Electronics and Communications in Japan, Vol. 95, No. 7, 2012
Translated from Denki Gakkai Ronbunshi, Vol. 131-C, No. 9, September 2011, pp. 1642–1649

Adaptive PSO for Online Identification of Time-Varying Systems

TAKESHI NISHIDA and TETSUZO SAKAMOTO
Kyushu Institute of Technology, Japan

SUMMARY

Novel adaptive PSO (particle swarm optimization) algorithms called OPSO (online PSO) and εPSO are proposed. These algorithms are designed to prevent increasing of calculation cost in order to embed into online systems, and they have high adaptive performances to environmental changes. The efficiency of the proposed PSO algorithms are demonstrated by numerical simulation and the online identification of an actual dynamic system. © 2012 Wiley Periodicals, Inc. Electron Comm Jpn, 95(7): 10–18, 2012; Published online in Wiley Online Library (wileyonlinelibrary.com). DOI 10.1002/ecj.11391

Key words: adaptive PSO; time-varying systems; online identification; OPSO; εPSO.

1. Introduction

PSO (Particle Swarm Optimization) is a statistical method of search for optimal solutions. This approach imitates swarm intelligence and is efficient for nonlinear programming problems [1]. The performance of PSO, offering robust and fast search for optimal solutions to nonlinear problems, nondifferentiable problems, multimodal problems, etc., has been verified by many studies, and a number of applications have been implemented [2]. PSO was first proposed for solving static optimization problems. Recently, the method was modified to deal with real dynamic systems, and new PSO algorithms can adapt to environmental changes caused by observation noise or time variation of a system, that is, to a varying search space [3–9]. Such PSO algorithms are designed for time-varying (dynamic) search spaces, and emphasis is placed on strategies for detecting environmental changes and avoiding convergence to local minimum.

On the other hand, while offering high performance, PSO is based on stochastic global search using numerous particles, which results in poor computational efficiency compared to conventional gradient methods. In order to apply PSO to a real online system, prompt response to environmental changes is required, and the calculations at every step must be completed within the sampling period, depending on the system configuration. Therefore, in addition to environmental adjustability, reduction of computational complexity is important for PSO incorporated into online systems. However, no adaptive PSO algorithms with reduced complexity intended for online systems have been proposed so far. Thus, in this study, we propose two PSO algorithms with adaptability to environmental changes while restraining increases in computational complexity. In addition, we demonstrate the effectiveness of the proposed algorithms compared to conventional methods by numerical simulations and experiments.

The paper is organized as follows. First we explain PSO in Section 2 and give a review of existing PSO algorithms in Section 3. Then we propose two types of PSO algorithms applicable to online systems in Sections 4 and 5, and discuss the strategic features of the proposed methods in Section 6. We next evaluate the performance of the proposed algorithms by numerical simulations in Section 7, and demonstrate their effectiveness by experiments with a real system in Section 8. A summary of the study is given in Section 9.

2. PSO Algorithm

PSO is an algorithm in which solutions are discovered by moving numerous search points, called particles, over the search space based on the past action history and the dynamically adjusted velocity. Consider the optimization of the following optimization function f: $\mathbb{R}^l \to \mathbb{R}$.

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}) \geq 0 \tag{1}$$

In a conventional PSO algorithm to solve this problem [2], the position and velocity of particles in the search space are denoted by $\boldsymbol{x}_k^{(m)} \in \mathbb{R}^l$ and $\boldsymbol{v}_k^{(m)} \in \mathbb{R}^l$, respectively. Here $m = [1, M] \in \mathbb{N}_+$ is the number of the particle, and $k = 1, 2, \ldots$ is a discrete time point. These are updated as follows.

Given: Parameters ω, c_1, c_2 are specified. $\boldsymbol{x}_0^{(m)}$ and $\boldsymbol{v}_0^{(m)}$ are set using uniform random numbers. The range of uniform random numbers is adjusted to the search space.

Step 1: The position of each particle is evaluated by using the minimizing scalar function $f(\cdot)$, and the position of every particle producing the smallest fitness value at the current instant (usually called *pbest*),

$$\hat{\boldsymbol{x}}^{(m)} := \begin{cases} \boldsymbol{x}_k^{(m)} & \text{if } f(\boldsymbol{x}_k^{(m)}) < f(\hat{\boldsymbol{x}}^{(m)}) \\ \hat{\boldsymbol{x}}^{(m)} & \text{otherwise} \end{cases} \tag{2}$$

is saved together with the fitness value $f(\hat{\boldsymbol{x}}^{(m)})$. Here := denotes substitution.

Step 2: The position producing the smallest fitness value in the entire swarm

$$\hat{\boldsymbol{x}}^{g} := \begin{cases} \hat{\boldsymbol{x}}^{(m)} & \text{if } f(\hat{\boldsymbol{x}}^{(m)}) < f(\hat{\boldsymbol{x}}^{g}) \\ \hat{\boldsymbol{x}}^{g} & \text{otherwise} \end{cases} \tag{3}$$

is called *gbest*, and is recognized as the optimal solution; its fitness value $f(\hat{\boldsymbol{x}}^g)$ is stored.

