INTERNATIONAL JOURNAOL OF INTELLIGENT CONTROL AND SYSTEMS
VOL. 12, NO.3, SEPTEMBER 2007, 217-229

# Neural Network Predictive Control for Mobile Robot Using PSO with Controllable Random Exploration Velocity

Xin CHEN and Yangmin LI

*Abstract*- **Being a kind of intelligent method, Neural Network Predictive Control (NNPC) has been used widely to control nonlinear systems. However if traditional gradient decent algorithm (GDA) is employed to generate control signals, the computational cost is relatively too large, so that NNPC seems not to be acceptable for systems with rapid dynamics. To make NNPC be more efficient to general control signals for rapid dynamics, such as mobile robots, this paper proposes an improved optimization technique, particle swarm optimization with controllable random exploration velocity (PSO-CREV), to take place of GDA in NNPC. Moreover from theoretical analysis, it is observed that PSO-CREV has higher search ability than the conventional PSO. Therefore for one time of optimization, PSO-CREV needs small iterations than GDA, and less population size than conventional PSO. Hence the computational cost of NNPC is reduced by using PSO-CREV, therefore NNPC using PSO-CREV is more feasible for control of rapid processes. NNPC for trajectory tracking of mobile robots is chosen as a test to compare performance of PSO-CREV with other algorithms to show its advantages, especially in computational time.**

*Index Terms*—**Predictive control, PSO with controllable random exploration velocity (PSO-CREV), stochastic approximation, mobile robot, trajectory tracking**

## 1. INTRODUCTION

Since generalized predictive control (GPC) was developed in 1980s [1,2], it has been broadly used for industrial applications. As a kind of model-based predictive control (MBPC), GPC has been originally developed with linear plant predictor model, which leads to a formulation that can be solved analytically. But in many industrial processes which usually exhibit multivariable and nonlinear dynamical behavior, and such complicated systems may be not easily controlled by the linear GPC method, because the nonlinearity of plants affects GPC's computational efficiency and accuracy greatly. Of course for applications involving nonlinear systems, a kind of linear model [3] can be derived to apply GPC, but its bad performance makes GPC be unfeasible in these applications. Instead of linear model, neural network is used as an attractive tool to identify the complex nonlinear systems [3-5]. Thus a so called neural network predictive control (NNPC) is widely used in many nonlinear systems [6-8].

In most NNPC controllers, gradient decent algorithm (GDA) is viewed as the basic iterative method to derive control signals. Hence to predict the control signals in future, a well-known Jacobian matrix should be calculated, whose dimension is determined by the prediction horizon. If prediction horizon is chosen large enough, the computational cost induced by the complex Jacobian is so large that NNPC can not make response in time. Therefore it is difficult to employ NNPC for nonlinear systems with rapid dynamics.

To speed up the convergence of NNPC, some improvements have been developed. For example, a recursive form for Jacobian matrix computation is proposed in order to avoid computing partial differentiation elements [9]. At the same time some improvements on NN structure are developed to reduce complexity of NN structure [10].

From the viewpoint of optimization, the process of computing future control signals is also the process to optimize certain performance criterions. Therefore some evolutionary computational technologies, such as GA, can be employed to finger out proper control signals. However due to relative slow convergence of GA, using it to obtain control signals is not able to make NNPC be feasible in rapid dynamic processes. Instead of GA, we think that, due to fast convergence of particle swarm optimization (PSO), PSO is a good alternative to gradient decent algorithm. Thus the complex computation of Jacobian can be avoided.

Since the particle swarm optimization (PSO) was developed, owing to its simplicity and high performance, it has been proven to be a powerful competitor to other evolutionary algorithms [11,12], and been widely used in many optimization processes [13-16]. Because PSO is inspired by the studies of various animal groups, it simulates social behavior among individuals (particles) "flying" through a multidimensional search space, where each particle represents a point at the intersection of all search dimensions. Hence the main character of PSO is that there are some particles cooperating with each other to search optimal solutions. The so called fast convergence means that smaller iterations are needed than GA to obtain optimal solution. The computational cost of one time of fitness evaluation is determined by the population size, or the number of particles. To make PSO alternate gradient decent rule in NNPC, we need to improve PSO exploration ability in order that a swarm with relative small population size can find the optimal solution, while the accuracy of control is still maintained. For this purpose, we develop a PSO with controllable random exploration velocity (PSO-CREV), which has high exploration ability, while little additional computations are appended comparing with the conventional PSO updating principle. From the proof about convergence of this PSO-CREV, the high exploration ability is assured. To test its feasibility, the PSO-CREV based NNPC is applied to control a mobile robot on real time.

The rest of this paper is organized as follows. In Section 2, the conventional NNPC is introduced briefly and some

Manuscript received October 24, 2006; revised March 30, 2007. X. CHEN and Y. LI are with Dept. of Electromechanical Engineering, University of Macau, Av. Padre Tomas Pereira, Taipa, Macao SAR, China, (e-mail: ya27407@umac.mo and ymli@umac.mo ).

reasons resulting in slow convergence of NNPC are discussed. Section 3 presents the improved PSO-CREV algorithm, whose convergence is also proved. The algorithm about PSO-CREV based on NNPC is introduced in Section 4, with an application of NNPC using PSO-CREV for mobile robot control followed in Section 5. Based on the theoretical analysis and tests, the conclusions are drawn in Section 6.

## 2. Overview on Neural Network Predictive Control

### 2.1 Conventional Algorithm of NNPC

Generally speaking, given an unknown nonlinear system, its model can be expressed in the form of NARMA models with time delay, i.e.,

$$\begin{aligned} y(t+1) = f(&y(t-1), y(t-2), \cdots, y(t-n_y), \\ &u(t-d-1), \cdots, u(t-d-n_u)). \end{aligned} \tag{1}$$

where $f(\cdot)$ represents an unknown nonlinear system, $d$ represents the time delay of the systems, $n_y$ and $n_u$ are the orders of the system. Hence the purpose of the generalized predictive control (GPC) is to find out the control law $u(t)$ such that the system can converge to a predetermined reference signal. Just as mentioned, GPC is originally developed for linear system control, where the structure of the system is known as linear form, so that the control value can be directly derived from partial differentiation. However if the model is nonlinear and unknown, a model in the form of neural network for system identification is necessary to apply GPC technology. Thus the NNPC is developed recently.

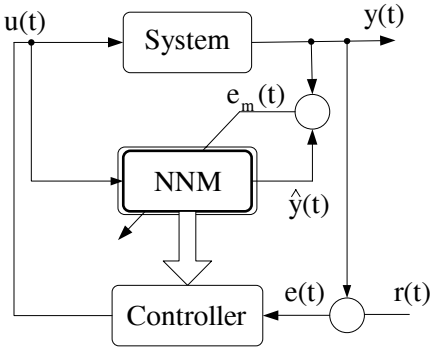



Fig. 1. The paradigm of neural network predictive control

Based on the main idea of NNPC, the general structure of NNPC is shown in Fig. 1. Obviously there are two aspects involved in NNPC: modeling plant and predictive controller.

**1. Plant model in form of neural network model**

For an unknown plant, many forms of neural network model (NNM) can be applied. Here we focus on the so-called multi-layer perceptron neural networks (MLP) for modeling. If $d$ is known, the input of the system is chosen as

$q = [y(t-1), y(t-2), \cdots, y(t-n_y), u(t-d-1), \cdots, u(t-d-n_u)]$ .

Hence a neural network designed to approximate the plant is expressed as

$$\begin{aligned} \hat{y}(t+1) = \hat{f}(q) = \hat{f}(&y(t-1), y(t-2), \cdots, \hat{y}(t-n_y), \\ &u(t-d), \cdots, u(t-d-n_u)) \\ = W[&tanh(Vq + b_1)] + b_2, \end{aligned} \tag{2}$$

where $q$ denotes the input of NNM, $W$ , $V$ , $b_1$ , and $b_2$ are the weights and biases matrices of the NN. Because the training of neural network is often operated off-line, any training algorithms in batch version can be developed to minimize the following criterion

$$J_M = \frac{1}{N_a} \sum_{t=1}^{N_a} (y(t) - \hat{y}(t))^2 \tag{3}$$

where $N_a$ represents the number of samples in training set.

**2. Predictive controller**

As a result, the well trained NN model can be chosen as the alternative to the real plant, so that not only the current control signals, but also the future control signals can be derived based on the plant model. To explain this purpose, we firstly define the following denotations. Let $T$ denote a specified predictive range, the reference point in future is represented by

$$R_t = [r(t+1), r(t+2), \cdots, r(t+T)]^T$$

and the predictive output of NNM is

$$\hat{Y}(t) = [\hat{y}(t+1), \hat{y}(t+2), \cdots, \hat{y}(t+T)]^T .$$

Therefore the error vector of predictive control is denoted by

$$E_t = [e(t+1), e(t+2), \cdots, e(t+T)]^T ,$$

where $e(t+i) = r(t+i) - \hat{y}(t+i)$ , $i = 1, \cdots, T$ . Based on the denotations, the purpose of NNPC is to derive a control vector of $U_t = [u(t-d+1), u(t-d+2), \cdots, u(t-d+T)]^T$ such that the following objective function is minimized.

$$J_C = \frac{1}{2}[E_t^T E_t]. \tag{4}$$

The conventional algorithm to derive control vector is based on gradient decent rule. That means the control signals are obtained using an iterative method, which is in the form of

$$U_t(k+1) = U_t(k) - \eta \frac{\partial J_C}{\partial U_t}(k), \tag{5}$$

Where

$$\frac{\partial J_C}{\partial U_t}(k)=\begin{bmatrix}\dfrac{\partial J_C}{\partial u(t+1)}\\ \vdots \\ \dfrac{\partial J_C}{\partial u(t+T)}\end{bmatrix}$$

$$=\begin{bmatrix}\dfrac{\partial \hat{y}(t+1)}{\partial u(t+1)} & 0 & \cdots & 0\\ \dfrac{\partial \hat{y}(t+2)}{\partial u(t+1)} & \dfrac{\partial \hat{y}(t+2)}{\partial u(t+2)} & \cdots & 0\\ \vdots & \vdots & \ddots & \vdots\\ \dfrac{\partial \hat{y}(t+T)}{\partial u(t+1)} & \dfrac{\partial \hat{y}(t+T)}{\partial u(t+2)} & \cdots & \dfrac{\partial \hat{y}(t+T)}{\partial u(t+T)}\end{bmatrix}^T.$$

This matrix is called the Jacobian matrix, which implies that NNPC requires a huge computational effort to generate control signals using the iterative algorithm in (5). To simplify computation, an improved iterative equation about the elements in Jacobian is proposed as follows:

$$\frac{\partial \hat{y}(t+n)}{\partial u(t+m-1)}=\frac{\partial \hat{y}(t+n-1)}{\partial u(t+m-1)}\left[1+\frac{\partial \hat{f}(X)}{\partial \hat{y}(t+n-1)}\right]$$

Unfortunately even using this simplified version, the computational cost of gradient decent rule is still huge with increasing of $T$.

