Send Orders for Reprints to reprints@benthamscience.net

The Open Automation and Control Systems Journal, **2014,** *6,* **69-76**

Open Access

Robot Path Planning Based on Chaos Concise Differential Evolution and RFNN Control

Xiaosheng Wang[1,*] and Gaochao Xu[2]

[1]*Information & Technology School, Shandong Women's University, Jinan, China*

[2]*Computer Science & Technology School, Jilin University, Changchun, China*

Abstract: In order to reduce the memory footprint and energy consumption of embedded microcontroller in mobile robot, the concise differential evolution algorithm based on chaotic local search (CDE-CLS) is proposed for online optimization of recurrent fuzzy neural network (RFNN) controller in robot path planning so that the robot can be adaptive real-time obstacle avoidance. The CDE-CLS algorithm reduces the memory footprint of the controller using virtual population and increases the ability to explore help to fast convergence introducing a simple and efficient chaotic local fine search and inhibit premature convergence perturbing the virtual population. Contrast tests on the typical Benchmark functions verify the global convergence and stability of the algorithm comparing with other concise evolutionary algorithm. Finally, the simulation result on the robot path planning controller shows the effectiveness of the proposed method.

Keywords: Chaotic local search, concise differential evolution, online optimization, recurrent fuzzy neural network (rfnn), robot path planning, virtual population.

1. INTRODUCTION

Online optimization of controller in mobile robot path planning control system also needs to constantly enrich the optimization method and introduce the new algorithm. Differential Evolution (DE) algorithm is an optimization algorithm with real-value encoding and random search in continuous space [1], which is simple, robust, fast convergence, and has the better performance than the particle swarm and other evolutionary algorithms [2], and can be applied to online optimization. However, in practical applications, the DE algorithm has also been exposed to many shortcomings, such as it is easily trapped into local optimum, slow convergence in the late and a certain blind search on solving difficult high-dimensional and multi-peak complex optimization problems, resulting in poor feasibility in practical engineering applications of the large scale and highly nonlinear and real-time requirement [3]. Therefore, some scholars had made improvements in DE. In [4] a novel mutation was used, i.e. using the history of the target individual optimal solution to guide the new population search direction. In [5] the binomial crossover strategy was used. The above methods can accelerate the convergence rate and reduce precocious probability, but stability of the algorithm needs to be strengthened. Additionally, the DE and other evolutionary algorithms, such as Genetic Algorithms (GA), Particle Swarm Optimization (PSO) are population-based meta-heuristic methods and generally require a general purpose computer to perform the optimization, but directly executing the optimization algorithm is not feasible for embedded controller with limited memory and power supply etc, as it consumes huge hardware resources for realization of the algorithm. For the embedded control engineering, such as a small robot control or a manipulator control and an intelligent vehicle control etc., it is not possible to use a high-power general-purpose computing device due to the limitations of the cost and the size. To solve these problems, some concise evolutionary algorithms have been proposed. In [6] a real-value compact genetic algorithm (RCGA) was proposed. The RCGA is based on the idea of virtual population to reduce the memory footprint, but it slows the search speed relatively and has the phenomenon of premature convergence. In [7] a memetic compact differential evolution (MCDE) algorithm was proposed to implement online optimization of a three-joint manipulator controller. The MCDE references the RCGA and combines an additional local search algorithm to improve the optimization efficiency, but it may increase the probability of the premature convergence during operation. In order to solve the above problems, this paper proposes a new modified concise differential evolution algorithm, namely a Concise DE based Chaotic Local Search (CDE-CLS). The CDE-CLS employs an intensively exploitative evolutionary framework based on a DE logic aided by a concise efficient fine exploitative chaotic local search algorithm to improve the algorithm's ability to help help explore fast convergence, and a perturbation mechanism was applied to the virtual populations to help global convergence. In addition, the CDE-CLS algorithm is run with less memory use due to the virtual populations. Thus, the CDE-CLS algorithmis not only concise and efficient but requires much less memory devices and is particularly suitable for embedded controller online

*Address correspondence to this author at the Information & Technology School, Shandong Women's University, Jinan, China;
Tel: +86 (0531)86526783; Fax: +86 (0531)86526786;
E-mail: wxs1010@126.com

1874-4443/14 2014 Bentham Open

optimization. The numerical results and simulation on recurrent fuzzy neural network controller optimization for robot path planning show effectiveness of the proposed algorithm.

