the critic network with the input χ_0 and the target $V_0(\chi_0)$. Calculate $\hat{g}_0(\chi_0)$ according to (27).
- Step 5. $i = i + 1$.
- step 6. The trigger mechanism: If there is no trigger signal, set $\chi_i = \chi_{i-1}, \hat{v}_{i-1} = v_{i-1}, \hat{g}_i(\chi_i) = \hat{g}_{i-1}(\chi_{i-1})$ and go to step 7. Otherwise, train the action network with the input χ_{i-1} and the target v_{i-1} to get the control policy $\hat{\pi}_{i-1}$. Then randomly choose a new state vector tuple χ_i from the state space, and feed χ_i into the action network to obtain the corresponding control vector tuple $\hat{v}_{i-1} = \hat{\pi}_{i-1}(\chi_i)$. Calculate $\hat{g}(\chi_i)$ according to (27).
- Step 7. Train the critic network: Feed χ_i and v_{i-1} into the model network to get χ'_i . Feed χ'_i into the critic network to get $\hat{V}_{i-1}(\chi'_i)$, and calculate the target of critic network $V_i(\chi_i)$ through (28). Train the critic network with the input χ_i and the target $V_i(\chi_i)$.
- Step 8. Update the control vector tuple: Calculate v_i with (34).
- Step 9. The terminal condition: If $i = i_{max}$, or
- $$\max \left(\left| \hat{V}_i(\chi_i) - \hat{V}_{i-1}(\chi_i) \right| \right) \leq \xi,$$
- go to Step 10; otherwise, go to Step 5.
- Step 10. Train the action network: Train the action network with the input χ_i and the target v_i to obtain the control policy π_{NoFt} .
- Step 11. Finish the algorithm.

network has converged in 200 iteration steps. The model network performance comparison of these two methods is shown in Fig. 7. As shown in the figure, at the end of the training process, the MSE of the randomly initialized BP method is 10^{-7} , whereas it is 10^{-10} for the less evolved GA-BP. It indicates that the model network trained by the less evolved GA-BP can approximate the dynamics of the system more accurately than that of randomly initialized BP.

Thereafter, we train the outer-loop corresponding to the former two trained model networks with the trigger-mechanism and the policy-based algorithm to obtain the control policies π_{NoFt} and π_{VI} . To compare the performance of these two control policies, we randomly choose

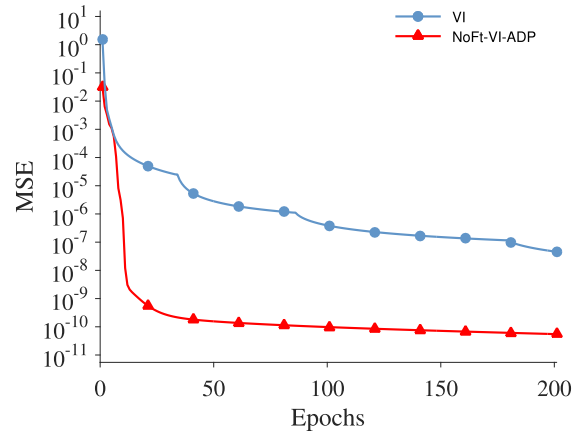


FIGURE 7. The model network performance comparison of Example 1.

10 groups of initial state vectors from the state space. To each group of these state vectors, we use π_{NoFt} and π_{VI} to control the system for M time steps and calculate the multistep lookahead cost as follows [18]

$$\sum_{j=0}^M U(x(k+j), u(k+j)). \tag{38}$$

In this way, we obtain the cost of π_{NoFt} represented as V_{NoFt} and the cost of π_{VI} represented as V_{VI} , respectively. Since it is a linear system and the utility function is in a quadratic form, we can solve a corresponding Algebraic Riccati Equation (ARE) to obtain the optimal control policy

$$\pi^*(x) = [-0.1252, -0.4488]x. \tag{39}$$

And the optimal cost

$$V^* = x^T \begin{bmatrix} 0.2376 & 0.1346 \\ 0.1346 & 0.7165 \end{bmatrix} x. \tag{40}$$

Then we subtract V^* from V_{VI} and V_{NoFt} . The results are shown in Fig. 8, where NoFt- σ means the trigger interval of NoFt-VI-ADP is σ . As we can see from the figure, all the NoFt methods can approximate the optimal control better than VI-ADP. The best one is NoFt-4 in this example. When the trigger interval is 4, it not only avoids too many times of action network approximation errors in every outer-loop training step but also explores enough different state vectors in the state space. To the initial state $x_0 = [1, -1]^T$, the state and control trajectories belong to π_{NoFt-4} and π^* are shown in Fig. 9 and Fig. 10, which imply that π_{NoFt-4} stabilizes the system and its state and control trajectories are generally coincide with that of π^* .