### 2.2 Problems of NNPC in Real-Time Applications

Although NNPC is good at controlling unknown nonlinear systems, it does not mean that practical implementation is without difficulties. The primary shortage results from its computational cost. Since the nonlinear predictive controller has very comprehensive demands, for processes with rapid dynamics, such as mobile robots, this requirement will essentially prevent practical utilization of NNPC. Summarily the two important reasons resulting in huge computational cost are listed as follows:

- To avoid premature convergence, the learning rate $\eta$ should not be selected too large, hence the convergence of gradient decent rule is relatively slow;
- Due to Jacobian matrix, if it is required to derive predictive control signals in future, the dimension of Jacobian becomes too large to be computed rapidly.

## 3. NNPC USING PARTICLE SWARM OPTIMIZATION WITH CONTROLLABLE RANDOM EXPLORATION VELOCITY

### 3.1 Problems in NNPC Using Conventional PSO

To apply NNPC in the plants with rapid dynamics, the iterative algorithm should be speeded up by overcoming the shortages mentioned in Section 2.2. PSO algorithm can replace GDA to avoid computation of Jacobian and reduce iterations referred in (5).

Roughly speaking, if the control vector $U_t$ is viewed as a solution vector in a certain solution space, a group of particles can be chosen to represent potential solutions of $U_t$. Through interaction among particles, they converge to a position in solution space, which is the optimal control signal found by PSO. Since there is no GDA involved in PSO, the computation of Jacobian with huge computational cost can be avoided. At the same time, since PSO is a powerful evolutionary computation technique with fast convergence, it implies that using PSO may reduce the number of iterations referred in (5).

Compared with the advantages, there are some difficulties (or shortages) induced by conventional PSO. Consider the conventional PSO updating principle as

$$\begin{aligned}v_{id}(n+1)&=v_{id}(n)+c_1r_{1d}(n)[P_{id}^d(n)-X_{id}(n)]\\&\quad+c_2r_{2d}(n)[P_{id}^g(n)-X_{id}(n)],\end{aligned}\tag{6}$$

$$X_{id}(n+1)=X_{id}(n)+v_{id}(n+1),\tag{7}$$

where $d=1,\cdots,D$, $D$ is the dimension of solution space, $r_{1d}\sim U(0,1)$ and $r_{2d}\sim U(0,1)$ represent the two random numbers in the range of $[0,1]$; $c_{1i}$ and $c_{2i}$ represent the acceleration coefficients; $P_i^d(n)$ represents the best position found by particle $i$ so far, $P_i^g(n)$ represents the global best position found by particle $i$'s neighborhood.

Since $r_{1d}$ and $r_{2d}$ are added as the proportional coefficients to $P_{id}^d(n)-X_{id}(n)$ and $P_{id}^g(n)-X_{id}(n)$, the intensity of exploration behavior of PSO is entirely determined by the decreasing rate of $P_{id}^d(n)-X_{id}(n)$ and $P_{id}^g(n)-X_{id}(n)$, which can not be adjusted freely. Then if a swarm converges quickly to a certain position which may be not the global best solution, particles also give up attempts for exploration. This character brings a risk that if a swarm converges to a local best solution, particles can not use their poor exploration ability to escape from this suboptimal solution. So the accuracy of PSO-NNPC is dubitative.

To overcome this shortage, the only method for traditional PSO is to increase the number of particles to enhance exploration ability. But for application involving rapid dynamics, increasing population size also means increasing computational burden and reducing the feasibility of PSO-NNPC! Hence instead of adding particles, improving exploration ability of individual particles seems to be a direct way to improve accuracy of PSO-NNPC, while the computational cost is kept acceptable. For this purpose, we propose a new PSO algorithm, named PSO with controllable random exploration velocity (PSO-CREV), which has high exploration ability.

### 3.2 Definition of PSO-CREV

**Definition** The PSO-CREV is described as follows. Let $\mathsf{F}(n)$ be a sequence of sub-$\sigma$-algebra of $\mathsf{F}$ such that $\mathsf{F}(n) \subset \mathsf{F}(n+1)$, for all $n$. For a swarm including $M$ particles, the position of particle $i$ is defined as $X_i = [x_{i1} \quad x_{i2} \quad \cdots \quad x_{iD}]^T$, where $D$ represents the dimension of swarm space. The updating principle for individual particle is defined as

$$\begin{aligned} v_{id}(n+1) &= \varepsilon(n)\Big[ v_{id}(n) + c_1 r_{1id}(n)(P_{id}^d(n) - X_{id}(n)) \\ &\quad + c_2 r_{2id}(n)(P_{id}^g(n) - X_{id}(n)) + \xi_{id}(n)\Big], \\ X_{id}(n+1) &= \alpha X_{id}(n) + v_{id}(n+1) \\ &\quad + \frac{1-\alpha}{\phi_{id}(n)}(c_1 r_{1id}(n) P_{id}^d(n) + c_2 r_{2id}(n) P_{id}^g(n)), \end{aligned} \tag{8}$$

where $d = 1, \cdots, D$, $c_1$ and $c_2$ are positive constants; $r_{1id}(n)$ and $r_{2id}(n)$ are $\mathsf{F}(n)$-measurable random variables; $P_i^d(n)$ represents the best position that particle $i$ has found so far, which is of the form $P_i^d(n) = \arg\min_{k \le n} F(X_i(k))$, where $F(\cdot)$ represents a fitness function to be decreased; $P_i^g(n)$ represents the best position found by particle $i$'s neighborhood, which is of the form $P_i^g(n) = \arg\min_{j \in \Pi_i} F(X_j(n))$; $\phi_i(n) = \phi_{1i}(n) + \phi_{2i}(n)$, where $\phi_{1i}(n) = c_1 r_{1i}(n)$, $\phi_{2i}(n) = c_2 r_{2i}(n)$.

Suppose the following assumptions hold:

1) $\xi_i(n)$ is a bounded random variable with continuous uniform distribution. It has a constant expectation denoted by $\Xi_i = \mathbf{E}\xi_i(n)$;
2) $\varepsilon(n) \to 0$ with $n$ increasing, and $\sum_{n=1}^{\infty} \varepsilon(n) = \infty$;
3) $0 < \alpha < 1$;
4) $r_{1d}$ and $r_{2id}(n)$ are independent random variables satisfying continuous uniform distribution in [0,1], or $r_{1d} \sim U(0,1)$ and $r_{2d} \sim U(0,1)$. And let $\Phi_{1i} = \mathbf{E}\phi_{1i}(n)$ and $\Phi_{2i} = \mathbf{E}\phi_{2i}(n)$ respectively.

Then swarm must converge with probability one. Let $P^* = \inf_{\lambda \in (\mathbf{R}^D)} F(\lambda)$ represent the global optimal position in solution space. Then swarm must converge to $P^*$ if $\lim_{n\to\infty} P_i^d(n) \to P^*$ and $\lim_{n\to\infty} P_i^g(n) \to P^*$.

*Proof:* See the Appendix.

**Remark** Since $\phi_{1i}(n)$ and $\phi_{2i}(n)$ are chosen as random variables satisfying continuous uniform distribution and nonzero expectation, $\mathbf{E}\left[ b_3(n) \left\| Q^r(n) - E_n Q^r(n) \right\|^2 \right]$ must be decreasing, if $P^*$ is the global best solution. As $n$ becomes large enough such that $b_3(n) \to \Phi(1-\alpha)^2$, the invariant set is expressed as $\left\{ \theta \middle| k(\theta(t)) = \Phi(1-\alpha)\mathbf{E}\left\| Q^r(t) - \mathbf{E}_n Q^r(t) \right\|^2 \right\}$. Obviously the only way to make this set become $\{\theta \mid k(\theta(t)) = 0\}$ is to make $Q^r(n) \to 0$ as $n \to \infty$, or $P^d(n)$ and $P^g(n)$ approach $P^*$. The proof does not assert that $\lim_{n\to\infty} Q^r(n) = 0$, because in the updating principle, there is no explicit information about the global best solution $P^*$. According to the following expressions, $P_i^d(n) = arg\min_{k \le n} F(X_i(k))$ and $P_i^g(n) = arg\min_{j \in \Pi_i} F(X_j(n))$, since $P_i^g(n)$ represents the result of exchanging $P_i^d(n)$ among particles, the key factor to make $P_i^d(n)$ approach the global best solution $P^*$ is to make particles' trajectories be closed to $P^*$, or to enhance the opportunity of exploring the vicinity around $P^*$. That is why the random velocity $\xi(n)$ is introduced to PSO-CREV to entitle particles to reach unknown solution space, which may be close to $P^*$.