2. CONCISE DIFFERENTIAL EVOLUTION BASED CHAOTIC LOCAL SEARCH (CDE-CLS) ALGORITHM

Assume that the fitness function is f(x) for optimization problems, the optimization problem can be described as follows:

$$\min\ f(x) \tag{1}$$

where x is a vector of n design variables in the decision space D, that is, $x=[x_1, x_2, ..., x_n]$. Without the loss of generality, it is assumed that the parameters are standardized so that each search interval is in [-1,1].

2.1. CDE-CLS Algorithm Processes

Step 1. By the Gaussian probability distribution function (PDF) within U[-1,1] an $n\times 2$ probability vector VP is generated:

$$VP(t)=[\delta(t)\quad \sigma(t)] \tag{2}$$

where, VP is an $n\times 2$ matrix, n is the dimension of the solution space. $\delta(t)=[\delta_1(t), \delta_2(t),\ldots, \delta_n(t)]^T$ and $\sigma(t)=[\sigma_1(t), \sigma_2(t),\ldots, \sigma_n(t)]^T$ are mean vector and standard deviation vector of a PDF within the interval [-1,1] for each design variable I, respectively. The height of the PDF has been standardized so as to maintain their area which is equal to 1. t is the generation.

Step 2. (Initialization) t=0. For each design variable i, set $\delta_i(0)=0$, $\sigma_i(0)=\varepsilon$, i=1,2,$\cdots$, n, ε=10.

Step 3. An individual is sampled from VP, and marked as elite x_{elite}.

Step 4. (Chaos local search) A random number rand (0,1) is Generated, if rand (0,1)<Ls then Step (a) to perform chaos local traversal search will be performed, else it would go to Step 5.

Step (a). A n-dimensional vector is generated randomly $V_0=[V_{0_1}, V_{0_2},\cdots, V_{0_n}]$, where $V_{0_k}\in(0,1)$ and $V_{0_k}\notin[0.25,0.5,0.75]$, ($k$=1,2,…,$n$). Set μ=4 and let $V_{(q+1)_k}=\mu V_{q_k}(1-V_{q_k})$, q=0, 1,..., Q-1, thereby Q chaos vector are generated iteratively.

Step (b). Let $X_{q+1}=x_{elite}+\beta V_{q+1}$, where β is the chaos adjustment parameters and its value is determined by the Eq. (3).

$$\beta=\begin{cases}1, & if\quad rand(0,1)\geq 0.5\\ -1, & otherwise\end{cases} \tag{3}$$

The β is introduced so as to ensure the best individual traverse towards the positive and negative directions.

Step (c). The fitness values of X_{q+1} are computed to find the local optimal individual X_{opt}, and the fitness of the X_{opt} is compared with fitness value of the x_{elite}, if the X_{opt} is better than the x_{elite}, then the x_{elite} is replaced with X_{opt}, otherwise it cannot be replaced.

Step (d). Go to Step 8.

Step 5. (Mutation operating) 3 individuals are generated by means of VP: x_{k1}, x_{k2} and x_{k3}. The mutation is performed according to Eq.(4), then a temporary offspring x'_{off} is generated:

$$x'_{off}=x_{k3}+F(x_{k1}-x_{k2}) \tag{4}$$

where F$\in$ [0,1] is a scale factor.

Step 6. (Crossover operating) $x'_{elite}=x_{elite}$. Each gene of the individual x'_{off} is exchanged with the corresponding gene of x'_{elite} with a uniform probability and then final offspring x_{off} is generated:

$$x_{off}[j]=\begin{cases}x'_{off}[j] & if\ (rand(0,1))<Cr\\ x'_{elite}[j] & otherwise\end{cases} \tag{5}$$

where j =1,2, ..., n is the index of the gene under checking, Cr is the constant value namely crossover rate. This crossover strategy is a binomial crossover (bin). Since the binomial crossover strategy is usually better than the exponential crossover in performance [4].

Step 7. (Selection operating) The fitness value of final offspring x_{off} is computed and compared with that associated with x_{elite}. If $f(x_{off})<f(x_{elite})$ then the elite is replaced with x_{off}, else the elite x_{elite} is preserved.

Step 8. (The VP is updated) The VP is updated by Eqs. (6), and (7) [6, 7].

$$\delta(t+1)=\delta(t)+\frac{b-l}{Np}+\rho\cdot\lambda\cdot(rand(0,1)-0.5) \tag{6}$$

$$(\sigma(t+1))^2=(\delta(t))^2-(\delta(t+1))^2+\frac{b^2-l^2}{Np}+\rho\cdot\lambda\cdot rand(0,1) \tag{7}$$

Where, $$\rho=\begin{cases}1, & if\ rand(0,1)<Mp\\ 0, & otherwise\end{cases} \tag{8}$$

The Np is the virtual population size, b is the individual of the current winner and l is the current loser among elite and newly generated offspring. In Eqs. (6) and (7), the last term is an additional perturbation, where λ is the maximum amplitude of perturbation. Mp is a constant representing perturbation probability.