Finally, we compare the time consumption between NoFt-VI-ADP and VI-ADP. The algorithms are implemented on a computer with an Intel i7 CPU. As shown in Fig. 11, although NoFt-VI-ADP costs more time than VI-ADP to train the model network, it costs less in the outer-loop training process when σ is bigger than 1. On the whole, the time consumption

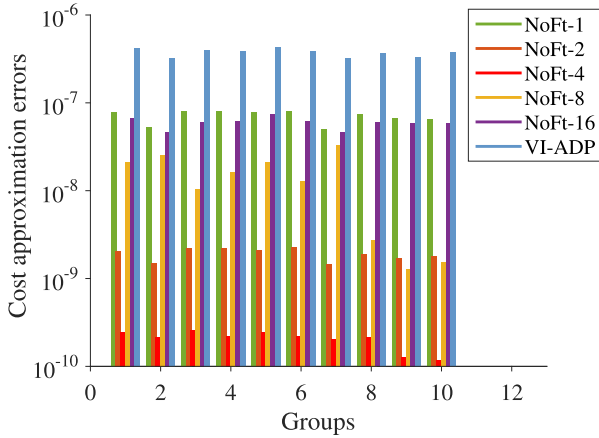


FIGURE 8. The cost approximation errors comparisons of Example 1. NoFt- σ means the trigger interval of NoFt-VI-ADP is σ . The differences are obtained by subtracting the optimal costs V^* from the corresponding near-optimal costs.

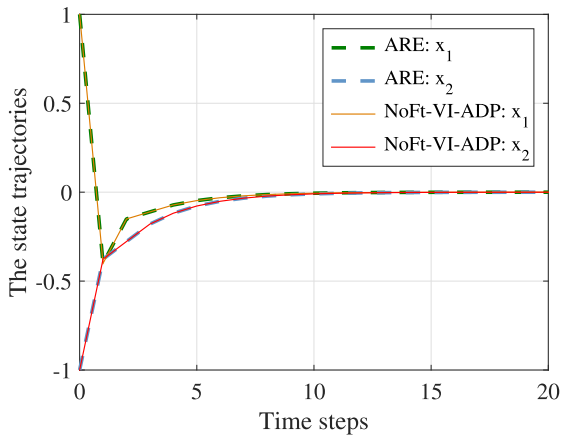


FIGURE 9. The state trajectories comparisons of Example 1 with $\sigma = 4$.

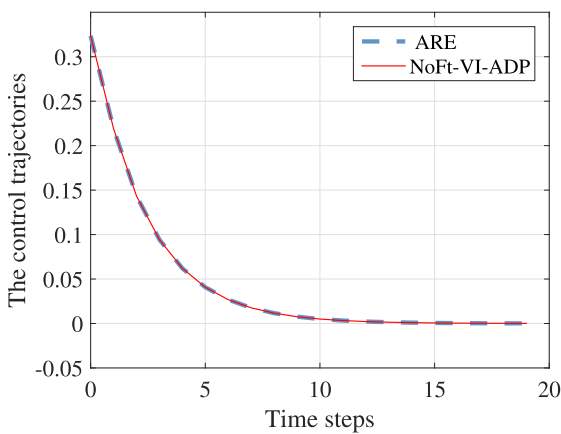


FIGURE 10. The control trajectory comparison of Example 1 with $\sigma = 4$.

of NoFt-VI-ADP is shorter than VI-ADP. i.e., NoFt-VI-ADP can be trained faster than VI-ADP.

The numerical results illustrate that, compared with the conventional VI-ADP, NoFt-VI-ADP can obtain a nearer optimal control policy and be trained faster.

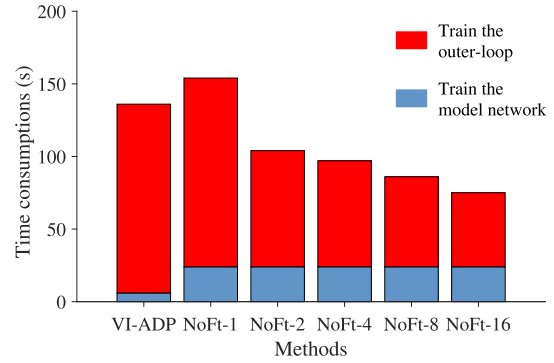


FIGURE 11. The time consumption comparisons of Example 1.