Step 3: The positions and velocities of the particles are updated by the equations

$$\begin{aligned} \boldsymbol{v}_k^{(m)} &= \omega \boldsymbol{v}_{k-1}^{(m)} + c_1 r_1 (\hat{\boldsymbol{x}}^{(m)} - \boldsymbol{x}_{k-1}^{(m)}) \\ &\quad + c_2 r_2 (\hat{\boldsymbol{x}}^{g} - \boldsymbol{x}_{k-1}^{(m)}) \\ \boldsymbol{x}_k^{(m)} &= \boldsymbol{x}_{k-1}^{(m)} + \boldsymbol{v}_k^{(m)} \end{aligned}$$

Here r_1, r_2 = [0, 1] are uniform random numbers.

Step 4: Step 1 is repeated with $k := k + 1$ until the convergence condition is satisfied.

In the above conventional PSO algorithm, $f(\hat{\boldsymbol{x}}^g)$ decreases monotonically, as is obvious from Eq. (3). Now consider the case in which the evaluation function changes from $f(\cdot)$ to $f^*(\cdot)$ so that the optimal solution changes from $\hat{\boldsymbol{x}}^g$ to $\hat{\boldsymbol{x}}^{g*}$. In conventional PSO, variation of the evaluation function is not considered, and therefore the fitness value $f(\hat{\boldsymbol{x}}^g)$ is not updated unless $f(\hat{\boldsymbol{x}}^{g*}) < f(\hat{\boldsymbol{x}}^g)$, and in this case $\hat{\boldsymbol{x}}^g$ becomes trapped in an inappropriate solution. In particular, when the evaluation function $f(\hat{\boldsymbol{x}}^g)$ converges after a sufficient time period, such environmental changes have an even higher probability of remaining undetected. In other words, time variation of the evaluation function $f(\cdot)$ is not assumed in the conventional PSO algorithm, and therefore optimal solutions cannot be updated according to environmental changes such as moving of the optimal solution or a change of evaluation function.

3. Related Research

3.1 Adaptive PSO

Here we give an overview of existing PSO algorithms proposed to deal with environmental changes.

In early studies [3, 4], the memory of all particles was reset periodically, and the *pbest* values were rearranged according to the current position vectors. However, it was pointed out that it was difficult to determine the initialization frequency [8]. That is, so long as the periodicity of environmental changes is not known in advance, the particle memories must be reset frequently so as to assure adaptation to the environment; however, frequent initialization degrades convergence speed and destabilizes the search.

Then APSO (Adaptive PSO) [5] was proposed using the notion of *sentries* to detect and respond to environmental changes. The sentry particles are designed specially to detect environmental changes, and are distributed over the search space. When a sentry particle detects an environmental change, it informs the other sentry particles, and the ordinary particles are forced to reset their memories. However, a sentry particle can detect local changes only where it resides, and is unable to distinguish between environmental changes and observation noise; as a result, the memories of all particles are reset too often in real applications with observation noise.

After that, new PSO algorithms with robustness to environmental changes were proposed: for example, CPSO (Charged PSO) [6], which introduces a model based on repulsion and attraction between particles to deal with dynamic environments, and MSO (Multi-Swarm Optimization) [7], which combines multiple swarms. In MSO, multiple swarms are used to avoid convergence of all particles to the same optimal solutions. These algorithms modify the search to make it more efficient and to allow it to escape from local minimum. However, continuous or regular environmental changes are not assumed; as a result all particles are initialized when an environmental change is detected, and the problem of excessive initialization remains. In addition, multiple parameters are used to determine the repulsion between particles, and frequent adjustment of such parameters adds to computational complexity.

To solve the above problems, a number of PSO algorithms have been developed recently. One of these is DAPSO (Distributed Adaptive PSO) [8], which adapts to continuous environmental changes by multiplying the particle fitness by a constant. In this algorithm, adaptation to environmental changes, with no significant increase in computational complexity, is made possible due to a simple procedure in which *pbest* is multiplied by a constant slightly greater than 1 for minimum search at the current instant. In addition, in MPSO (Modified PSO) [9], the particles are evaluated at every instant with respect to environmental changes and the move of the optimal solution in order to achieve accurate particle updating. This algorithm increases the computational complexity but provides prompt adaptation to continuous environmental changes.

However, none of the previous studies considers reduction of computational complexity, which is important for implementation in real online systems.

4. OPSO

We configured the following OPSO (Online PSO) algorithm in which a time-varying evaluation function $f_k(\cdot)$ is used to explicitly express environmental changes, and *pbest* and *gbest* are assumed to be time-varying values, respectively, $\hat{\boldsymbol{x}}_k^{(m)}$ and $\hat{\boldsymbol{x}}_k^g$.

4.1 Algorithm

Given: Parameters ω, c_1, c_2 are specified. $\boldsymbol{x}_0^{(m)}$ and $\boldsymbol{v}_0^{(m)}$ are set using uniform random numbers. The algorithm starts at Step 1 at $k = 1$.