Step 9. $t=t+1$. If the termination condition is met (such as to achieve the maximum generation), then the algorithm is ended and the obtained elite solution is outputted, otherwise the parameters Ls is updated by Eq. (9), through Step 4.

$$Ls(t)=Ls^{\min}+\frac{Ls^{\max}-Ls^{\min}}{t\max}\times t \tag{9}$$

where t is the current generation, $t$max is the maximum generation of the algorithm, Ls^{min} and Ls^{max} are the maximum and minimum values of Ls, respectively.

Table 1. Fitness value and standard deviation comparison for three concise algorithms.

Function	RCGA	MCDE	CDE-CLS
	Fitness Value (Standard Deviation)	Fitness Value (Standard Deviation)	Fitness Value (Standard Deviation)
$f_1(x)$	1.91E+04 (9.61E+03)	6.53E-25 (8.45E-25)	2..62E-02 (2.17E-02)
$f_2(x)$	2.01E+09 (2.23E+09)	1.48E+04 (6.91E+04)	5.64E-02 (3.67E-02)
$f_3(x)$	1.86E+01 (4.12E-01)	1.87E+00 (1.73E+00)	1.65E+00 (4.65E-01)
$f_4(x)$	2.31E-03 (4.23E-03)	6.87E-03 (1.90E-02)	5.45E-03 (1.81E-03)
$f_5(x)$	2.134E+02 (2.81E+01)	6.49E+01 (1.42E+01)	1.64E+01 (3.21E+00)

2.2. CDE-CLS Algorithm Features

It is through parameters *Ls* that the chaotic local search (CLS) algorithm is determined whether it needs to be executed to replace the normal mutation and crossover. After each iterate, the CLS performs local fine search near the elite individual for *Q* times, if the found individual is better than the previous elite, then the elite is replaced. The usual chaotic model is one-dimensional Logistic map [8]. As can be seen from Eq. (9) that CLS executed probability *Ls* increases with the iteration number, that is, at a later stage, the CDE-CLS algorithm will have more opportunities to execute the CLS algorithm to deeply develop viable solutions and to prevent from premature convergence.

Another important feature of the proposed algorithm CDE-CLS is that, due to its simplicity and low memory requirement (only four memory use, i.e. two for the VP, one for x_{elite}, and one for the offspring) using virtual population compared with population-based version, it can easily be implemented into embedded hardware characterized by a limited memory and power, such as PIC32 microcontroller with 8-32KB RAM. The optimization of a population-based algorithm is likely to overflow in the computational power or the available memory resources.

The balance between global and local search in the proposed CDE-CLS algorithm is obtained by using perturbation in VP update rule. The perturbation mechanism then inhibits the algorithmic premature convergence and force the algorithm to search elsewhere in the decision space, possibly exploring new promising solutions.

3. NUMERICAL RESULTS

In order to verify the effectiveness of the proposed algorithm CDE-CLS, it is tested on five typical Benchmark functions, and compared with other concise algorithms RCGA and MCDE, lasting elitist strategy is used in the RCGA, and lasting elite, DE/rand/1 mutation and the binomial crossover strategy and other parameters set same as [7]. These test functions selected are as follows: $f_1(x)$ is a Sphere function, $f_2(x)$ is a Rosenbrock function, $f_3(x)$ is a Ackley's function, $f_4(x)$ is a Griewank function and $f_5(x)$ is a Rastring function, each function mathematics expression are given in [9, 10]. For each function, $n = 30$.

For three algorithms, the termination condition of each single run is fixed as $5000 \times n = 150{,}000$ fitness evaluations. The CDE-CLS algorithm parameters are: $Q = 10$, $F = 0.5$, $Cr = 0.7$, $Np = 2 * n = 60$, $\lambda = 0.2$, the probabilities of activating chaotic local search *Ls* and perturbing the virtual population *Mp* have been set as 0.002 and 0.001, respectively. Table **1** shows the average fitness and standard deviation value detected by each algorithm over 30 times runs. The best results are highlighted in bold face.

Numerical results showed that the algorithm CDE-CLS is better than RCGA and MCDE on average. Simultaneously, standard differential results showed that CDE-CLS algorithm is more stable.

For example of $f_4(x)$, the problem of low-dimensional and $f_5(x)$, high dimensional, Fig. (**1**) and Fig. (**2**) showed the convergence rate of CDE-CLS algorithm is close to the MCDE, and clearly better than RCGA. For solving $f_5(x)$ high-dimensional problem, RCGA and MCDE appear at premature convergence, while CDE-CLS can converge to better solutions. Therefore, CDE-CLS algorithm not only can guarantee the convergence rate, but also improve the optimization accuracy and stability.