B. EXAMPLE 2: DISCRETE-TIME NONLINEAR SYSTEM

The nonlinear example is chosen from [32] with modifications. The system functions are given as follows:

$$\begin{aligned} \begin{bmatrix} x_1(k+1) \\ x_2(k+1) \end{bmatrix} &= \begin{bmatrix} 0.9x_1(k) + 0.1x_2(k) \cos(x_1(k)) \\ -0.05x_1(k) + 0.95x_2(k) + 0.1x_2(k) \sin(x_1(k)) \end{bmatrix} \\ &+ \begin{bmatrix} -0.1 \\ 0 \end{bmatrix} u(k), \end{aligned} \tag{41}$$

and the utility function is in a quadratic form with

$$Q = \begin{bmatrix} 0.1 & 0 \\ 0 & 0.1 \end{bmatrix}, \quad R = 0.1. \tag{42}$$

Then, the utility function is expressed as follows:

$$U(x(k), u(k)) = 0.1x_1(k)^2 + 0.1x_2(k)^2 + 0.1u(k)^2. \tag{43}$$

The state space of the system is selected as $\Omega = \{(x_1, x_2) : -1 \leq x_1 \leq 1, -1 \leq x_2 \leq 1\}$. We randomly choose N groups of (x_s, u_s, x'_s) from the system as the training set. Hyper-parameters required in this example are shown in Table 5.

TABLE 4. Parameters of the linear system.

Items	VI-ADP	NoFt-VI-ADP
Model network	Configuration: 3-8-2 Epochs: $E^m = 200$ Dataset scale: $N = 2000$	Configuration: 3-8-2 Epochs: $E^m = 200$ Dataset scale: $N = 2000$ Population: $Q = 100$ Generation: $G = 20$ Crossover rate: $P_c = 0.4$ Mutation rate: $P_m = 0.01/0.002^a$
Action network	Configuration: 2-10-1 Epochs: $E^a = 200$	Configuration: 2-10-1 Epochs: $E^a = 200$
Critic network	Configuration: 2-10-1 Epochs: $E^c = 100$	Configuration: 2-10-1 Epochs: $E^c = 100$
Outer-loop	Accuracy: $\xi = 10^{-6}$ Max loops: $i_{max} = 30$ Samples: $p = 2000$ lookahead steps: $M = 30$	Accuracy: $\xi = 10^{-6}$ Max loops: $i_{max} = 30$ Samples: $p = 2000$ Trigger interval: $\sigma = 2, 4, 8, 16$ lookahead steps: $M = 30$

^aThe former half-generations use the former mutation rate, and the later half-generations use the latter mutation rate which is behind the slash.

First, we train the model network with both the less evolved GA-BP algorithm and the randomly initialized BP algorithm.

TABLE 5. Parameters of the nonlinear system.

Items	VI-ADP	NoFt-VI-ADP
Model network	Configuration: 3-20-2 Epochs: $E^m = 200$ Dataset scale: $N = 2000$	Configuration: 3-20-2 Epochs: $E^m = 200$ Dataset scale: $N = 2000$ Population: $Q = 100$ Generation: $G = 20$ Crossover rate: $P_c = 0.4$ Mutation rate: $P_m = 0.01/0.002^a$
Action network	Configuration: 2-20-1 Epochs: $E^a = 200$	Configuration: 2-10-1 Epochs: $E^a = 200$
Critic network	Configuration: 2-20-1 Epochs: $E^c = 100$	Configuration: 2-10-1 Epochs: $E^c = 100$
Outer-loop	Accuracy: $\xi = 10^{-3}$ Max loops: $i_{max} = 50$ Samples: $p = 2000$ lookahead steps: $M = 60$	Accuracy: $\xi = 10^{-3}$ Max loops: $i_{max} = 50$ Samples: $p = 2000$ lookahead steps: $M = 60$ Trigger interval: $\sigma = 2, 4, 8, 16$

^aThe former half-generations use the former mutation rate, and the later half-generations use the latter mutation rate which is behind the slash.