Step 1: The optimal solutions of every particle at the previous instant $\hat{\boldsymbol{x}}_{k-1}^{(m)}$ are reevaluated at the current instant, and the smallest value is found:

$$\tilde{\boldsymbol{x}}_k^g = \arg\min\left\{f_k(\hat{\boldsymbol{x}}_{k-1}^{(m)})\right\} \tag{4}$$

Step 2: The velocity and position of every particle are updated:

$$\begin{aligned} \boldsymbol{v}_k^{(m)} &= \omega\boldsymbol{v}_{k-1}^{(m)} + c_1 r_1(\hat{\boldsymbol{x}}_{k-1}^{(m)} - \boldsymbol{x}_{k-1}^{(m)}) \\ &\quad + c_2 r_2(\tilde{\boldsymbol{x}}_k^g - \boldsymbol{x}_{k-1}^{(m)}) \\ \boldsymbol{x}_k^{(m)} &= \boldsymbol{x}_{k-1}^{(m)} + \boldsymbol{v}_k^{(m)} \end{aligned}$$

Step 3: The position of every particle producing the smallest fitness value is saved together with this value $f_k(\hat{\boldsymbol{x}}_k^{(m)})$.

$$\hat{\boldsymbol{x}}_k^{(m)} = \begin{cases} \boldsymbol{x}_k^{(m)} & \text{if } f_k(\boldsymbol{x}_k^{(m)}) < f_k(\hat{\boldsymbol{x}}_{k-1}^{(m)}) \\ \hat{\boldsymbol{x}}_{k-1}^{(m)} & \text{otherwise} \end{cases}$$

Step 4: The best solution in the entire swarm is found:

$$\hat{\boldsymbol{x}}_k^g = \arg\min\left\{f_k(\hat{\boldsymbol{x}}_k^{(m)})\right\}$$

This is taken as the OPSO estimate at instant k.

Step 5: The algorithm returns to Step 1 at $k = k + 1$.

4.2 Features and properties of OPSO

In order to find the best solution for time-varying function $f_k(\cdot)$, the previous values of *pbest* are first reevaluated at Step 1 using the current evaluation function, and the corresponding particle position $\tilde{\boldsymbol{x}}_k^g$ is found. This particle position is the *pbest* closest to the best solution for the current evaluation function. Since the evaluation function varies with time, it might happen so that $f_k(\hat{\boldsymbol{x}}_{k-1}^{(m)}) \geq f_{k-1}(\hat{\boldsymbol{x}}_{k-1}^{(m)})$; in this case, $f_k(\hat{\boldsymbol{x}}_{k-1}^{(m)})$ does not decrease monotonically. As a result, the fitness value $f_k(\hat{\boldsymbol{x}}_k^{(m)})$ of $\hat{\boldsymbol{x}}_k^{(m)}$ calculated in Step 3 does not decrease monotonically, nor does the fitness value $f_k(\hat{\boldsymbol{x}}_k^g)$ of $\hat{\boldsymbol{x}}_k^g$ calculated at Step 4. Thus, the procedure introduced at Step 1 solves the problem of monotonic decrease of $f(\cdot)$ that prevented adaptation to environmental changes in the conventional PSO; in the proposed algorithm, the environmental changes are incorporated by updating *pbest* and *gbest*.

At Steps 2 and 3, the velocities and positions of the particles are updated using $\tilde{\boldsymbol{x}}_k^g$. Finally, at Step 4, the updated particles are evaluated, the personal best solutions $\hat{\boldsymbol{x}}_k^{(m)}$ and global best solution $\hat{\boldsymbol{x}}_k^g$ are obtained, and the algorithm proceeds to the next time point. Due to the introduction of Step 1, updating and evaluation of the particle velocity and position are performed in reverse order compared to the conventional PSO. That is, the particles are updated at Step 1 so as to reflect adaptation to environmental changes, and after that, every particle is evaluated to obtain the best solution at the current instant.

OPSO has the following features: (1) the fitness values of *pbest* and *gbest* do not decrease monotonically due to the use of a time-varying evaluation function; (2) no design parameters are added; (3) the calculation procedures are simple and the increase of computational complexity is restrained; (4) compared to the conventional PSO, the increase of computational complexity in OPSO is confined to Step 1.

5. εPSO

In the εPSO algorithm described below, the problem of a monotonic decrease of $f(\cdot)$ in the conventional PSO is solved by introduction of a mechanism to forget the optimal value (*pbest*) with the passage of time. This algorithm produces an easier procedure and allows further restraint of the increase in computational complexity than OPSO.

Just as in OPSO, a time-varying evaluation function $f_k(\cdot)$ is used to explicitly express environmental changes, and *pbest* and *gbest* are assumed to be time-varying values, respectively $\hat{\boldsymbol{x}}_k^{(m)}$ and $\hat{\boldsymbol{x}}_k^g$.

5.1 Algorithm

Given: Parameters ω, c_1, c_2 are specified. $\boldsymbol{x}_0^{(m)}$ and $\boldsymbol{v}_0^{(m)}$ are set using uniform random numbers. The algorithm starts at Step 1 with $k = 1$.

Step 1: A small positive constant is added to the fitness value of the best solution and the past fitness value is forgotten:

$$f_k(\hat{\boldsymbol{x}}_{k-1}^{(m)}) := f_{k-1}(\hat{\boldsymbol{x}}_{k-1}^{(m)}) + \varepsilon \quad (\forall m) \tag{5}$$

Here := denotes substitution.