### 3.3 Properties Caused by Stochastic Behavior

Based upon the proof presented in the previous section, some properties that enhance particles' exploration ability can be figured out. These properties can explain why PSO-CREV is more efficient to search optimal solution with small population size, so that it can be applied in NNPC for rapid dynamics.

#### *3.3.1 Property 1: Divergence ahead of Convergence*

In the proof, it is mentioned that before $n$ is large enough to make (A.9) hold, the individual updating principle is nonconvergent so that particle will move away from the best position recorded by itself and its neighborhood. But during this divergent process, particles are still recording their individual best solutions and exchanging information with each other. Hence this phenomenon can be viewed as a strong exploration that all particles wander in the solution space and record the best solutions found so far. And when $n > N_k$, particles start to converge to the best solutions found so far.

In a sense, this temporary divergence can be viewed as a kind of inherent exploration ability. But this behavior is uncontrollable, and the intensity of stochastic behavior still depends on cognitive and social components, hence the basic drawback lies in that the uncontrollable convergence rate is not improved by this property.

#### *3.3.2 Property 2: Additional Random Search Velocity*

The random velocity $\xi(n)$ is the key improvement of PSO-CREV. Obviously without the additional stochastic behavior, or $\xi(n) = 0$, PSO-CREV behaves much like the traditional PSO with relatively fast convergence. Since the global optimal solution is unknown, the way to make particles converge to $P^*$ is improving opportunities to

approach (explore) the vicinity of $P^*$. Hence a nonzero $\xi(n)$ is very useful to drive particles into unknown solution space besides the effect brought by cognitive and social components.

Moreover it is easy to prove that PSO-CREV with zero $\mathbf{E}\xi(n)$ converges more quickly than the one with nonzero $\mathbf{E}\xi(n)$. Hence in applications, $\xi(n)$ with zero expectation is more preferable than nonzero one.

*3.3.3 Property 3: Adjustable Bound of Random Search Velocity*

Of course nonzero $\xi(n)$ enhances the exploration ability of PSO-CREV. But at the same time its property of independence on social and cognitive components brings a drawback that, when particles have approached the optimal solution, they can hardly converge quickly caused by nonzero $\xi(n)$. Therefore although $\xi(n)$ enhances particle's exploration ability, $\xi(n)$ may delay the convergence of particles at the same time.

Fortunately from the assumption 2) of the definition of PSO-CREV, it is noted that the only constraint of $\xi(n)$ should be bounded. Hence to control convergence rate of PSO-CREV (to speed up convergence indeed), a time-varying $\xi(n) = w(n)\bar{\xi}(n)$ is proposed, where $\bar{\xi}(n)$ represents a stochastic velocity with zero expectant and constant value range, $w(n)$ represents a time-varying positive coefficient. By this varying $w(n)$, the bound of the random search velocity can be adjusted according to the different requirements of optimization problems. Normally we expect that during the iterations shortly after beginning, the random search velocity should be strong, so that particles will be more active to search unknown space in solution space. Hence the bound of random velocity should be large enough. On the contrary, at the end period of iterations this random search velocity should be suppressed, so that it will not affect the convergence of particles. Hence during this period, the bound should be small and small. Considering both aspects, the dynamic strategy of $w(n)$ in this paper is defined as

$$w(n) = \begin{cases} 1, & n < \frac{1}{4}N_b; \\ \lambda_1 w(n-1), & n \geq \frac{1}{4}N_b, n < \frac{3}{4}N_b; \\ \lambda_2 w(n-1), & n \geq \frac{3}{4}N_b, \end{cases} \quad (9)$$

where $N_b$ represents the total number of iterations, and $\lambda_1$ and $\lambda_2$ are positive constants less than 1, which make $\xi(n)$ be a decreasing random velocity. For example, if the total iterations for one time optimization is set to 200, we choose $\lambda_1 = 0.99$ and $\lambda_2 = 0.8$. Hence during iteration 1 to 50, the intension of $\xi(n)$ is strong, so that particles have more opportunities to reach unknown solution space. And after that, the bound of $\xi(n)$ decreases iteration by iteration on the different rates within different periods, especially in the last quarter of iterations, $\xi(n)$ has trivial effect on the convergence of PSO-CREV finally. In one word, such a time-varying bound of random search velocity makes PSO-CREV meet the both requirements of strong exploration ability and fast convergence.

## 4 THE ALGORITHM OF NNPC USING PSO-CREV

Comparing the introduction of NNPC in Section 2 with PSO-CREV in Section 3, we know that the control vector $U_t$ at time $t$ should be chosen as solution vector in PSO-CREV, which is denoted by $X$ in the definition of PSO-CREV. And the fitness function that should be optimized by PSO-CREV is of the form in (4). Consequently the algorithm of NNPC using PSO-CREV is summarized as follows.

1) Select $T$.
2) Initialize PSO-CREV, where the dimension of solution space equals the predictive horizon $T$ multiplying with the dimension of control signal.
3) At the time $t$, initialize particles' positions. Each particle represents a potential predictive control signal, i.e. $X_i = [u(t-d+1), \cdots, u(t-d+T)]$.
4) Use PSO-CREV to optimize predictive control signal. To evaluate fitness, the input of NNM consists of three parts, the potential control signals, the real output of plant, and the predictive output of NNM. That means for $1 \leq j \leq T$, if $j < n_y$,
$$q_j = [\hat{y}(t+j-1) \quad \hat{y}(t+j-2) \quad \cdots \quad \hat{y}(t+1) \quad y(t) \quad \cdots \quad y(t+j-n_y) \quad u(t+j-d) \quad \cdots \quad u(t+j-d-n_u)].$$
5) In case of PSO-CREV converges, apply $u(t-d+1)$ found by PSO-CREV to close the control loop.
6) Return to 3).

## 5 APPLICATION ON MOBILE ROBOT CONTROL

In this section, we propose a simulation where NNPC using PSO-CREV algorithm is applied to control a mobile robot with nonholonomic constraints.

### 5.1 Real Plant and the Structure of NNM

**1. Discrete-Time Model of Mobile Robot**

A car-like mobile robot is shown in Fig. 2 in the simulation, which has two driving wheels and a caster wheel. In this model, the coordinate of the robot is denoted by $p = [x \quad y \quad \theta]^T$, where $[x, y]$ represents the center point of the two driving wheels, $\theta$ represents the heading angle of the robots. Consequently the kinematic model of

the mobile robot in the universal frame is expressed as following discrete-time form.

$$\begin{bmatrix} x(t+1) \\ y(t+1) \\ \theta(t+1) \end{bmatrix} = \begin{bmatrix} x(t) \\ y(t) \\ \theta(t) \end{bmatrix} + \begin{bmatrix} u_v \cos(\theta + \frac{u_\omega}{2}) \\ u_v \sin(\theta + \frac{u_\omega}{2}) \\ u_\omega \end{bmatrix} \quad (10)$$

where $u_v$ denotes the translational velocity of the position $[x, y]$, and $u_\omega$ denotes the rotational velocity of the mobile robot.

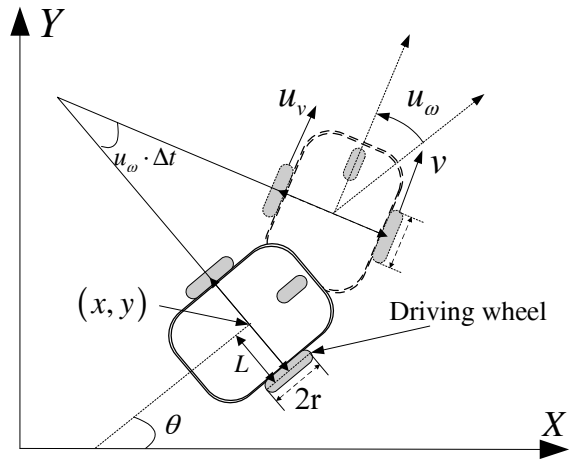



Fig. 2. Kinematics of a mobile robot.

## 2. NNM for Mobile Robot

As mentioned in Section 2, a two-layer feed forward neural network is employed to model mobile robot. To improve the accuracy of the NNM, we choose three NNs to model three coordinates respectively. Assume that the time delay of the system $d$ equals 0, but the order of mobile robot is unknown. Then the NNM can be expressed as

$$\hat{y}_j(t) = \hat{f}(q) = \hat{f}(y(t-1), u(t), u(t-1)),$$

where $y(t) = [y_x(t), y_y(t), y_\theta(t)]^T$, the subscript $j$ represents the three general coordinates $x$, $y$, and $\theta$ respectively. Therefore the dimension of input to NNM equals 7 (two control signals with dimension two and one output history with dimension three). And the structure of single NNM is chosen as $7-5-1$ with biases in hidden and output layers, which is shown in Fig. 3.

### 5.2 NNPC for Mobile Robot

When NNM is trained to approximate a plant, there are two kinds of feedback methods which send different signals to NN for training. The first method is called the parallel connection for system identification, where the output of NNM is fed back to NN training, and the second method is called the series-parallel connection for system identification, where the real output of the plant is sent to NNM for training [18]. Because the series-parallel configuration is simple and easy to be realized, in this paper NNM is trained by series-parallel way.

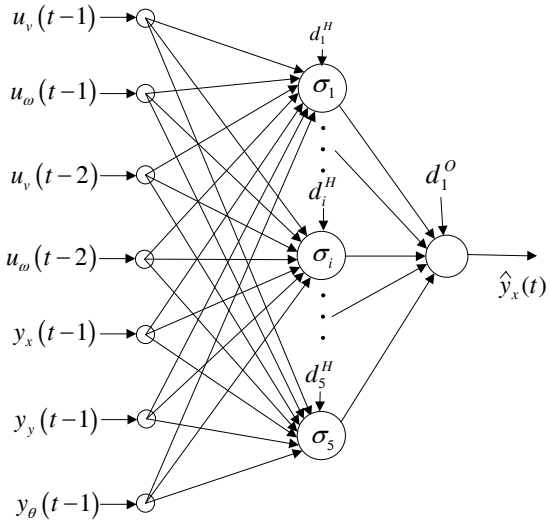



Fig. 3. The neural network model for X-dimension.