4. MOBILE ROBOT PATH PLANNING BASED ON CDE-CLS ALGORITHM AND RECURRENT FUZZY NEURAL NETWORK CONTROL

4.1. Mobile Robot Kinematics Model

Assume that the robot's current coordinate is (x_r, y_r) and the goal point coordinate is (x_g, y_g). As shown in Fig. (**3**), E

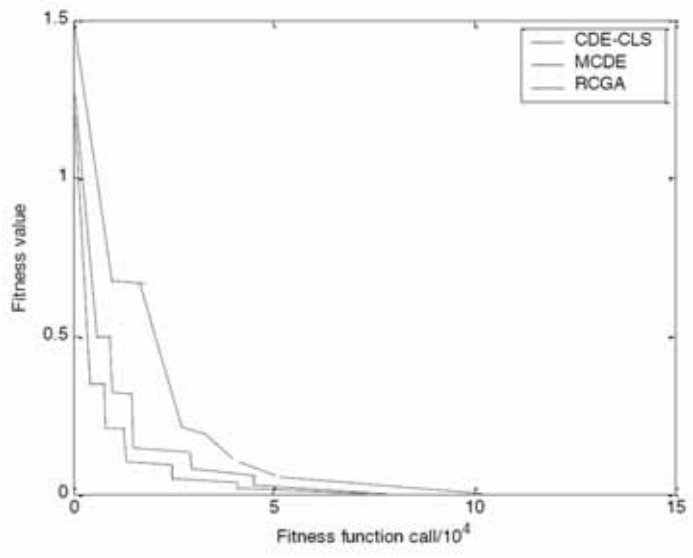


Fig. (1). Optimization curve of the f_4 (x).

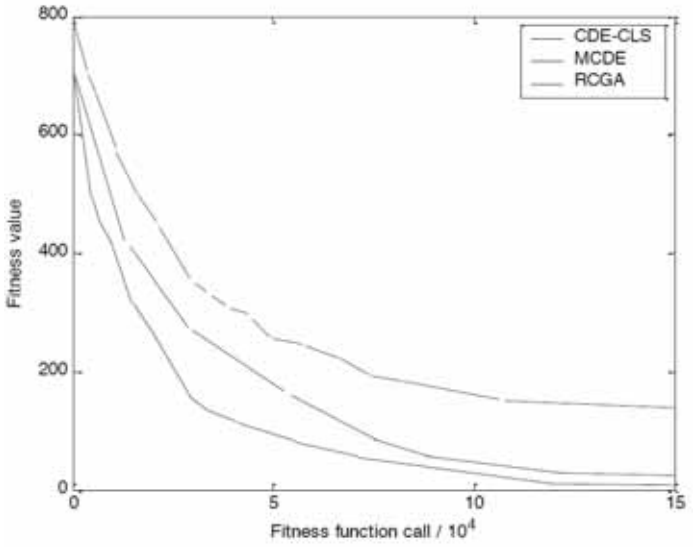


Fig. (2). Optimization curve of the f_5 (x).

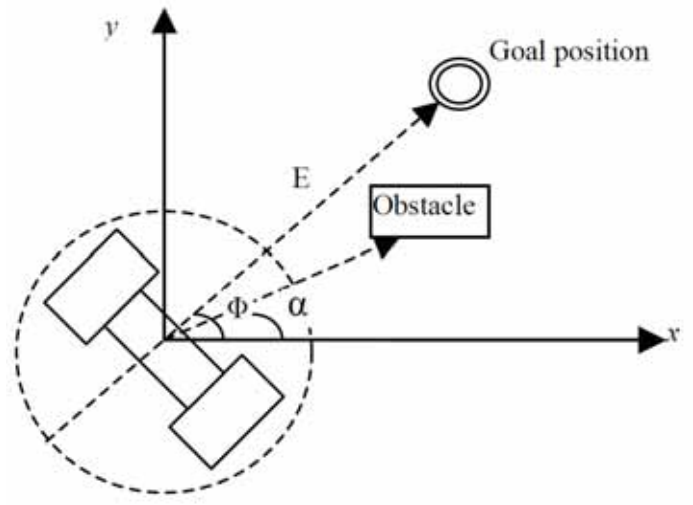


Fig. (3). Mobile robot kinematics model diagram.