Step 2: The position of every particle producing the smallest fitness value,

$$\hat{\boldsymbol{x}}_k^{(m)} = \begin{cases} \boldsymbol{x}_k^{(m)} & \text{if } f_k(\boldsymbol{x}_k^{(m)}) < f_k(\hat{\boldsymbol{x}}_{k-1}^{(m)}) \\ \hat{\boldsymbol{x}}_{k-1}^{(m)} & \text{otherwise} \end{cases}$$

is saved together with the fitness value $f_k(\hat{\boldsymbol{x}}_k^{(m)})$.

Step 3: The best solution of the entire swarm is saved:

$$\hat{\boldsymbol{x}}_k^g = \arg\min \left\{ f_k(\hat{\boldsymbol{x}}_k^{(m)}) \right\}$$

Step 4: The positions and velocities of the particles are updated:

$$\begin{aligned} \boldsymbol{v}_k^{(m)} &= \omega \boldsymbol{v}_{k-1}^{(m)} + c_1 r_1 (\hat{\boldsymbol{x}}_{k-1}^{(m)} - \boldsymbol{x}_{k-1}^{(m)}) \\ &\quad + c_2 r_2 (\hat{\boldsymbol{x}}_k^g - \boldsymbol{x}_{k-1}^{(m)}) \\ \boldsymbol{x}_k^{(m)} &= \boldsymbol{x}_{k-1}^{(m)} + \boldsymbol{v}_k^{(m)} \end{aligned}$$

Step 5: The algorithm returns to Step 1 at $k = k + 1$.

5.2 Features and properties of εPSO

First a small number ε is added to $f_{k-1}(\hat{\boldsymbol{x}}_{k-1}^{(m)})$ at Step 1. The past fitness value is gradually increased over time so that the fitness values of *pbest* for every particle are forgotten. This procedure makes it possible to discover optimal solutions that have moved because of environmental changes. For example, if the best solution $\hat{\boldsymbol{x}}_k^g$ has moved to $\hat{\boldsymbol{x}}_{k+l}^{g*}$ at a time that is l steps after the environment changed at some instant k, then the particles should search solutions satisfying the following condition:

$$f_{k+l}(\hat{\boldsymbol{x}}_{k+l}^g) < f_k(\hat{\boldsymbol{x}}_k^g) + \varepsilon l$$

As the difference between both sides increases over time, the particles move stochastically to find solutions in the vicinity of $\hat{\boldsymbol{x}}_{k+l}^{g*}$, so that the possibility of updating *pbest* increases over time. In addition, when a particle finds an optimal solution with the same fitness value as the past best, the new solution is adopted.

However, the value of ε must be set properly when using this algorithm. When ε is too small, adaptability deteriorates, and when it is too large, solutions are forgotten and re-discovered excessively frequently. However, εPSO search is not sensitive to the value of ε so long as it is set properly in a certain range, and the value can be adjusted with relative ease via by trial and error.

As shown above, εPSO has the following features: (1) the fitness value of *gbest* does not decrease monotonically because *pbest* is forgotten; (2) the only additional parameter is ε; (3) processing is simple, and the increase in computational complexity compared with the conventional PSO is smaller than in OPSO.

6. Preservation Strategy for *pbest* and *gbest*

The preservation strategy for *pbest* and *gbest* is a highly important element of adaptive PSO which affects the adaptability to the environment. Here we discuss the strategies adopted in OPSO and εPSO as compared to the existing algorithms mentioned in Section 3.

In the algorithms proposed in Refs. 3–7, *pbest* and *gbest* are reset every time an environmental change is detected, and the past memories of the particles are deleted. This preservation strategy of *pbest* and *gbest* has been shown to be inefficient [8].

On the other hand, DAPSO [8] allows the extraction of new solutions with fitness values within a constant multiple of the stored *pbest*. That is, the fitness values of *pbest* are degraded by multiplying by a constant; this strategy is intended to prevent *pbest* and *gbest* from sticking at local minimums, and to provide adaptability to environmental changes. The preservation strategy for *pbest* and *gbest* in εPSO can be considered an extension of this approach. That is, in εPSO, the degradation of the fitness value of *pbest* is increased over time so that the adaptability to the environment gradually improves. Due to this strategy, εPSO offers better adaptability than DAPSO with nearly the same computational complexity.

In MPSO [9], *gbest* and *pbest* are reevaluated at every instant using a time-varying evaluation function, and the particle velocity is corrected. The velocity is recalculated thoroughly with respect to the positional relations of *gbest* and *pbest* at the previous and current instants, which allows accurate adaptation to environmental changes but results in a significant increase in computational complexity. This preservation strategy of *pbest* and *gbest* is simplified in OPSO. Specifically, correction aiming at adaptation to environmental changes is restricted to reevaluation of *pbest* and *gbest* so as to restrain the increase in computational complexity. We may assume that the appropriate velocity cannot be calculated within the current iteration in some cases; on the other hand, this stochastic difference in speed may be expected to converge to zero after sufficient time.