Hence according to the general NNPC diagram shown in Fig. 1, the structure diagram of the mobile robot control is shown in Fig. 4.

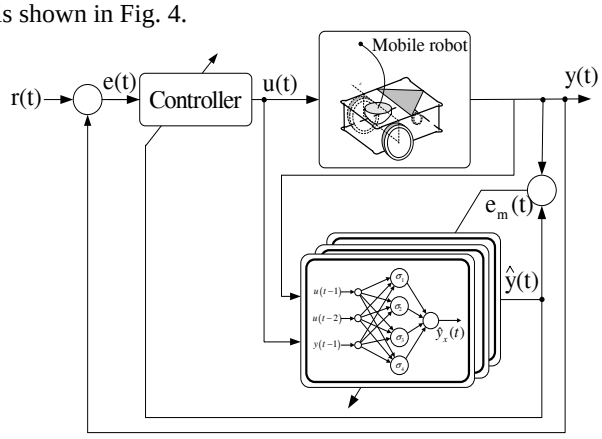



Fig. 4. The diagram of NNPC using PSO-CREV for mobile robot.

### 5.3 Test Setup

From the introduction of NNPC, it is known that there are two parts involving NN training.

1. **System Identification**

The first part is system identification using series-parallel method, which is normally trained off-line. Here the gradient descent with momentum and adaptive learning rate BP is chosen as the training algorithm to obtain the NNM of mobile robot. All parameters involved in the training are chosen as: the initial learning rate is set to 0.02

the ratio to increase learning rate is 1.05; the ratio to decrease learning rate is 0.7; the momentum constant is set to 0.9; the batch mode of training with 500,000 epochs is adopted. Because the main issue in the paper is to investigate the application of NNPC using PSO-CREV in rapid dynamic systems, training of NNM is less concerned. Hence only the setup for NNM training is introduced here, and it is noted that using this BP training method, the NNM can approximate the structure of real mobile robot very well.

2. **NNPC**

The second part involving NN training is NNPC on line, which is what we concern in the paper. A well trained NNM is employed to generate predictive control signals. Besides PSO-CREV, three other algorithms are selected as comparison, so that we can compare them in two aspects with the accuracy of NNPCs and computational cost.

In the test, a mobile robot is required to follow two kinds of trajectories, a circle one and a sine one. In the test of cycle trajectory, the linear velocity and steering velocity driving a robot should be constant under desired circumstances. Hence the test referring circular trajectory can test whether the outputs of NNPCs using different learning algorithms are stable or not. Obviously given a learning algorithm, if control signals generated by NNPC using this learning algorithm are stable (in other words the control signals are less fluctuant), it means that this learning algorithm is with high ability to find the desired control signals. On the other hand, the test referring sine trajectory will test the performance of NNPCs when the desired control signals are time-varying. The parameters involved in the two tests are shown as follows:

- **Desired cycle trajectory**: The desired cycle trajectory results from the movement of a virtual mobile robot with the constant velocities $[0.1m/s \quad 0.1\pi rad/s]$ in the interval $[0s \quad 20s]$.
- **Desired sine trajectory**: The desired sine trajectory results from the movement of a virtual mobile robot with the following velocities in interval $[0s \quad 21s]$.

  $u_v(t) = 0.2m/s;$

  $u_\omega(t) = -2.5^2 \sin(5x^d(t-1)) \cdot u_v \cos(\theta^d(t-1))^3 rad/s,$

  where $p^d(t) = [x^d(t) \quad y^d(t) \quad \theta^d(t)]^T$ represents the coordinate of the virtual robot at time $t$.

And it is assumed that the sample interval is $0.1s$. That means NNPC outputs control signals on every $0.1s$. In both tests, the predictive horizon is set to 2, i.e., $T = 2$.

Four algorithms involved in the test of NNPC are PSO-CREV, the conventional gradient decent algorithm (GDA), PSO with linear decreasing inertia weight (PSO-LDIW) [19], and GA. The configurations of these algorithms are introduced as follows:

- **PSO-CREV:** The updating principle is of the form (8) with a decreasing $\xi(n)$ whose $w(n)$ is in the form of (9). All parameters are chosen as: $c_1 = c_2 = 3.5$, $\alpha = 0.95$, $a = 3$, $b = 0.35$, $\lambda_1 = 0.99$, and $\lambda_2 = 0.8$. The *lbest* version [20] of neighborhood is used where each particle exchanges information with four other particles. Due to high exploration ability of PSO-CREV, there are only 5 particles enclosed in PSO-CREV that is even less than the dimension of input of NNM (the dimension of each NN is 7).
- **PSO-LDIW:** A conventional PSO with time-varying inertia weights is chosen, whose parameters are chosen as: $c_1 = c_2 = 2$. The inertial weight is set to change from 0.9 to 0.4 over the iterations. The *lbest* PSO is used to realize social component. In order to obtain acceptable results, two kinds of population sizes with 10 and 20 particles are used in the conventional PSO paradigm.
- **GA:** A standard GA algorithm with selection, crossover, and mutation operations is employed to optimize control signals. The crossover probability is set to 0.8, while the mutation probability is chosen as 0.1.
- **Conventional GDA:** The details of iterative algorithm of GDA can be found in [9]. Here $\eta$ is set to 0.03.

### 5.4 Simulation Results of NNPCs Using Different Learning Algorithms

Because PSOs algorithms converge very quickly, and the major concern of this paper is to investigate the efficiency of NNPC learning, the optimization processes of PSO-CREV and PSO-LDIW include only 200 iterations of fitness evaluation at each sample time. And the maximal iteration for GDA algorithm is set to 2,000 iterations, if it can not make tracking error, $E_t$, be below 0.001 within 2,000 iterations. The maximal generation of GA is also set to 2,000. But since in fact GA converges so slowly that there is seldom opportunity to evolve acceptable control signal, the performance of GA learning will be ignored. All tests will be run for 20 times, and the runs with the best performance with respect to each algorithm are picked up as the results shown in the following figures.

#### *5.4.1 Tacking Circular Trajectory*

At the beginning, the mobile robot is placed on the original position $[-0.15 \quad -0.05]$ with head angle $\frac{\pi}{8} rad$. Then the robot is controlled by four different algorithms respectively to follow a circular trajectory.

The traces resulting from NNPCs based on all four algorithms (here two PSO-LDIWs with two population sizes are employed, and the results of GA are ignored) are shown in Fig. 5 with the desired trajectory as comparison. And the tracking errors over sample times are shown in Fig. 6 (a). Here the tracking error is defined as $E = \left\| p^d - p \right\|$, where $p^d = [x^d \quad y^d \quad \theta^d]^T$, $p = [x_c \quad y_c \quad \theta]^T$. Finally the control signals generated by NNPCs over time are shown in Fig. 6 (b)

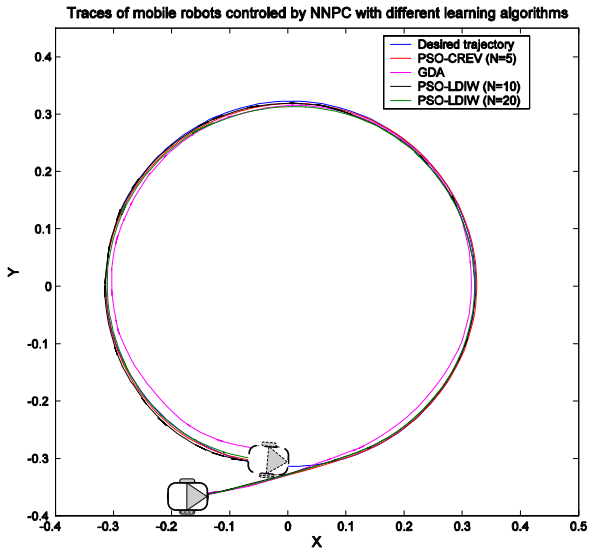


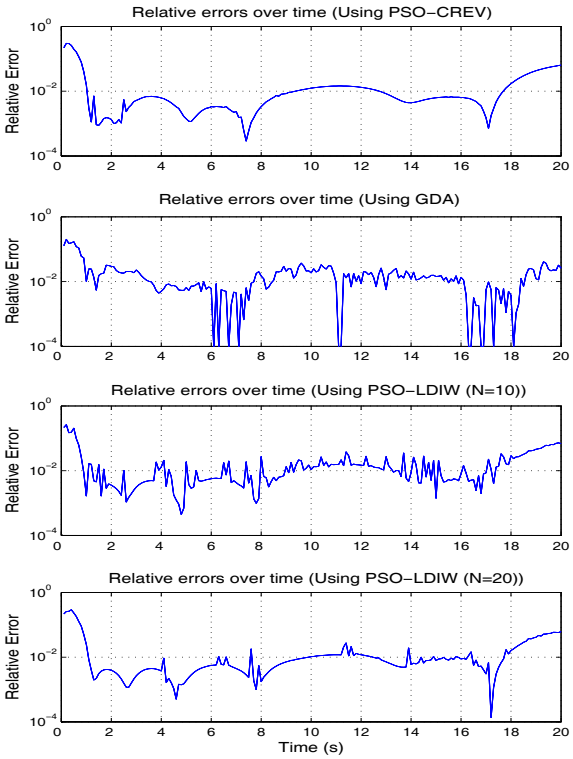

Fig. 5. All traces of a robot in tracking circular trajectory.