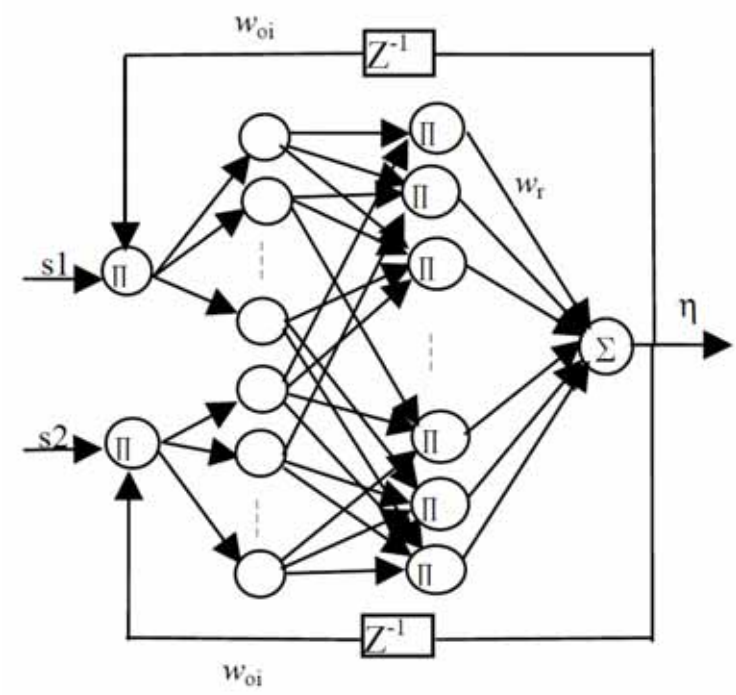


Fig. (4). Fuzzy recurrent neural network structure.

is a vector which the robot (x_τ, y_τ) points to goal (x_g, y_g). The modulus of the E and its direction angles are:

$$E_\tau = \sqrt{(x_\tau - x_g)^2 + (y_\tau - y_g)^2} \tag{10}$$

$$\alpha_\tau = \arctan\frac{y_g - y_\tau}{x_g - x_\tau} \tag{11}$$

α_τ is the angle between the current robot and goal point, which is constantly revised according to current position of the robot, always pointing to the goal position. The subscript τ is time.

In reactive navigation [11], the mobile robot plans local path based on sensor information. In the process of the robot moving, if there are no obstacles around the robot, the robot moves toward the goal point at angle α. If there is an obstacle ahead, a perturbation η needs to be artificially added, thereby establishing the following equation:

$$\Phi_\tau = \alpha_\tau + m\eta_\tau \tag{12}$$

where, Φ_τ is pre aiming direction of the robot, m is a proportionality coefficient, η_τ is a perturbation angle added and its value will be determined by the adaptive fuzzy recurrent neural network (RFNN) controller according to the robot current environment in [-30°, 30°], the negative number represents an increased perturbation amount in clockwise direction, positive number represents an increased amount in counterclockwise direction for the robot. The robot is closer to the obstacle, the absolute value of η_τ is greater. When there are no obstructions ahead i.e. η_τ=0, the robot moves toward the goal, while there is an obstacle ahead i.e. $\eta_\tau \neq 0$, the robot moves forward according to the goal direction after the offset of the additional perturbation.

4.2. Fuzzy Recurrent Neural Network Controller

The fuzzy recurrent neural network (RFNN) controller real-time outputs perturbation angle and online adjustments pre-aiming direction of the mobile robot so that the mobile robot can trend a collision free goal. Designed RFNN is a four-layer structure shown in Fig. (**4**). The two-input & one-output structure is selected to reduce system complexity, inputted data are measured obstacle distance around left and right sides by the sensor of the robot (minimum measured value should be taken among them respectively), the output is a disturbance angle in [-30°, 30°] based on the current robot environment. Each hidden layer node represents a fuzzy subset, which is divided into five fuzzy subsets: {NB, NS, ZE, PS, PB}; If the two feedback connections are removed, the structure of the network becomes a feed-forward Fuzzy Neural Network (FNN).

The first layer is the input layer, its output is expressed as:

$$y_i^1(k) = \prod s_i(k) w_{oi} \eta(k-1) \qquad i = 1,2 \tag{13}$$

where, $y_i^1(k)$ is the output of the layer, w_{oi} is recurrent weight from output to input layer, k is the number of iterations.

The purpose to introduce recurrent layer is to describe the nonlinear dynamic behavior of the system through additional state feedback neurons. The recurrent node memories output the value of the previous time, equivalent to a step of delay operator. It can be seen that the recurrent layer can store past information of the system to make the network increase the processing capability of dynamic information.