The convergence trajectories of the first column of the less evolved GA-BP model network output layer’s weights are shown in Fig. 12. It shows that the model network has converged in 200 iteration steps. The model network performance comparison is shown in Fig. 13. As shown, at the end of the training process, the MSE of the randomly initialized BP algorithm is 10^{-6} , whereas it is 10^{-8} for the less evolved GA-BP algorithm. This implies that considering the nonlinear system, the less evolved GA-BP can approximate its dynamics better than the randomly initialized BP algorithm.

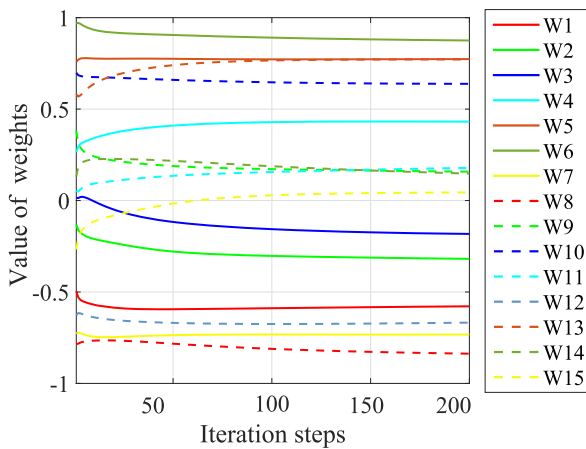


FIGURE 12. The model network weights convergence trajectories of Example 2.

After that, we train the outer-loop with the trigger-mechanism and the policy-based algorithm to obtain π_{NoFt} and π_{VI} , respectively. Then we use these control policies to control the system to obtain their corresponding costs V_{NoFt} and V_{VI} . To compare the approximation accuracy of π_{NoFt} and π_{VI} , we subtract V_{NoFt} from V_{VI} . The results are shown in Fig. 14. It shows that all the difference is positive, which means that π_{NoFt} approximates the optimal control policy better than π_{VI} . In this example, among all the trigger intervals, the best trigger interval is 8. To the initial state

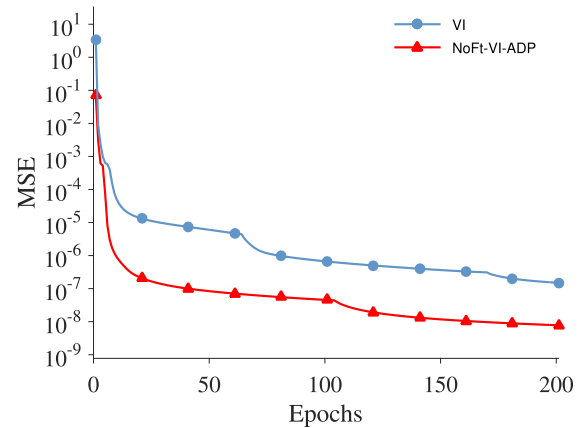


FIGURE 13. The model network performance comparison of Example 2.

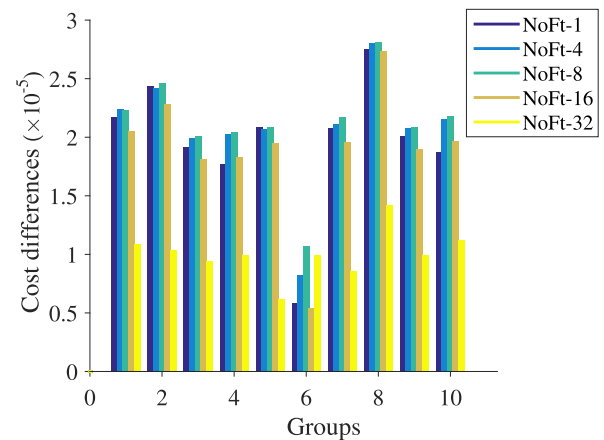


FIGURE 14. The cost differences between VI-ADP and NoFt-VI-ADP of Example 2. NoFt- σ means the trigger interval of NoFt-VI-ADP is σ . To each group of the initial state vector, the difference is obtained by subtracting the cost of NoFt-VI-ADP from the corresponding cost of VI-ADP.