From the traces and relative errors over time, it is observed that for tracking circular trajectory, there are not too many differences between the performances of all NNPCs using four learning algorithms. All traces of mobile robots are close enough to the desired trajectory. And the curves of relative errors are somewhat similar. Comparing all curves, it is observed that the shapes of all curves have similar trends, especially at the final period, the tracking errors of all algorithms increase slightly. That means such increasing errors must be induced by the approximation error of NNM. Hence only from the traces it seems that PSO-CREV learning algorithm is not superior to other algorithms.

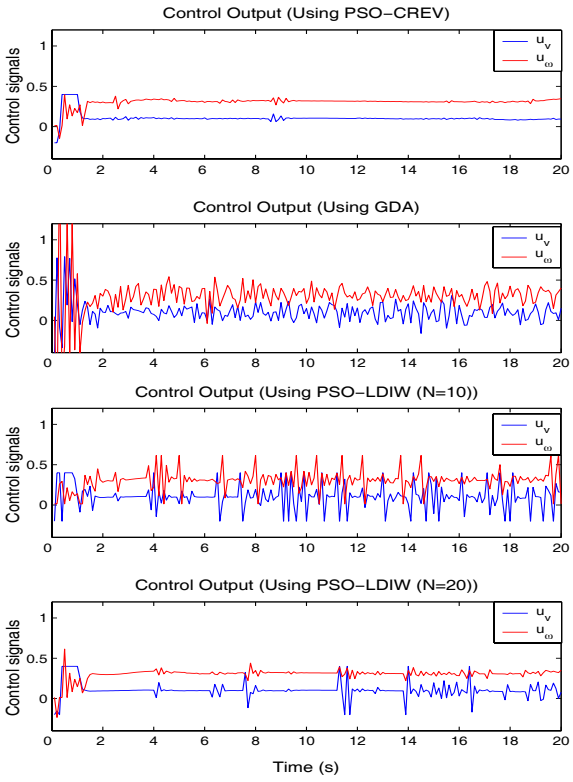
However from Fig. 6 (b), it is clearly observed that comparing with other algorithms, the curves of control signals generated by NNPC using PSO-CREV are smooth with less fluctuations. That means at every sample time, PSO-CREV converges to the best solution very well. Therefore the control signals are almost the same at every sample time. As a comparison, NNPC using GDA seems hardly to derive the constant control signals within 2,000 iterations. Although the total iterations for NNPC using PSO-LDIW is set to 200, that is the same as PSO-CREV, the curves with respect to NNPCs using PSO-LDIW also show some sudden changes of control signals. Hence from the Fig. 6 (b) it is concluded that PSO-CREV is more efficient to find the best solution within only 200 iterations.

### *5.4.2 Tracking Sine Trajectory*

This test is proposed to test performance of all algorithms under the circumstance of desired time-varying control signals. At the beginning, the mobile robot is placed on the original position $[0 \quad -0.15]^T$ with head angle $atan(2.5)rad$ .




(a) The relative errors over time.




(b) The control signals over time.

Fig. 6. The relative errors and control signals of tracking circular trajectory.

Similarly the traces resulting from NNPCs are shown in Fig. 7. The tracking errors over sample times are shown in Fig. 8 (a). And the control signals generated by NNPCs are shown in Fig. 8 (b).

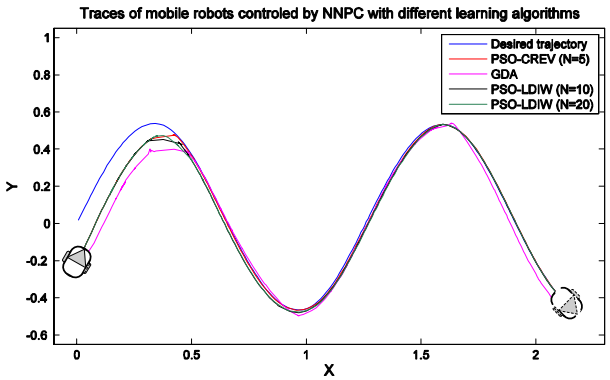


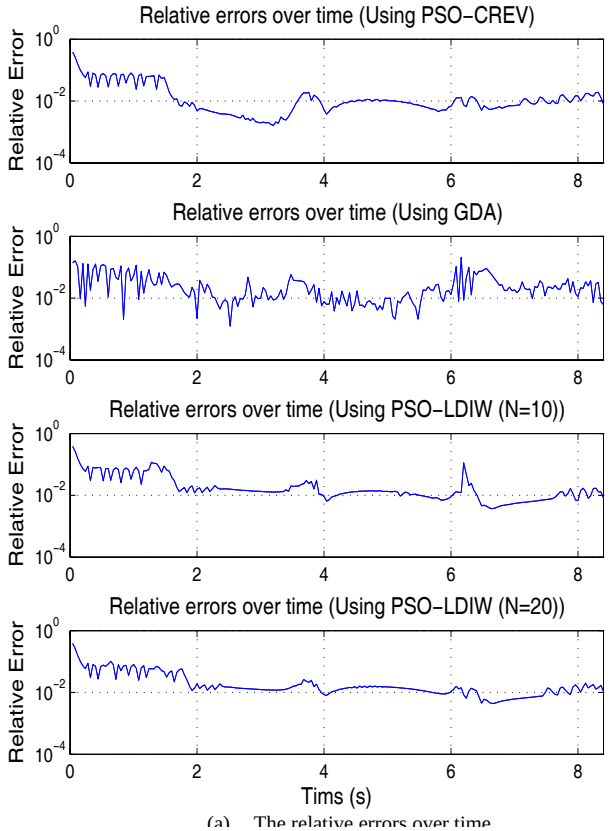

Fig. 7. All traces of a robot in tracking sine trajectory.

Undoubtedly four NNPCs are all able to make mobile robot follow the desired sine trajectory. And Fig. 8 (a) implies that comparing with other algorithms, PSO-CREV can generate more optimal control signals to make relative error be smaller than other algorithms, although such a "higher" performance is not too much impressive. Again from the Fig. 8 (b) showing control signals, the output of NNPC using PSO-CREV looks smoother than other algorithms, except comparing with PSO-LDIW with 20 particles. But it is noted that the population size of the later is as four times as the former. That means to get the similar performance, PSO-LDIW needs more computations, and results in longer computational time. Hence it is verified again that if the number of iterations is set to only 200, PSO-CREV is more effective to find optimal control signals within such a short of period.

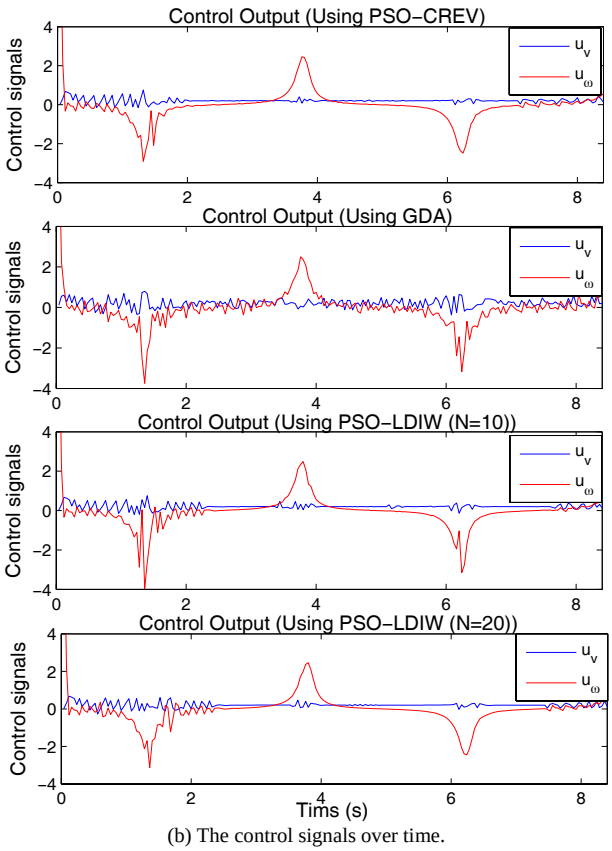
It should be noted when population size of PSO-LDIW is set to 5, NNPC using PSO-LDIW always prematurely converges within 200 iterations. Hence NNPC using PSO-LDIW with small population size (for example 5 particles) can hardly generate proper control signals to realize trajectory tracking. To show this shortage of PSO-LDIW with small population size, a test is proposed, where the sine trajectory tracking is repeated 50 times by using three kinds of PSO algorithms, including PSO-CREV with 5 particles, and PSO-LDIW with 5 and 20 particles respectively. Since the sample interval is set to 0.1s, there are 210 sample times in one run of tracking process. Then the root mean square error of one run is chosen as

$$RMSE = \left\{ \frac{1}{210} \sum_{k=1}^{210} \left[ \left( y_x^d(k) - y_x(k) \right)^2 + \left( y_y^d(k) - y_y(k) \right)^2 \right] \right\}^{1/2},$$

where $\left[ y_x^d(k) \quad y_y^d(k) \right]$ represents the reference point on the desired sine trajectory at time $k$, and $\left[ y_x(k) \quad y_y(k) \right]$ is the actual position of robot.




(a) The relative errors over time




(b) The control signals over time.

Fig. 8. The simulation results of the test referring sine trajectory.

All RMSEs with respect to three algorithms over 50 runs are depicted in Fig. 9. Clearly the performance of PSO-LDIW with 5 particles is so poor that there are few runs whose RMSEs are less than 0.06. On the contrary, PSO-CREV and PSO-LDIW perform much better. It is observed that there is one run of PSO-LDIW failing to track trajectory. Hence comparing with PSO-LDIW, PSO-CREV undoubtedly has higher exploration ability, so that PSO-CREV with much smaller population size can perform as good as, even better than, PSO-LDIW.