The second layer is a fuzzification layer, has 10 nodes, each node represents a membership function. Here we use the Gaussian function as the membership function, the input-output relationship is:

$$u_{ij}^2(k) = -\frac{(y_i^1(k) - a_{ij})^2}{(b_{ij})^2}, \quad y_{ij}^2(k) = \exp(u_{ij}^2(k))$$

i=1,2 ; j=1,2,…,5 (14)

where a_{ij} and b_{ij} represent the central value and width value of the Gaussian function respectively.

The third layer is fuzzy reasoning layer. The number of nodes is equal to the number of rules, up to 25 (i.e. 5^2) nodes. The input-output relationship is:

$$u_j^3(k) = \prod_{i=1}^{2} y_{ij}^2(k), \quad y_r^3(k) = u_r^3(k) \tag{15}$$

where, j =1,2, ...,5; r =1,2,...,25 or 11 (experience value)

The fourth layer as output layer completes defuzzification and generates the network output.

$$u^4(k) = \sum_{r=1}^{N} w_r y_r^3(k),$$
$$\eta(k) = y^4(k) = \frac{u^4(k)}{\sum_{r=1}^{N} y_r^3(k)} \tag{16}$$

where, w_r is a weight value of the r-th fuzzy rule acting on output node.

The system uses the proposed CDE-CLS algorithm online training RFNN. The optimized RFNN parameters includes: the central values a_{ij} and width b_{ij} of the membership function, network recurrent connection weights w_{oi}, the weights of the output layer w_r, i = 1,2; j = 1,2, ..., 5; r =1 , 2, ..., 25. These parameters as a gene sequence constitute one chromosome. Due to a total of 47 parameters to be optimized, the dimension is 47 for each individual, and real-coded.

Fitness function for evaluating individual x is as follows:

$$f(x_i) = \frac{1}{T}\sum_{i=1}^{T}(\eta_i^d(t) - \eta_i(t))^2 \tag{17}$$

where, T is the time points, η_i^d and η_i are desired output and the actual output on RFNN. The optimization goal is minimizing $f(x_i)$, when $f(x_i) = 0$, the output value and the desired value are consistent.

4.3. Robot Path Planning Simulation

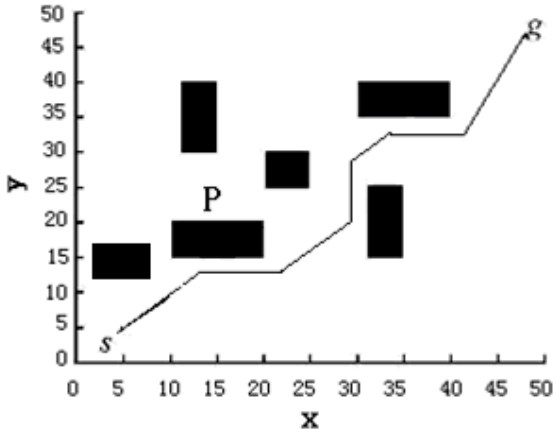
Before simulation is performed, the input parameters of the RFNN should be normalized to guarantee the network convergence. Computer simulation is done to test performance of proposed algorithm for robot path planning in Matlab 7.0. The simulation results are shown in Fig. (**5**), where, the box represents obstacle. The starting position of the robot movement is s and the goal position is g. Set $\Phi s=\alpha_s= 45°$, m=6. The CDE-CLS algorithm has been run with the same parameter setting specified in section 3. The CDE-CLS algorithm runs independently twenty-five times. After CDE-CLS algorithm performs optimization to RFNN, the optimal solution x = {a, b, w} is generated as the RFNN's optimization parameters. The Optimal RFNN controller controls the robot movement from the starting point to the goal point avoiding obstacles in real time.

The RFNN network training error is shown in Fig. (**6**), after 10000 fitness evaluations the error value is reduced to 0.062° or less.

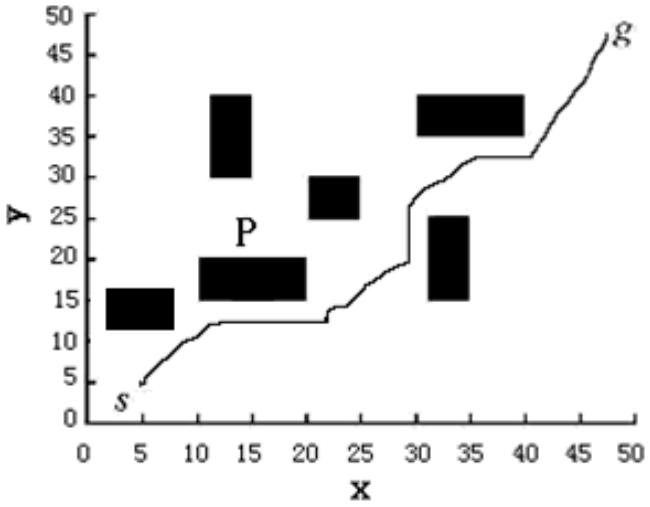
Fig. (**5b**) shows that the robot path is not smooth due to directly using RFNN network with the BP learning algorithm (without the CDE-CLS algorithm training), and leaned parameters of RFNN are imprecise.