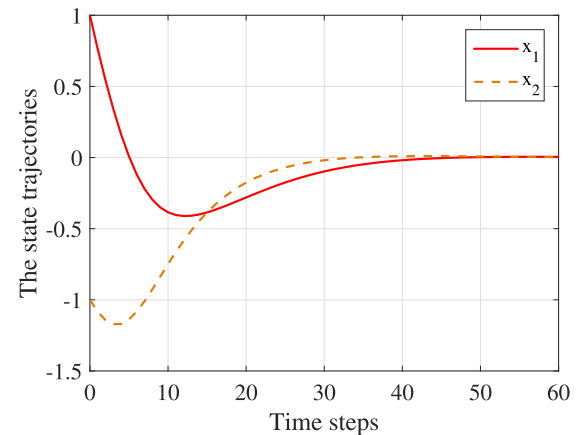


FIGURE 15. The NoFt-VI-ADP state trajectories of Example 2 with $\sigma = 8$.

$x_0 = [1, -1]^T$, the state and control trajectories of π_{NoFt-8} are shown in Fig. 15 and Fig. 16, which imply that π_{NoFt-8} stabilizes the system.

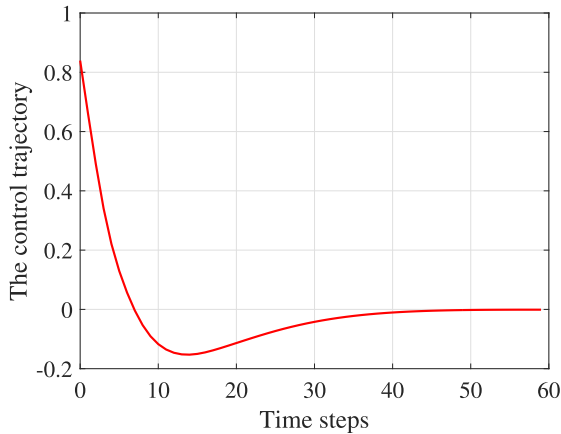


FIGURE 16. The NoFt-VI-ADP control trajectory of Example 2 with $\sigma = 8$.

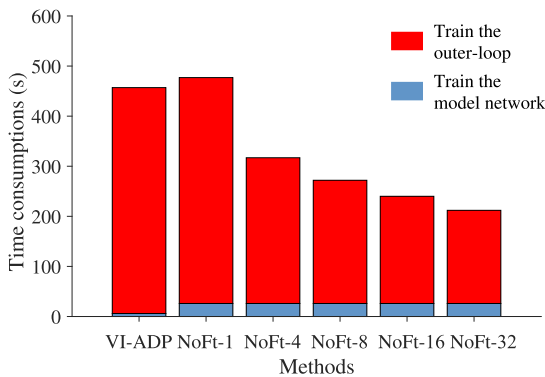


FIGURE 17. The time consumption comparisons of Example 2.

Besides, the time consumption comparison between NoFt-VI-ADP and VI-ADP is demonstrated in Fig. 17. As we can see, when σ is bigger than 1, the total time consumptions of NoFt-VI-ADP are less than VI-ADP.

The numerical results illustrate that considering the nonlinear system, the proposed NoFt-VI-ADP can generate a more accurate control policy and be trained faster than VI-ADP.

VI. CONCLUSION

In this research, we proposed a nearer optimal and faster trained VI-ADP algorithm NoFt-VI-ADP, for discrete-time nonlinear systems. Before training the model network, we use a less evolved GA to evolve the weights and biases for a few generations, which can improve the approximation accuracy of the model network. Different from the conventional VI-ADP algorithm that trains the action network in each outer-loop, we proposed a trigger mechanism to decide when to train the action network, which can save much training time. The simulation results show that the proposed algorithm can generate a nearer optimal control policy and save more training time than the conventional VI-ADP algorithm.

The proposed NoFt-VI-ADP algorithm relies on the complete information about the state of the investigated system,

and it works only on offline ADP. The implementation of the proposed methods on partial observation systems and online optimal control problems will be researched in our future works.