7. Numerical Simulation

We use a time-varying evaluation function to compare the performance of the conventional adaptive PSO and the two proposed algorithms.

7.1 Problem setting

Consider minimization of the following time-varying function:

$$f_k(x_1, x_2) = 1 - \exp\left[-\frac{\{x_1 - 250 - 125\sin(0.01k)\}^2}{2 \cdot 40^2} - \frac{\{x_2 - 250 + 125\cos(0.01k)\}^2}{2 \cdot 40^2} \right]$$

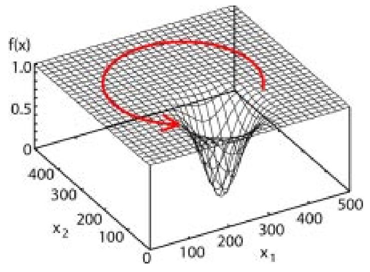


Fig. 1. Target function (this function moves with passage along the red line). [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

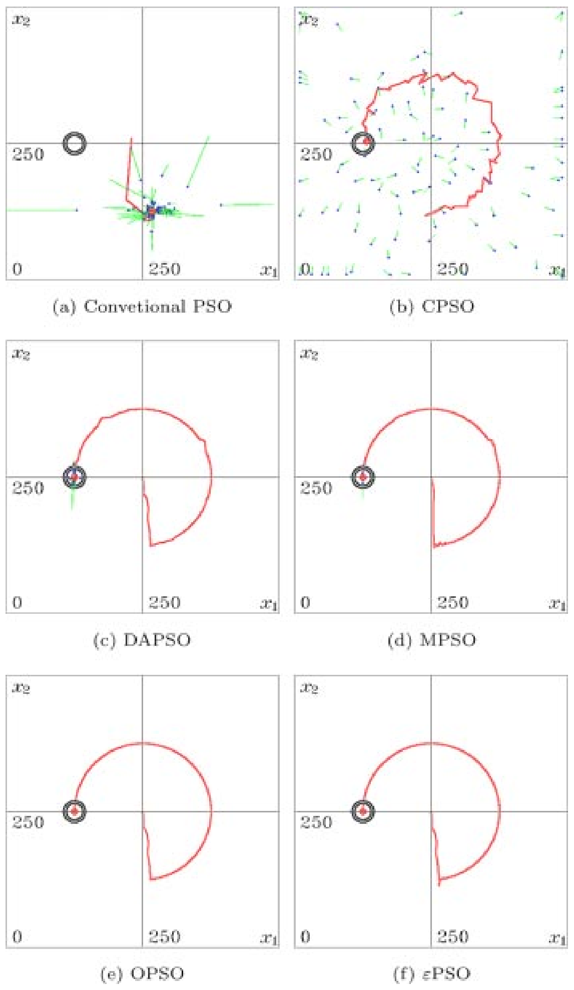


Fig. 2. Results of numerical simulation. (The red solid line represents the trajectory of $\hat{x}_k^g$ and the red circle shows the position of $\hat{x}_k^g$ at $k = 471$. The blue dots and the green line segments represent the positions and velocities of particles; the gray double circle represents the position of the global minimum solution.) [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

The shape and time variation of this function are illustrated in Fig. 1. The evaluation function for the optimal solution has the value 0; its position moves along a circle with its center at $(x_1, x_2) = (250, 250)$ and with a radius of 125, and returns to the initial point in about 628 steps. The objective of the PSO algorithms in this simulation is to continuously track $\hat{x}_k^g$ as the position of the optimal solution for an evaluation function moving at a constant speed. In this simulation we compare conventional PSO, CPSO, DAPSO, MPSO, OPSO, and εPSO. In all the algorithms, the parameters are $\omega = 0.729$, $c_1 = c_2 = 1.4955$ in accordance with Ref. 10, and the number of particles is $M = 100$. The specific parameters for every PSO algorithm are set as proposed by the respective developers.† In εPSO, the constant ε is set empirically to 0.1.‡

7.2 Adaptability

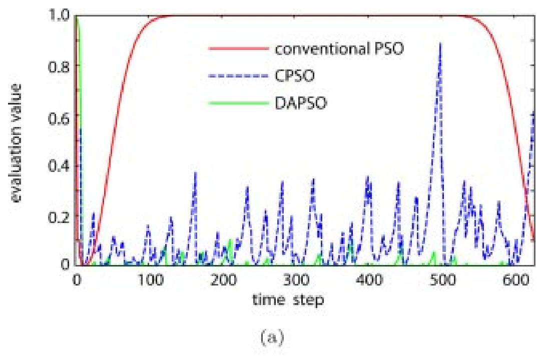
The trajectories of $\hat{x}_k^g$ obtained by applying the algorithms to the target function are shown in Fig. 2. The time evolution of $f_k(\hat{x}_k^g)$ is illustrated in Fig. 3. As can be seen from Figs. 2(a) and 3(a), in conventional PSO the updating stops as soon as a minimum of $f(\hat{x}^g)$ is found, and thus a move of the solution position cannot be followed. As can be seen from Figs. 2(b) and 3(a), in CPSO the particles are initialized at every iteration and the trajectory of $\hat{x}_k^g$ is unstable. As shown in Fig. 2(c), DAPSO updates $f_k(\hat{x}_k^g)$ at a certain interval, thus following the function variation. However, pulsations occur in the transition of $f_k(\hat{x}_k^g)$, which results in distortion of the trajectory. In contrast, the distortion of the $f_k(\hat{x}_k^g)$ trajectory is smaller in MPSO, as shown in Fig. 2(d), and OPSO and εPSO offer smooth tracking of the target function without trajectory distortion. In addition, as can be seen from Fig. 3(b), MPSO, OPSO, and εPSO show smaller transitions of $f_k(\hat{x}_k^g)$ than the conventional PSO, CPSO, and DAPSO. However, the transitions of $f_k(\hat{x}_k^g)$ are larger and include pulsations in the case of MPSO as compared to OPSO and εPSO.