Fig. (**5c**) shows that when the robot is close to the obstacle and needs to avoid the obstacle, its motion direction should be deviated from the obstacle and beyond the measurement range of the sensor, the robot then moves toward the obstacle because the controller have the ability to store past information, thus the oscillation is generated in the process of obstacle avoidance. The results showed that the feedforward fuzzy neural network controller can not accurately describe the dynamic performance of the system.

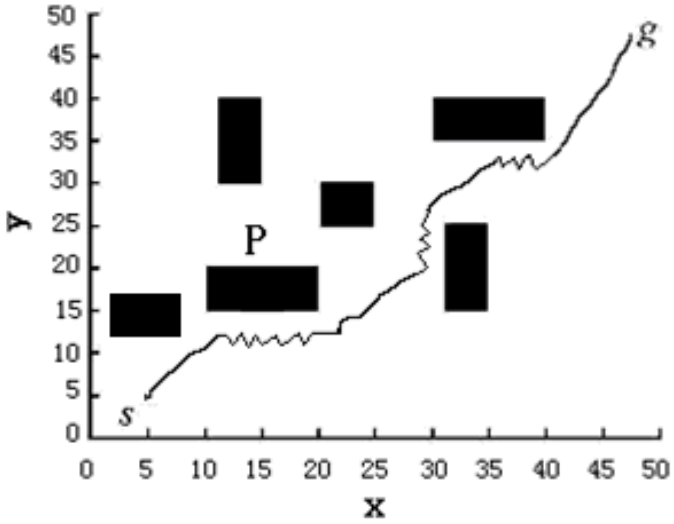
In order to test the robot controller adaptability to the environment and adaptive ability to unknown environment, we change the robot local environment. The experimental results showed that a behavior which is well evolved well using proposed evolutionary algorithms in an environment has better adaptability to unknown environment, as shown in Fig. (**5d**). The robot's starting point and the goal point have been changed, a new obstacle Q has been added, after the robot has run 15 steps, obstruction P begins to run from left to right at the same speed as the robot, the robot can avoid real -time obstacle by controller evolved.



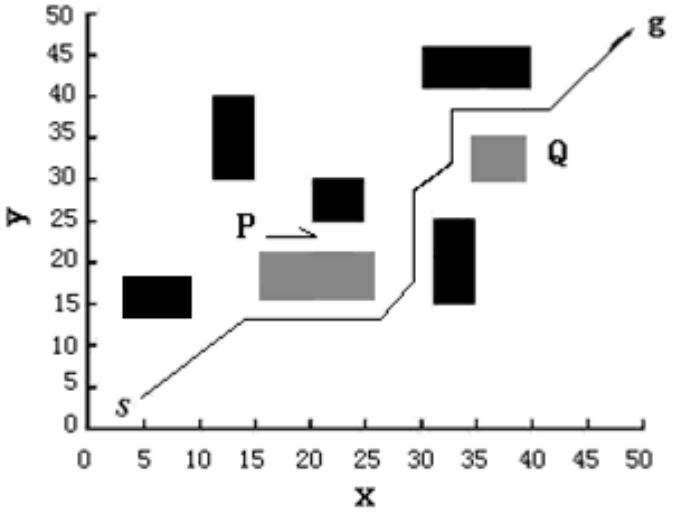
(a) Robot path by CDE-CLS & RFNN controller



(b) Robot path by only RFNN controller



(c) Robot path by CDE-CLS & feedforward fuzzy neural network (FNN) controller



(d) Robot path by CDE-CLS & RFNN controller in an unknown environment

Fig. (5). The simulation results.