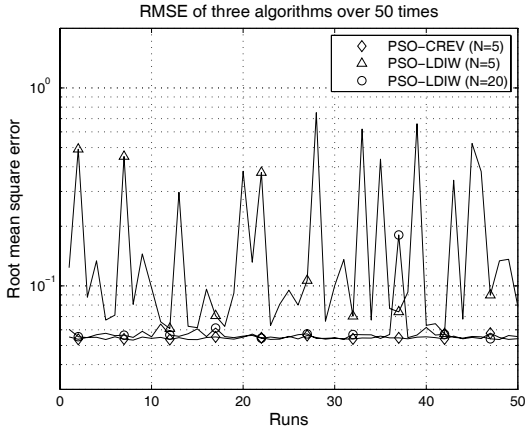



Fig. 9. Comparison of tracking performance of NNPC using PSO-CREV and PSO-LDIW with different population sizes.

### *5.4.3 Comparison of Computational Cost*

Now let us check the computational cost of all NNPCs. Since at each sample time, all algorithms need to evaluate the fitness, i.e. the predictive tracking errors, it is easy to estimate the numbers of computing outputs of NNM at each iteration. If let computational time of PSO-CREV algorithm be one unit, the numbers of fitness evaluation and computational times for one optimization process with respect to all algorithms are listed in Table 1, which result from the average of all 20 runs in the test referring sine trajectory.

Table 1: Comparison of computational cost of NNPCs using different learning algorithms.

| Algorithm | Number of fitness evaluation | Relative computational time |
|---|---|---|
| PSO-CREV (N=5) | $5\times 200 = 1000$ | 1 |
| GDA | 2000 | 10.2654 |
| PSO-LDIW (N=10) | $10\times 200 = 2000$ | 1.9161 |
| PSO-LDIW (N=20) | $20\times 200 = 4000$ | 4.4596 |
| GA | $30\times 3000 = 90000$ | 13.3430 |

Obviously computational time of PSO-CREV is far less than other algorithms, especially only ten percent of the time consumed by GDA. Hence we can conclude that PSO-CREV algorithm can enhance optimization efficiency of NN training on line, therefore it is more feasible to be employed in NNPC than traditional NNPC using GDA in the applications involving rapid dynamics.

## 6. CONCLUSIONS

This paper proposes a useful optimization method, PSO-CREV, to enhance computational efficiency of NNPC, so that NNPC can be used in the systems with rapid dynamics. As a rapid convergence algorithm, PSO looks like a good alternative to GDA for searching optimal solution. But a swarm with too many particles also brings too much computational cost. To overcome this drawback, we develop an improved stochastic PSO which has high exploration ability, so that PSO-CREV with small population size has powerful enough search ability, which is often observed in conventional PSO with large population size. The tests show that PSO-CREV converges very quickly, while the computational time consumed by it is far less than conventional PSO and GDA. Hence this PSO-CREV enhances the efficiency of NNPC very much. And NNPC based on PSO-CREV becomes feasible for rapid dynamic processes.

The improved PSO has higher performance than conventional PSO, such as DEPSO [21], MPSO [22], CPSO [23], etc.. But these improvements have the same characteristics that they employ additional operations with the basic updating principle (6) and (7) intact. Hence it is inevitable that these improvements induce increase of computational time, which contradicts the requirements of reducing computational cost. Comparing with them, PSO-CREV is undoubtedly a better alternative to conventional PSO.

## 7. APPENDIX

### Proof of the convergence of PSO-CREV

Because we are interested in the behavior that particles approach the global best solution, a new relative position is defined as $Y(n) = X(n) - P^*$. Substituting it into (8) yields

$$\begin{aligned} v(n+1) &= v(n) + \big[-(1-\varepsilon(n))v(n) - \varepsilon(n)\phi(n)Y(n) \\ &\quad + \varepsilon(n)(\phi_1(n)Q^d(n) + \phi_2(n)Q^g(n)) + \varepsilon(n)\xi(n)\big], \\ Y(n+1) &= Y(n) + (\alpha-1)Y(n) + v(n+1) \\ &\quad + \frac{1-\alpha}{\phi(n)}(\phi_1(n)Q^d(n) + \phi_2(n)Q^g(n)) \\ &= Y(n) + \Big[-(1-\alpha+\varepsilon(n)\phi)Y(n) + \varepsilon(n)v(n) \\ &\quad + \frac{1-\alpha+\varepsilon(n)\phi(n)}{\phi(n)}(\phi_1(n)Q^d(n) + \phi_2(n)Q^g(n))\Big], \end{aligned} \tag{A.1}$$

where $Q^d(n) = P^d(n) - P^*$, $Q^g(n) = P^g(n) - P^*$.

Let $Q^r(n) = \frac{1}{\phi(n)}[\phi_1(n)Q^d(n) + \phi_2(n)Q^g(n)]$, and $\theta(n) = [v(n) \quad Z(n)]^T = [v(n) \quad Y(n) - \mathbf{E}_n Q^r(n)]^T$, where $\mathbf{E}_n$ denotes the expectation conditioned on the $\sigma$-algebra $\mathsf{F}_n$. Then it follows that

$$\theta(n+1) = \theta(n) + \varepsilon(n)H(n), \tag{A.2}$$

where $H(n)=\frac{1}{\varepsilon(n)}\begin{bmatrix}h_1 & h_2\end{bmatrix}^T$, where

$$h_1=-(1-\varepsilon(n))v(n)-\varepsilon(n)\phi(n)Z(n)+\varepsilon(n)\phi(n)[Q^r(n)-\mathbf{E}_nQ^r(n)]+\varepsilon(n)\xi(n);$$

$$h_2=-(1-\alpha+\varepsilon(n)\phi(n))Z(n)+\varepsilon(n)v(n)+(1-\alpha+\varepsilon(n)\phi(n))\cdot[Q^r(n)-\mathbf{E}_nQ^r(n)].$$

A Lyapunov function is defined as

$$L(\theta(n))=\frac{1}{2}\theta^T\begin{bmatrix}1 & 0\\ 0 & \Phi\end{bmatrix}\theta=\frac{1}{2}(v^2(n)+\Phi Z^2(n)),\quad (A.3)$$

where $\Phi=\mathbf{E}(\phi)$.

**(1) Properties on the derivative of Lyapunov function**

If $\mathbf{E}|H(n)|<\infty$, let $\gamma(n)=\mathbf{E}_nH(n)$. Then we have

$$\gamma(n)=\frac{1}{\varepsilon(n)}\begin{bmatrix}-(1-\varepsilon(n))v(n)-\varepsilon(n)\Phi Z(n)+\varepsilon(n)\Xi\\ -(1-\alpha+\varepsilon(n)\Phi)Z(n)+\varepsilon(n)v(n)\end{bmatrix}.\quad (A.4)$$

Using a truncated Taylor series expansion, we have

$$\mathbf{E}_nL(\theta(n+1))-L(\theta(n))=\varepsilon(n)L'_\theta(\theta(n))\gamma(n)+\mathbf{E}_n(\varepsilon^2(n)H(n)^T\begin{bmatrix}1 & 0\\ 0 & \Phi\end{bmatrix}H(n)).\quad (A.5)$$

Calculating the first term on the right side yields

$$\begin{aligned}&L_\theta{}^T(\theta(n)\gamma(n)\\&=\frac{1}{\varepsilon(n)}\mathbf{E}_n\{-v^T(n)[(1-\varepsilon(n))v(n)+\varepsilon(n)\Phi Z(n)-\varepsilon(n)\Xi]\\&\quad-\Phi Z^T(n)[(1-\alpha+\varepsilon(n)\Phi)Z(n)-\varepsilon(n)v(n)]\}\\&=-\frac{1}{\varepsilon(n)}[(1-\varepsilon(n))\|v(n)\|^2+\Phi(1-\alpha+\varepsilon(n)\Phi)\|Z(n)\|^2\\&\quad-\Xi v^T(n)\varepsilon(n)].\end{aligned}\quad (A.6)$$

Let $(\tilde{H}(n))^2$ denote $H(n)^T\begin{bmatrix}1 & 0\\ 0 & \Phi\end{bmatrix}H(n)$. After some calculations, we have

$$\begin{aligned}\mathbf{E}\left[\|\tilde{H}(n)\|\,|\,\mathsf{F}(n)\right]^2=\frac{1}{\varepsilon^2(n)}\mathbf{E}_n\{&a_1(n)\|v(n)\|^2+a_2(n)\|Z(n)\|^2\\&+a_3(n)\|Q^r(n)-\mathbf{E}_nQ^r(n)\|^2\\&+a_4(n)v^T(n)Z(n)\\&+a_5(n)[Q^r(n)-\mathbf{E}_nQ^r(n)]\\&+a_6(n)\xi(n)+(\varepsilon(n)\xi(n))^2\},\end{aligned}\quad (A.7)$$

where

$$a_1(n)=(1-\varepsilon(n))^2+\Phi\varepsilon^2(n);$$

$$a_2(n)=\Phi(1-\alpha+\varepsilon(n)\phi(n))^2+(\varepsilon(n)\phi(n))^2;$$

$$a_3(n)=\Phi(1-\alpha+\varepsilon(n)\phi(n))^2+(\varepsilon(n)\phi(n))^2;$$

$$a_4(n)=2\varepsilon(n)[\phi(n)(1-\varepsilon(n))-\Phi(1-\alpha+\varepsilon(n)\phi(n))];$$

$$\begin{aligned}a_5(n)=&2[\Phi\varepsilon(n)(1-\alpha+\varepsilon(n)\phi(n))\\&-\varepsilon(n)\phi(n)(1-\varepsilon(n))]v^T(n)\\&-2[\Phi(1-\alpha+\varepsilon(n)\phi(n))^2+\varepsilon^2(n)\phi^2(n)]Z^T(n)\\&-\varepsilon(n)\phi(n)\xi^T(n);\end{aligned}$$

$$a_6(n)=-2\varepsilon(n)\{(1-\varepsilon(n))v^T(n)+\varepsilon(n)\phi(n)Z^T(n)\}.$$