†In DAPSO, the parameter P is set to 2.0.

‡It was confirmed that adaptation became impossible when ε was set below 10^{-5}. On the other hand, in this example with a continuously varying evaluation function, PSO performance was degraded due to excessive forgetting and rediscovery of solutions when ε was set large.

7.3 Computational complexity

The average processing time per step consumed by the algorithms is shown in Fig. 4. The measurement was performed using Vine Linux operated at a CPU clock speed



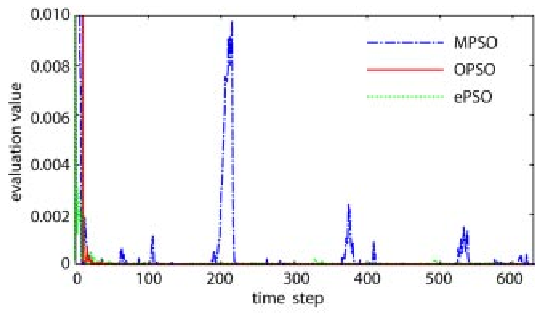


Fig. 3. Time evolution of $f_k(\hat{x}_k^g)$. [The results of conventional PSO, CPSO, and DAPSO are shown in (a), and the results of MPSO, OPSO, and εPSO are shown in (b).] [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

of 2.40 GHz. As indicated by these results, among MPSO, OPSO, and εPSO, which demonstrated high adaptability in the simulations, εPSO has the lowest computational complexity. In addition, OPSO outperforms MPSO in adaptability, with lower computational complexity.

The numerical simulations showed that OPSO and εPSO proposed in this study are superior to existing adaptive PSO algorithms by virtue of higher adaptability and lower computational complexity.

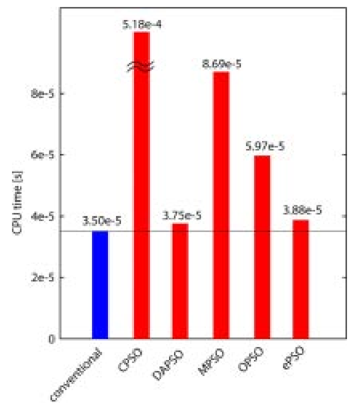


Fig. 4. CPU execution time at each step. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

8. Experiments

Below we present experimental results obtained by applying the proposed algorithms to an online system. We performed the same experiments with MPSO and conventional PSO for comparison.

8.1 Experimental object

The experiments were performed on the velocity control system of a servo motor installed in a film transportation system. The parameters of the discrete transfer function were estimated online. Let the input be the torque generated by the servo motor, and the output be the film velocity. Assuming that the system dynamics can be approximated by a combination of zero-order hold and first-order delay components, the discrete transfer function can be expressed as follows:

$$G(z^{-1}) = \frac{Y(z^{-1})}{U(z^{-1})} = \frac{K\left\{\exp\left(\frac{1}{T_a}T_s\right)+1\right\}z^{-1}}{1-\exp\left(-\frac{1}{T_a}T_s\right)z^{-1}} \tag{6}$$

Here T_a is the time constant and T_s (= 10 ms) is the sampling time. Thus, the system output at instant k can be expressed as

$$y_k = \frac{b_k z^{-1}}{1+a_k z^{-1}} u_k = \boldsymbol{z}_k^T \boldsymbol{x}_k \tag{7}$$

Here

$$a_k = -\exp\left(-T_s/T_a\right) \tag{8}$$

$$b_k = K\left\{\exp\left(T_s/T_a\right)+1\right\} \tag{9}$$

In addition, $\boldsymbol{x}_k \equiv (-a_k, b_k)^T$ and $\boldsymbol{z}_k \equiv (y_{k-1}, u_{k-1})^T$. In preliminary identification experiments, the nominal values were $T_a = 1.43$ and $K = 0.455$; based on these values, $a = -0.993$ and $b = 3.16 \times 10^{-3}$.