REFERENCES

- [1] T. Nguyen-Van, "A discrete-time state estimation for nonlinear systems with noises," *IEEE Access*, vol. 8, pp. 147089–147096, 2020, doi: [10.1109/ACCESS.2020.3014377](https://doi.org/10.1109/ACCESS.2020.3014377).
- [2] M. Ha, D. Wang, and D. Liu, "Generalized value iteration for discounted optimal control with stability analysis," *Syst. Control Lett.*, vol. 147, Jan. 2021, Art. no. 104847, doi: [10.1016/j.sysconle.2020.104847](https://doi.org/10.1016/j.sysconle.2020.104847).
- [3] Y. Wang, H. R. Karimi, H.-K. Lam, and H. Yan, "Fuzzy output tracking control and filtering for nonlinear discrete-time descriptor systems under unreliable communication links," *IEEE Trans. Cybern.*, vol. 50, no. 6, pp. 2369–2379, Jun. 2020, doi: [10.1109/TCYB.2019.2920709](https://doi.org/10.1109/TCYB.2019.2920709).
- [4] A. Heydari, "Stability analysis of optimal adaptive control using value iteration with approximation errors," *IEEE Trans. Autom. Control*, vol. 63, no. 9, pp. 3119–3126, Sep. 2018, doi: [10.1109/TAC.2018.2790260](https://doi.org/10.1109/TAC.2018.2790260).
- [5] Q. Wei, F. L. Lewis, D. Liu, R. Song, and H. Lin, "Discrete-time local value iteration adaptive dynamic programming: Convergence analysis," *IEEE Trans. Syst., Man, Cybern. Syst.*, vol. 48, no. 6, pp. 875–891, Jun. 2018, doi: [10.1109/TSMC.2016.2623766](https://doi.org/10.1109/TSMC.2016.2623766).
- [6] P. J. Werbos, "Advanced forecasting methods for global crisis warning and models of intelligence," *General Syst. Yearbook*, vol. 22, pp. 25–38, Jan. 1977.
- [7] J. Liang, G. K. Venayagamoorthy, and R. G. Harley, "Wide-area measurement based dynamic stochastic optimal power flow control for smart grids with high variability and uncertainty," *IEEE Trans. Smart Grid*, vol. 3, no. 1, pp. 59–69, Mar. 2012, doi: [10.1109/TSG.2011.2174068](https://doi.org/10.1109/TSG.2011.2174068).
- [8] D. V. Prokhorov and D. C. Wunsch, "Adaptive critic designs," *IEEE Trans. Neural Netw.*, vol. 8, no. 5, pp. 997–1007, Sep. 1997, doi: [10.1109/72.623201](https://doi.org/10.1109/72.623201).
- [9] X. Xu, Z. Hou, C. Lian, and H. He, "Online learning control using adaptive critic designs with sparse kernel machines," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 24, no. 5, pp. 762–775, May 2013, doi: [10.1109/TNNLS.2012.2236354](https://doi.org/10.1109/TNNLS.2012.2236354).
- [10] R. Enns and J. Si, "Helicopter trimming and tracking control using direct neural dynamic programming," *IEEE Trans. Neural Netw.*, vol. 14, no. 4, pp. 929–939, Jul. 2003, doi: [10.1109/TNN.2003.813839](https://doi.org/10.1109/TNN.2003.813839).
- [11] J. Si and Y.-T. Wang, "On-line learning control by association and reinforcement," *IEEE Trans. Neural Netw.*, vol. 12, no. 2, pp. 264–276, Mar. 2001, doi: [10.1109/IJCNN.2000.861307](https://doi.org/10.1109/IJCNN.2000.861307).
- [12] M. Geist and O. Pietquin, "Algorithmic survey of parametric value function approximation," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 24, no. 6, pp. 845–867, Jun. 2013, doi: [10.1109/TNNLS.2013.2247418](https://doi.org/10.1109/TNNLS.2013.2247418).
- [13] K. Zhang, H. Zhang, Y. Mu, and C. Liu, "Decentralized tracking optimization control for partially unknown fuzzy interconnected systems via reinforcement learning method," *IEEE Trans. Fuzzy Syst.*, early access, Jan. 13, 2020, doi: [10.1109/TFUZZ.2020.2966418](https://doi.org/10.1109/TFUZZ.2020.2966418).
- [14] K. Zhang, H.-g. Zhang, Y. Cai, and R. Su, "Parallel optimal tracking control schemes for mode-dependent control of coupled Markov jump systems via integral RL method," *IEEE Trans. Autom. Sci. Eng.*, vol. 17, no. 3, pp. 1332–1342, Jul. 2020, doi: [10.1109/TASE.2019.2948431](https://doi.org/10.1109/TASE.2019.2948431).
- [15] D. Vrabie, O. Pastravanu, M. Abu-Khalaf, and F. L. Lewis, "Adaptive optimal control for continuous-time linear systems based on policy iteration," *Automatica*, vol. 45, no. 2, pp. 477–484, Feb. 2009, doi: [10.1016/j.automatica.2008.08.017](https://doi.org/10.1016/j.automatica.2008.08.017).
- [16] D. Vrabie and F. L. Lewis, "Generalized policy iteration for continuous time systems," in *Proc. Int. Joint Conf. Neural Netw.*, Jun. 2009, pp. 3224–3231, doi: [10.1109/IJCNN.2009.5178964](https://doi.org/10.1109/IJCNN.2009.5178964).
- [17] K. G. Vamvoudakis and F. L. Lewis, "Online actor-critic algorithm to solve the continuous-time infinite horizon optimal control problem," *Automatica*, vol. 46, no. 5, pp. 878–888, May 2010, doi: [10.1016/j.automatica.2010.02.018](https://doi.org/10.1016/j.automatica.2010.02.018).
- [18] D. Liu and Q. Wei, "Policy iteration adaptive dynamic programming algorithm for discrete-time nonlinear systems," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 25, no. 3, pp. 621–634, Mar. 2014, doi: [10.1109/TNNLS.2013.2281663](https://doi.org/10.1109/TNNLS.2013.2281663).
- [19] A. Al-Tamimi, F. L. Lewis, and M. Abu-Khalaf, "Discrete-time nonlinear HJB solution using approximate dynamic programming: Convergence proof," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 38, no. 4, pp. 943–949, Aug. 2008, doi: [10.1109/TSMCB.2008.926614](https://doi.org/10.1109/TSMCB.2008.926614).