Substituting (A.6) and (A.7) into (A.5), and using inequality $\|uv\|\le\frac{1}{2}(\|u\|^2+\|v\|^2)$, we obtain equation array

$$\begin{aligned}\mathbf{E}_nL(\theta(n+1))-L(\theta(n))\le{}&b_1(n)\|v(n)\|^2+b_2(n)\|Z(n)\|^2\\&+\mathbf{E}_n\left[b_3(n)\|Q^r(n)-\mathbf{E}_nQ^r(n)\|^2\right]\\&+\mathbf{E}_n\left[b_4(n)(Q^r(n)-\mathbf{E}_nQ^r(n))\right]\\&+b_5(n)\mathbf{E}_n\|\xi(n)\|^2+\tfrac{1}{2}\sqrt{\varepsilon(n)}\|\Xi\|^2,\end{aligned}\quad (A.8)$$

where

$$b_1(n)=-\varepsilon(n)+(1+\Phi)\varepsilon^2(n)+(\tfrac{5}{2}+\varepsilon^2(n)-2\varepsilon(n))\varepsilon^{\frac{3}{2}}(n);$$

$$\begin{aligned}b_2(n)=&-\Phi\alpha(1-\alpha)-\varepsilon(n)\Phi^2(2\alpha-1)+\varepsilon^2(\Phi+2)\mathbf{E}_n(\phi^2(n))\\&+\sqrt{\varepsilon(n)}\mathbf{E}_n[\phi(n)(1-\varepsilon(n))-\Phi(1-\alpha+\varepsilon(n))]^2;\end{aligned}$$

$$b_3(n)=\Phi(1-\alpha+\varepsilon(n)\phi(n))^2+(\varepsilon(n)\phi(n))^2;$$

$$\begin{aligned}b_4(n)=&2[\Phi\varepsilon(n)(1-\alpha+\varepsilon(n)\phi(n))-\varepsilon(n)\phi(n)(1-\varepsilon(n))]v^T(n)\\&-2[\Phi(1-\alpha+\varepsilon(n)\phi(n))^2+\varepsilon^2(n)\phi^2(n)]Z^T(n)\\&-\varepsilon(n)\phi(n)\xi^T(n);\end{aligned}$$

$$b_5(n)=\sqrt{\varepsilon(n)}+\varepsilon^2(n)(1+\mathbf{E}_n(\phi^2(n))).$$

Obviously at beginning, if $\varepsilon(n)$ is large enough, $b_1(n)$ and $b_2(n)$ must be positive so that $\mathbf{E}_nL(\theta(n+1))-L(\theta(n))\ge 0$. Since $\varepsilon^2(n)$ and $\varepsilon^{\frac{3}{2}}(n)$ decrease faster than $\varepsilon(n)$, when $n$ is large enough, $b_1(n)$ and $b_2(n)$ are negative. Along with $n\to\infty$, $\mathbf{E}_n\left[b_4(n)(Q^r(n)-\mathbf{E}_nQ^r(n))\right]\to 0$, and $b_5\mathbf{E}_n[\xi(n)]^2\to 0$. That means there exists $N_k<\infty$ such that when $n>N_k$, there is a positive non-decreasing function $k(\theta(n))$ to satisfy

$$\begin{aligned}&\mathbf{E}_nL(\theta(n+1))-L(\theta(n))\\&\le-k(\theta(n))+\mathbf{E}_n\left[b_3(n)\|Q^r(n)-E_nQ^r(n)\|^2\right].\end{aligned}\quad (A.9)$$

For a large $n$, (A.9) implies that the right side of (A.9) is negative outside of a neighborhood of the set $\left\{\theta\middle|k(\theta(t))\le\mathbf{E}_n\left[b_3(n)\|Q^r(t)-\mathbf{E}_nQ^r(t)\|^2\right]\right\}$. Outside such decreasing neighborhood, $L(\theta(n))$ has the supermartingale property. Then supermartingale convergence theorem implies that neighborhood of the set $\left\{\theta\middle|k(\theta(t))=\mathbf{E}_n\left[b_3(n)\|Q^r(t)-\mathbf{E}_nQ^r(t)\|^2\right]\right\}$ is recurrent, that is, $\theta(n)$ returns to it infinitely often with probability

one. Since $\mathbf{E}_n\left[b_3(n)\left\|Q^r(t)-\mathbf{E}_nQ^r(t)\right\|^2\right]\to\Phi(1-\alpha)^2\mathbf{E}_n\left\|Q^r(t)-\mathbf{E}_nQ^r(t)\right\|^2$ as $n\to\infty$, $\theta(n)$ returns to neighborhood of $\left\{\theta\middle|k(\theta(t))=\Phi(1-\alpha)^2\mathbf{E}_n\left\|Q^r(t)-\mathbf{E}_nQ^r(t)\right\|^2\right\}$ infinitely often as $n\to\infty$.

**(2) Proof on bound of $E\|H(n)\|^2$**

For $n\le N_k$, where $N_k$ is defined as above, the proof of the bound of $\mathbf{E}\|H(n)\|^2$ is very similar to that of Lemma 5.4.1 in [17]. From (A.6) and (A.7), we observe that if $\varepsilon(n)<\infty$, $\mathbf{E}\|H(n)\|^2$ and $\mathbf{E}[L_\theta{}^T(\theta(n)\gamma(n))]$ are growing at most as $O(|\theta|^2)$. Therefore there are two positive constants $K_1$ and $K_2$ such that

$$\mathbf{E}\|H(n)\|^2+\mathbf{E}[L_\theta{}^T(\theta(n)\gamma(n))]\le K_1L(\theta(n))+K_2. \quad (A.10)$$

Firstly considering that $\mathbf{E}L(\theta(0))<\infty$ and (A.10), we know that $\mathbf{E}\|H(0)\|^2<\infty$. Suppose that $\mathbf{E}L(\theta(n))\le\infty$ for some $n$. Then $\mathbf{E}\|H(n)\|^2<\infty$. Using truncated Taylor series expansion shown in (A.5), (A.10) implies that there is a real $K_3$ such that the right side of (A.5) is bounded above by $\varepsilon K_3[1+L(\theta(n))]$. Since it is assumed that $\mathbf{E}L(\theta(n))<\infty$, it follows that $\mathbf{E}L(\theta(n+1))<\infty$ and $\mathbf{E}\|H(n+1)\|^2<\infty$. Thus by induction, it is proved for $n\le N_k$, $\mathbf{E}L(\theta(n))<\infty$ and $\mathbf{E}\|H(n)\|^2\le\infty$.

Since as $n\to\infty$, $K_1$ and $K_2$ go to infinity, instead of induction introduced above to prove the bound of $\mathbf{E}\|H(n)\|^2$, another induction method is introduced. In the last iteration of previous induction, it is obtained that $\mathbf{E}L(\theta(N_k+1))<\infty$ and $\mathbf{E}\|H(N_k+1)\|^2<\infty$. We suppose that for some $n>N_k$, $\mathbf{E}L(\theta(n))<\infty$. Because the second term on the right side of (A.9) is $\mathsf{F}(n)$-measurable, we have $\mathbf{E}L(\theta(n+1))<\infty$ from (A.9). From (A.5) and (A.9), we have

$$\varepsilon L'_\theta(\theta(n+1))\gamma(n+1)+\mathbf{E}_{n+1}(\varepsilon^2H(n+1)^T\begin{bmatrix}1&0\\0&\Phi\end{bmatrix}H(n+1))$$
$$\le -k(\theta(n+1))+\mathbf{E}_n\left[b_3(n+1)\left\|Q^r(n+1)-\mathbf{E}_nQ^r(n+1)\right\|^2\right].$$

Since the right side of inequality is $\mathsf{F}(n+1)$-measurable and converges to zero according to (A.9), the left side must be bounded. Consequently if $\gamma(n+1)<\infty$, $\mathbf{E}\|H(n+1)\|^2$ must be bounded, or $\mathbf{E}\|H(n+1)\|^2<\infty$. Therefore for $n>N_k$, it is also proved that $\mathbf{E}\|H(n)\|^2<\infty$. By these two inductions we have concluded $\mathbf{E}\|H(n)\|^2<\infty$ for each $n$.

**(3) Proof on the asymptotic rate of change conditions**

Define

$$M^0(t)=\sum_{i=0}^{m(t)-1}\varepsilon(i)\delta M(i),\quad \delta M(n)=H(n)-\mathbf{E}_nH(n),$$

where $m(t)$ denotes the unique value of $n$ such that $t_n\le t<t_{n+1}$. Since it is proved that $\mathbf{E}\|H(n)\|^2<\infty$ for each $n$, it is obvious that $\delta M(n)$ must be bounded. Now we should prove that for each $\mu$ and $T$, the following equation hold.

$$\lim_{n\to\infty}P\left\{\sup_{j\ge n}\max_{0\le t\le T}\left|\sum_{i=m(jT)}^{m(jT+t)-1}\varepsilon(i)\delta M(i)\right|\ge\mu\right\}=0. \quad (A.11)$$

Assume that (A.11) does not hold, then we can choose a sequence $\{\delta M(n)\}$ such that

$$\lim_{n\to\infty}\max_{0\le t\le T}\left|\sum_{i=m(jT)}^{m(jT+t)-1}\varepsilon(i)\delta M(i)\right|\ge\mu.$$