The time evolution of the system input and output in the experiments is illustrated in Fig. 5. The motor torque

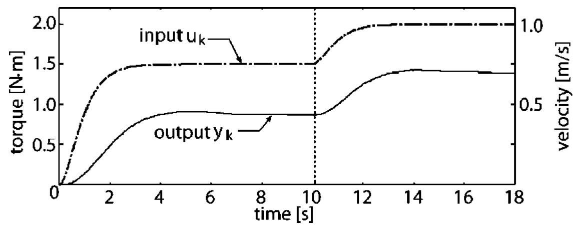


Fig. 5. Time evolution of system input and output.

was generated using a second-order filter. The reference value was changed from 1.5 Nm to 2.0 Nm at 10 s. In this experiment, the roll radius increases as film is reeled in, and hence the system parameters vary over time. In addition, the drive unit includes friction and other nonlinear dynamic components, and therefore the system parameters vary nonlinearly when the reference value is changed. In this experiment, the system parameter $\boldsymbol{x}_k$ in Eq. (7) was estimated online based on the input–output relationship in Fig. 5 using different PSO algorithms.

8.2 PSO settings

The following evaluation function was applied to the particles in every PSO algorithm:

$$f_k\left(\boldsymbol{x}_k^{(m)}\right) = \sum_{j=0}^{I} |y_{k-j} - \hat{y}_{k-j}| = \sum_{j=0}^{I} \left|y_{k-j} - \boldsymbol{z}_{k-j}^{T}\boldsymbol{x}_k^{(m)}\right|$$

Here I is the number of steps in backward evaluation; I was set to 100. The common parameters for all algorithms were set at $\omega = 0.729$, $c_1 = 1.4955$, $c_2 = 1.4955$, $M = 100$. For εPSO, the constant ε was set to 10^{-4} by trial and error. In addition, the MPSO-specific parameter was set in accordance with Ref. 9, that is, $\lambda = 0.1$.

8.3 Experimental results

First, the time evolution of *gbest*, that is, $f_k(\hat{\boldsymbol{x}}_k^g)$, is shown in Fig. 6. As can be seen from the diagram, *gbest* decreases monotonically to converge to a certain value in the conventional PSO; on the other hand, in the other PSO algorithms, *gbest* increases and decreases over time. In particular, modification of the reference value at 10 s affected the input–output relation, and as a result, $f_k(\hat{\boldsymbol{x}}_k^g)$ increased and then decreased again due to adaptive parameter estimation.

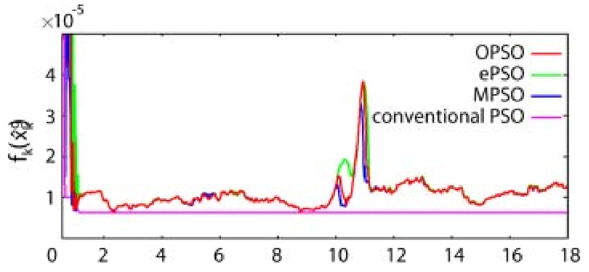


Fig. 6. Time evolution of $f_k(\hat{\boldsymbol{x}}_k^g)$. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

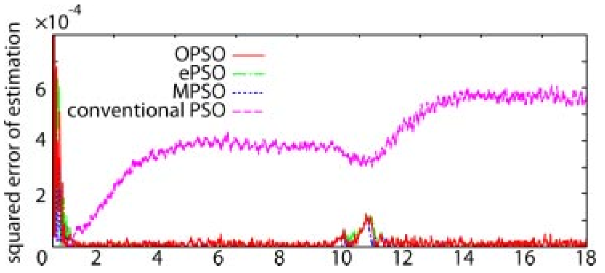


Fig. 7. Estimation error $|y_k - \boldsymbol{z}_k^T\hat{\boldsymbol{x}}_k^g|$. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

Figure 7 shows the time evolution of the difference between the output obtained from the estimated parameter $\hat{\boldsymbol{x}}_k^g$ and the actual output, that is, the estimation error $|y_k - \boldsymbol{z}_k^T\hat{\boldsymbol{x}}_k^g|$. As can be seen from the diagram, the conventional PSO cannot adapt to time variation of the system, and the estimation error grows over time; in contrast, the other PSO algorithms provide accurate estimation of the system output.

The time evolution of the estimate parameters a_k and b_k is illustrated in Fig. 8. As can be seen from the diagrams, *gbest* converges in the conventional PSO, and as a result, the estimated parameters are not updated and deviate from the nominal values. In contrast, in the other PSO algorithms, parameter estimation adapts promptly to the system change, and the estimated parameters agree well with the nominal values.

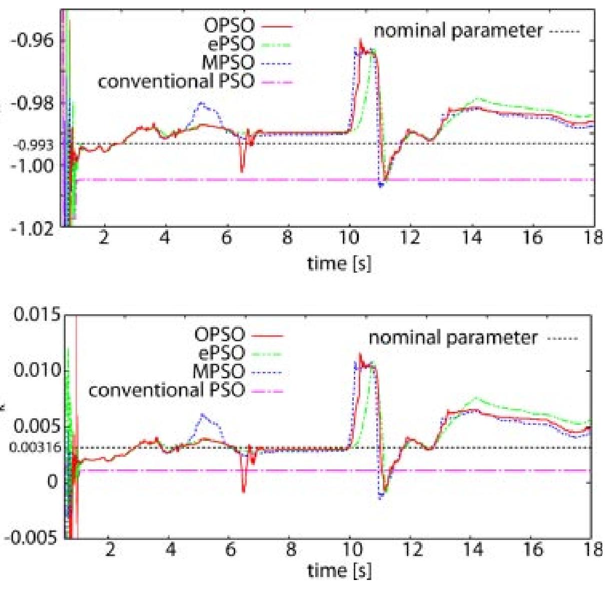


Fig. 8. Time evolution of estimated parameters. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