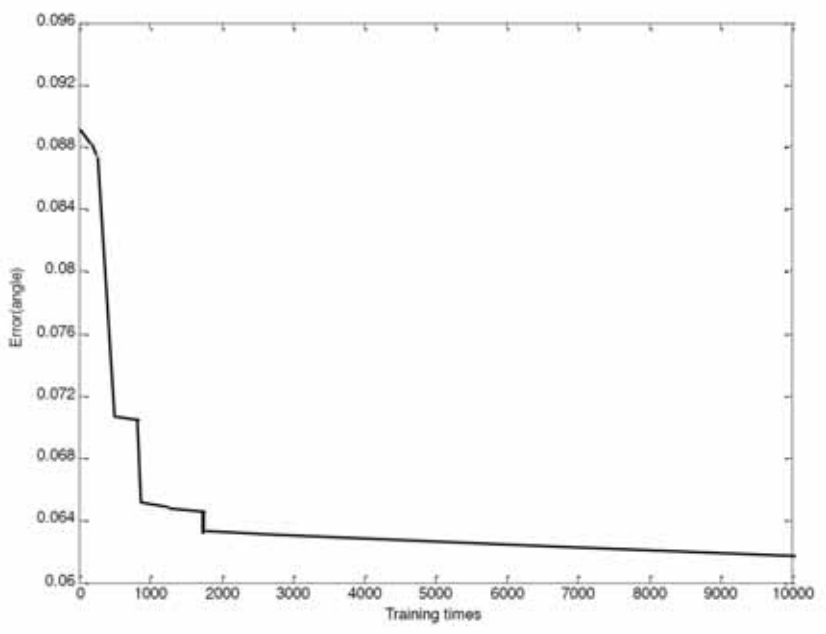


Fig. (6). Neural network training error.

CONCLUSION

This paper proposes a differential evolution algorithm based on virtual population and chaotic local search, the algorithm has a concise structure, fast convergence, high precision of optimization, and less demand for memory, it is more suitable for online optimization within the embedded controller with limited resources. Additionally, chaotic local search can assist the DE to change the direction of the search to improve the ability to global converge. The simulation results on controlling the robot path planning based on RFNN controller show the effectiveness of the proposed CDE-CLS algorithm.

CONFLICT OF INTEREST

The authors confirm that this article content has no conflicts of interest.

ACKNOWLEDGEMENTS

This research is supported by Symbolic Computation and Knowledge Engineering of Jilin University Key Laboratory of Ministry of Education (93K-17-2010-K05), and the Natural Science Foundation of Shandong Province (ZR2013-FL025).

REFERENCES

[1] R. Storn and K. Price, "Differential evolution-a simple and efficient heuristic for global optimization over continuous spaces," *Journal of Global Optimization*, vol. 11, no. 4, pp. 341-359, 1997.

[2] J. Vesterstrom and R. Thomsen, "A comparative study of differential evolution, particle swarm optimization and evolutionary algorithms on numerical benchmark problems," In: *Proceedings of the 2004 Congress on Evolutionary Computation*, 2004, pp. 1980-1987.

[3] S. Das and A. Abraham. "Differential evolution using a neighborhood-based mutation operator," *IEEE Trans on Evolutionary Computation*, vol. 13, no.3, pp. 526-553, 2009.

[4] X.J. Bi, G.A. Liu and J. Xiao, "Dynamic adaptive differential evolution based on novel mutation strategy," *Journal of Computer Research and Development*, vol. 49, no. 6, pp. 1288-1297, 2012.

[5] J. Lampinen and I. Zelinka, "On stagnation of the differential evolution algorithm," In: *Proceedings of 6th International Mendel Conference on Soft Computing*, 2000, pp. 76-83.

[6] E. Mininno, F. Cupertino, and D. Naso, "Real-value compact genetic algorithms for embedded microcontroller optimization," *IEEE Trans on Evolutionary Computation*, vol. 12, no. 2, pp. 203-219, 2008.

[7] F. Neri and E. Mininno, "Memetic compact differential evolution for cartesian robot control," *IEEE Computational Intelligence Magazine*, vol. 5, no. 2, pp. 54-65, 2010.

[8] R. Caponetto, L. Fortuna and S. Fazzino, "Chaotic sequences to improve the performance of evolutionary algorithms," *IEEE Trans on Evolutionary Computation*, vol. 7, no. 3, pp. 289-304, 2003.

[9] A.K. Qin, V.L. Huang, P.N. Suganthan, "Differential evolution algorithm with strategy adaptation for global numerical optimization," *IEEE Trans on Evolutionary Computation*, vol. 13, pp. 398-417.

[10] J. Vesterstrøm, R. Thomsen, "A comparative study of differential evolution, particle swarm optimization and evolutionary algorithms on numerical benchmark problems," In: *Proceedings of the IEEE Congress on Evolutionary Computation*, 2004, vol. 3, pp. 1980-1987.

[11] E. Istvån and H. Gåbor, "Artificial neural network based mobile robot navigation," In: *Proceedings of 6th IEEE International Symposium on Intelligent Signal Processing*, 2009, pp. 241-246.

Received: August 13, 2013 Revised: August 28, 2013 Accepted: August 28, 2013