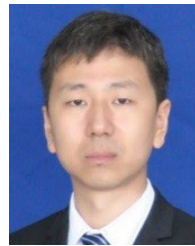
- [20] T. Dierks, B. T. Thumati, and S. Jagannathan, "Optimal control of unknown affine nonlinear discrete-time systems using offline-trained neural networks with proof of convergence," *Neural Netw.*, vol. 22, nos. 5–6, pp. 851–860, Jul. 2009, doi: [10.1016/j.neunet.2009.06.014](https://doi.org/10.1016/j.neunet.2009.06.014).
- [21] H. Zhang, Y. Luo, and D. Liu, "Neural-Network-Based near-optimal control for a class of discrete-time affine nonlinear systems with control constraints," *IEEE Trans. Neural Netw.*, vol. 20, no. 9, pp. 1490–1503, Sep. 2009, doi: [10.1109/TNN.2009.2027233](https://doi.org/10.1109/TNN.2009.2027233).
- [22] H. Li and D. Liu, "Optimal control for discrete-time affine non-linear systems using general value iteration," *IET Control Theory Appl.*, vol. 6, no. 18, pp. 2725–2736, Dec. 2012, doi: [10.1049/iet-cta.2011.0783](https://doi.org/10.1049/iet-cta.2011.0783).
- [23] Q. Wei, D. Liu, and Q. Lin, "Discrete-time local value iteration adaptive dynamic programming: Admissibility and termination analysis," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 11, pp. 2490–2502, Nov. 2017, doi: [10.1109/TNNLS.2016.2593743](https://doi.org/10.1109/TNNLS.2016.2593743).
- [24] C. Li, J. Ding, C. Liu, and F. L. Lewis, "A novel on-line VI-ADP for nonlinear discrete-time Systems*," in *Proc. IEEE 15th Int. Conf. Control Autom. (ICCA)*, Jul. 2019, pp. 1176–1190.
- [25] B. Luo, Y. Yang, H.-N. Wu, and T. Huang, "Balancing value iteration and policy iteration for discrete-time control," *IEEE Trans. Syst., Man, Cybern. Syst.*, vol. 50, no. 11, pp. 3948–3958, Nov. 2020.
- [26] H. Jing, X. Xu, and J. Wang, "Research on genetic neural network algorithm and its application," in *Proc. Int. Conf. Virtual Reality Intell. Syst. (ICVRIS)*, Aug. 2018, pp. 223–226.
- [27] X. Dai, J. Wang, and J. Zhao, "Research on multi-robot task allocation based on BP neural network optimized by genetic algorithm," in *Proc. 5th Int. Conf. Inf. Sci. Control Eng. (ICISCE)*, Jul. 2018, pp. 478–481.
- [28] Q. Jiang, R. Huang, Y. Huang, S. Chen, Y. He, L. Lan, and C. Liu, "Application of BP neural network based on genetic algorithm optimization in evaluation of power grid investment risk," *IEEE Access*, vol. 7, pp. 154827–154835, 2019, doi: [10.1109/ACCESS.2019.2944609](https://doi.org/10.1109/ACCESS.2019.2944609).
- [29] R. K. Belew, J. Mcinerney, and N. N. Schraudolph, "Evolving networks: Using genetic algorithm with connectionist learning," in *Proc. ICVIRS*, San Diego, CA, USA: Univ. California, Feb. 1991, p. 547.
- [30] D. Liu, Q. Wei, D. Wang, X. Yang, and H. Li, "Value iteration ADP for discrete-time nonlinear systems," in *Adaptive Dynamic Programming With Applications in Optimal Control*. Cham, Switzerland: Springer, 2017, pp. 37–87.
- [31] F. L. Lewis and V. L. Syrmos, *Optimal Control*, vol. 1, no. 2. New York, NY, USA: Wiley, 1995, pp. 1045–1206.
- [32] L. Dong, J. Yan, X. Yuan, H. He, and C. Sun, "Functional nonlinear model predictive control based on adaptive dynamic programming," *IEEE Trans. Cybern.*, vol. 49, no. 12, pp. 4206–4218, Dec. 2019, doi: [10.1109/TCYB.2018.2859801](https://doi.org/10.1109/TCYB.2018.2859801).
- [33] M. L. Lamali, N. Fergani, and J. Cohen, "Algorithmic and complexity aspects of path computation in multi-layer networks," *IEEE/ACM Trans. Netw.*, vol. 26, no. 6, pp. 2787–2800, Dec. 2018, doi: [10.1109/TNET.2018.2878103](https://doi.org/10.1109/TNET.2018.2878103).
- [34] M. Bianchini and F. Scarselli, "On the complexity of neural network classifiers: A comparison between shallow and deep architectures," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 25, no. 8, pp. 1553–1565, Aug. 2014, doi: [10.1109/TNNLS.2013.2293637](https://doi.org/10.1109/TNNLS.2013.2293637).