Table 1. Average CPU time of calculation of each iteration of the PSO algorithms

	MPSO	OPSO	εPSO
time [ms]	0.337	0.170	0.099

Table 1 shows the average calculation time per sampling period. Here the PC performance is the same as in the simulations. As can be seen from the table, the computational complexity of OPSO and εPSO is 50.4% and 29.4%, respectively, compared to MPSO. When PSO is used to estimate objects with high-order response or control systems dealing with multiple subsystems, the time and resources required for PSO calculations increase linearly or exponentially, and hence reduction of computational complexity is very important.

The above experiments showed that with OPSO and εPSO algorithms proposed in this study, the same level of optimization performance as in conventional PSO can be achieved with lower computational complexity.

9. Conclusions

In this paper, we first explained that the conventional PSO cannot adapt to observation noise and changes of system parameters because the evaluation function $f(\cdot)$ decreases monotonically. To solve this problem, we proposed OPSO and εPSO algorithms, which offer high adaptability, while restraining the increase of computational complexity. The former algorithm uses explicit representation of the time variation of evaluation function, and includes a procedure preventing the fitness value of *pbest* from monotonically decreasing. The latter algorithm solves the problem of monotonic decrease by forgetting the past fitness values of *pbest*. In addition, these algorithms are designed straightforwardly, which makes them suitable for online identification of system parameters. We carried out numerical simulations to verify the effectiveness of the proposed algorithms by comparison with existing adaptive PSO algorithms. In addition, we investigated the performance of the proposed algorithms by experiments with an online system. The results demonstrated that the proposed OPSO and εPSO algorithms offer the same optimization performance as existing methods but with higher speed.

In our simulations we used the recommended parameters for existing PSO algorithms; in the future, detailed examination is needed to determine how the parameter settings affect the properties of the proposed algorithms. Another topic for future research is the relationship between the properties of the time-varying evaluation function and the PSO parameter settings. In addition, in this study, we evaluated the performance of the proposed methods using simulations and experiments of relatively low order; further evaluation should involve higher-order search space.

Topics for further development will include configuring a self-tuning control system based on the proposed adaptive PSO, because accurate online estimation of the coefficient vectors of discrete transfer functions makes possible self-tuning adaptive control.

REFERENCES

1. Kennedy J, Eberhart RC. Particle swarm optimization. Proc of IEEE Int Conf on Neural Network IV, p 1942–1948, 1995.
2. Ishigame A. Particle swarm optimization. Global optimization technique. J SICE 2008;47:459–465. (in Japanese)
3. Carlisle A, Dozier G. Adaptive particle swarm optimization to dynamic environments. Proc of Int Conf on Artificial Intelligence, p 429–433, 2000.
4. Eberhart RC, Yuhui S. Tracking and optimizing dynamic systems with particle swarms. Proc of Congress on Evolutionary Computation, p 94–100, 2001.
5. Carlisle A, Dozier G. Tracking changing extrema with adaptive particle swarm optimizer. Proc of Soft Computing, Multimedia Biomedicine, Image Processing and Financial Engineering, p 265–270, 2002.
6. Blackwell TM. Swarms in dynamic environments. LNCS 2004;2723:1–12.
7. Blackwell T, Branke J. Multi-swarms optimization in dynamic environments. LNCS 2004;3005:489–500.
8. Cui X, Potok TE. Distributed adaptive particle swarm optimizer in dynamic environment. Proc of IEEE Int Parallel and Distributed Processing Symposium, p 244–250, 2007.
9. Zhang X, Qin YDZ, Qin G, Lu J. A modified particle swarm optimizer for tracking dynamic system. LNCS 2005;3612:592–601.
10. Clerc M, Kennedy J. The particle swarm: Explosion, stability, and convergence in a multidimensional complex space. IEEE Trans EC 2002;6:58–73.
11. Mizoguchi K, Sakamoto T. Self-tuning decentralized controller design of web tension control system. Proc of EUROSIM Congress on Modeling and Simulation, 2010.
12. Doi M, Kamiya T, Mori Y. A study on robust asymptotic tracking performance for generalized minimum variance control. Trans IEE Japan 1999;119-C:1420–1426.

AUTHORS (from left to right)

Takeshi Nishida (member) received a bachelor's degree from Kyushu Institute of Technology in 1998, completed the doctoral program in 2002, and joined the faculty as a research associate, becoming an assistant professor in 2007. His research interests are outdoor mobile robots. He holds a D.Eng. degree, and is a member of RSJ, SICE, JNNS, IEICE, and other societies.

Tetsuzo Sakamoto (senior member) completed the doctoral program at Kyushu University in 1984 and joined the faculty as a research associate. He became a research associate at Kyushu Institute of Technology in 1985, then a lecturer and associate professor; professor since 2002. His research interests are analysis and control of linear drives, maglev, and web tension systems. He holds a D.Eng. degree, and is a member of SICE and other societies.