Since

$$\max_{0\le t\le T}\left|\sum_{i=m(jT)}^{m(jT+t)-1}\varepsilon(i)\delta M(i)\right|\le\sum_{i=m(jT)}^{m(jT+T)-1}\varepsilon(i)\left|\delta M(i)\right|$$
$$\le[m(jT+T)-m(jT)-1)\varepsilon(m(jT)]\delta M^m,$$

where $\delta M^m=\max_{m(jT)\le i\le m(jT+1)-1}\left|\delta M(i)\right|$. It follows that

$$\lim_{n\to\infty}[m(jT+T)-m(jT)-1)\varepsilon(m(jT)]\delta M^m\ge\mu.$$

From Assumption 1), we know $\lim_{n\to\infty}\varepsilon(n)\to0$. Then $\delta M^m$ must be infinity and grow at least as $O(\varepsilon^{-1}(n))$. That contradicts the previous sentence that $\delta M(n)$ is bounded. Therefore (19) holds. According to Theorem 5.3.2 in [17], the following conditions for asymptotic rate of change hold:

$$\lim_{n\to0}\sup_{j\ge n}\max_{0\le t\le T}\left|M^0(jT+t)-M^0(jT)\right|=0. \quad (A.12)$$

**(4) Completion of the proof**

Applying definition of $\delta M(n)$ and $\gamma(n)$, we define the sequence of shifted processes as

$$\theta^n(t)=\theta(n)+\sum_{i=n}^{m(t_n+t)-1}\varepsilon\gamma(i)+\sum_{i=n}^{m(t_n+t)-1}\varepsilon\delta M(i). \quad (A.13)$$

We have proved that, (A.9) implies that $\theta(n)$ returns to the neighborhood of $\left\{\theta\middle|k(\theta(t))=\Phi(1-\alpha)^2\mathbf{E}\left\|Q^r(t)-\mathbf{E}_nQ^r(t)\right\|^2\right\}$ infinitely often as $n\to\infty$. Since $Q^r(n)$ is bounded, and $\mathbf{E}_n\left\|Q^r(n)\right\|$ is nonincreasing over the iterations, $\Phi(1-\alpha)^2\mathbf{E}\left\|Q^r(t)-\mathbf{E}_nQ^r(t)\right\|^2$ must be bounded. Define a set as $\Psi_\lambda=\{\theta:L(\theta)\le\lambda\}$, and let $\Psi_\lambda^c$ represent the complement of the set $\Psi_\lambda$. Fix $\delta$ and $\Delta>2\delta+\Phi(1-\alpha)^2\mathbf{E}\left\|Q^r(n)-\mathbf{E}_nQ^r(n)\right\|^2>0$, and let $\tau$ denote a stopping time such that $\theta(\tau)\in\Psi_\delta$. Obviously for

large $n > N_k$ which makes (A.9) hold, and $\varepsilon(n)$ become trivial, $\gamma(n)$ can not force $\theta(n)$ out of $\Psi_\Delta$. Then the only way to force $\theta^n(t)$ out of $\Psi_\Delta$ infinitely often is by the effects of $\{\delta M_i, i > \tau\}$. But according to the definition of $M^0(t)$ and (A.12), it is implied that

$$\lim_{n\to 0} \sum_{i=\tau}^{m(n+t)} \varepsilon(i)\delta M(i) I_{\{\theta(i)\in\Psi_\Delta\}} = 0. \qquad \text{(A.14)}$$

Therefore for finite and large enough $\tau$, these martingale difference terms can not force $\theta^n(t)$ from $\Psi_\delta$ to $\Psi_\Delta$ infinity often. This implies $\theta(n)$ must converge to the set $\left\{\theta \middle| k(\theta(t)) = \Phi(1-\alpha)^2 \mathbf{E}\left\|Q^r(t) - \mathbf{E}_n Q^r(t)\right\|^2\right\}$ with probability one.

## REFERENCES


[1] Clarke, D., Mohtadi, C., and Tuffs, P., "Generalized Predictive Control-Part I: the Basic Algorithm", *Automatica*, Vol. 23, No. 2, 1987, pp. 137-148.

[2] Clarke, D., Mohtadi, C., and Tuffs, P., "Generalized Predictive Control-Part II: Extensions and Interpretations", *Automatica*, Vol. 23, No. 2, 1987, pp. 149-160.

[3] Nørgaard, M., Ravn, O., Poulsen, N. K., and Hansen, L.K., *Neural networks for modeling and control of dynamic systems.* Springer, London, 2000.

[4] Narendra, K. S. and Parthasarathy, K., "Identification and Control of Dynamical Systems Using Neural Networks", *IEEE Trans. on Neural Networks*, Vol. 1, No. 1, 1990, pp. 4-27.

[5] Saerens, M. and Soquet, A., "A Neural Controller", *Proc. of 1st International Conference of Artificial Neural Networks*, London, 1989, pp. 211-215.

[6] Tan, Y. and Cauwenberghe, A. V., "Neural-Network-Based D-step-Ahead Predictors for Nonlinear Systems with Time Delay", *Engineering Applications of Artificial Intelligence*, Vol. 12, No. 1, 1999, pp. 21-35.

[7] Piche, S., Sayyar-Rodsari, B., Johnson, D., and Gerules, M., "Nonlinear Model Predictive Control Using Neural Networks", *IEEE Contr. Syst. Mag.*, Vol. 20, No. 3, 2000, pp. 53-61.

[8] Huang, J. Q., Lewis, F. L., "Neural-Network Predictive Control for Nonlinear Dynamic Systems with Time-Delay", *IEEE Trans. Neural Networks*, Vol. 14, No. 2, 2003, pp. 238-245.

[9] Noriega, J. R. and Wang, H., "A Direct Adaptive Neural-Network Control for Unknown Nonlinear Systems and Its Application", *IEEE Tran. Neural Networks*, Vol. 9, No. 1, 1998, pp. 27-34.

[10] Yoo, S. J., Choi, Y. H., and Park, J. B., "Generalized Predictive Control Based on Self-Recurrent Wavelet Neural Network for Stable Path Tracking of Mobile Robots: Adaptive Learning Rates Approach", *IEEE Trans. Circuits and Systems*, Vol. 53, No. 6, 2006, pp. 1381-1394.

[11] Eberhart, R. C. and Kennedy, J., "A New Optimizer Using Particle Swarm Theory", *Proc. Of the 6th Int. Symp. on Micro Machine and Human, Science*, Nagoya, Japan, 1995, pp. 39-43.

[12] Kennedy, J. and Eberhart, R.C., "Particle Swarm Optimization", *Proc. of IEEE International Conference on Neural Network*, Perth, Australia, 1995, pp. 1942-1948.

[13] Yoshida, H., Kawata, K., Fukuyama, Y. and Nakanishi, Y., "A Particle Swarm Optimization for Reactive Power and Voltage Control Considering Voltage Stability", *Proc. of International Conference on Intelligent System Application to Power Systems*, Rio de Janeiro, Brazil, 1999, pp. 117-121.

[14] Abido, M. A., "Particle Swarm Optimization for Multimachine Power System Stabilizer Design", *Proc. of Power Engineering Society Summer Meeting*, 2001, pp. 1346-1351.

[15] Messerschmidt, L. and Engelbrecht, A. P., "Learning to Play Games Using a PSO-Based Competitive Learning Approach", *IEEE Trans. Evolutionary Computation*, Vol. 8, No. 3, 2004, pp. 280-288.

[16] Li, Y. and Chen, X., "Mobile Robot Navigation Using Particle Swarm Optimization and Adaptive NN", *Proc. of the First International Conference on Natural Computation (ICNC), Lecture Notes in Computer Science*, Springer, Vol. 3612, 2005, pp. 554-559.

[17] Kushner, H. J. and Yin, G. G., *Stochastic Approximation and Recursive Algorithms and Applications.* 2nd Edition, Springer, New York, 2003.

[18] Plett, G. L., "Adaptive Inverse Control of Linear and Nonlinear Systems Using Dynamic Neural Networks", *IEEE Trans. Neural Networks*, Vol. 14, No. 2, 2003, pp. 360 - 376.

[19] Shi, Y. and Eberhart, R. C., "Parameter Selection in Particle Swarm Optimization", *Proc. of the 7th Annual Conference on Evolutionary Programming*, New York, 1998, pp. 591-600.

[20] Shi, Y. and Eberhart, R.C., "An Empirical Study of Particle Swarm Optimization", *Proc. of IEEE Congress on Evolutionary Computation*, Washington, DC, 1999, pp. 1945-1949.

[21] Zhang, W. J. and Xie, X.F. , "DEPSO: Hybrid Particle Swarm with Differential Evolution Operator", *Proc. of IEEE Int. Conf. on Systems, Man and Cybernetics*, Washington DC, USA, 2003, pp. 3816-3821.

[22] Ratnaweera, A., Halgamuge, S. K., and Watson, H. C., "Self-Organizing Hierarchical Particle Swarm Optimizer with Time-Varying Acceleration Coefficients", *IEEE Trans. on Evolutionary Computation*, Vol. 8, No.3, 2004, pp. 240-255.

[23] van den Bergh, F and Engelbrecht, A. P., "A Cooperative Approach to Particle Swarm Optimization", *IEEE Trans. on Evolutionary Computation*, Vol. 8, No. 3, 2004, pp. 225-239.


**Xin Chen** received the B.S. degree in Industrial Automatization, and M.S. degree in Control Theory and Control Engineering from Central South University, Changsha, China in 1999 and 2002 respectively. He received Ph. D in Electromechanical Engineering from univer sity of Macau in 2007. His research interests include swarm intelligence, multiple robot coordination.

**Yangmin Li** received his B. S. and M.S. degrees from the Mechanical Engineering Department, Jilin University, Changchun, China in 1985 and 1988 respectively. He received his Ph.D from the Mechanical Engineering Department, Tianjin University, Tianjin, China in 1994. After that, he worked in South China University of Technology, International Institute for Software Technology of the United Nations University (UNU/IIST), University of Cincinnati, and Purdue University. He is currently a Professor majored in robotics, mechatronics, control, and automation in University of Macau. He is an IEEE senior member and a member of ASME. He has been serving as a council member and editor of Chinese Journal of Mechanical Engineering since 2004. He is active in organizing and participating international conferences, he has served as a member of international program committee for 35 international conferences. He has authored or co-authored about 160 papers in international journals and conferences.