JUNPING HU was born in Shaoyang, China, in 1965. He received the B.S., M.S., and Ph.D. degrees in mechanical engineering from Central South University, Changsha, China, in 1986, 1989, and 1996, respectively. He is currently a Professor with Central South University. His current research interests include intelligent control theory and hydraulic technology.



GEN YANG was born in Zhangjiajie, China, in 1989. He received the B.S. degree in mechanical design, manufacturing, and automation from the China University of Geosciences, Wuhan, China, in 2013, and the M.S. degree in mechanical engineering from Central South University, Changsha, China, in 2016, where he is currently pursuing the Ph.D. degree in mechanical engineering. His research interests include adaptive dynamic programming and reinforcement learning.



ZHICHENG HOU (Member, IEEE) received the Ph.D. degree in control engineering from the Université de Technologie de Compiègne, Compiègne, France, in 2016. He is currently an Associate Professor with the Guangzhou Institute of Advanced Technology, Chinese Academy of Sciences, Guangzhou, China. His current research interests include nonlinear control and multirobot systems.



GONG ZHANG (Member, IEEE) was born in Xiaogan, China, in 1979. He received the Ph.D. degree in mechanical-electro-hydraulic hybrid driving science from Southwest Jiaotong University, Chengdu, China, in 2009. He has served as a Senior Engineer with Bosch Automotive Products (Changsha) Company Ltd., from 2009 to 2011. He joined the Guangzhou Institute of Advanced Technology, Chinese Academy of Sciences, in 2011, where he is currently an Associate Professor. His main research interests include multi-robot intelligent collaboration and human–robot collaborative control. He was granted the Jiangsu Double-Plan Talent, China, in 2016, and the Guangzhou High-Caliber Talent, China, in 2019.



WENLIN YANG (Member, IEEE) received the Ph.D. degree in machine-electronic from the Shenyang Institute of Automation, Chinese Academy of Sciences, Shenyang, China, in 2009. He is currently an Associate Professor with the Guangzhou Institute of Advanced Technology, Chinese Academy of Sciences, Guangzhou, China. His current research interests include new underwater robot for inspection, maintenance and repair, mobile, and operating robot.



WEIJUN WANG received the Ph.D. degree in mechanical engineering from Hanyang University, Seoul, South Korea, in 2012. He is currently a Professor and the Center Chief of the Guangzhou Institute of Advanced Technology, Chinese Academy of Sciences, Guangzhou, China, and the Chief Executive Officer of Shenzhen CAS Derui Intelligent Technology Company Ltd., Shenzhen, China. His research interests include intelligent robots, human–computer interaction, and mechanical design.

...