Machine Intelligence & Robotic Control, Vol. 6, No. 2, 39–59 (2004)

Paper

Neural Network-Based Approaches vs. Potential Field Approach for Solving Navigation Problems of a Car-Like Robot*

Nirmal Baran Hui† and Dilip Kumar Pratihar†

Abstract: In the present work, neural network (NN)-based approaches are compared to a potential field approach for tackling navigation problems of a car-like robot in dynamic environment. Two different approaches are developed to determine an optimal NN. In the first approach, connecting weights of the NN are tuned by using a back-propagation (BP) learning algorithm and a systematic study is conducted to obtain optimal number of hidden neurons, whereas a genetic algorithm (GA) is used for designing a suitable NN automatically, in the second approach. Results of the above approaches are compared to those of a potential field method, through computer simulations, for solving the navigation problems of a car-like robot. The combined GA-NN approach is found to perform better than both the BPNN approach and potential field approach, for most of the randomly-generated test scenarios. CPU times of all the approaches have come out to be low and thus, these are suitable for online implementations.

Keywords: Neural Network, Back-Propagation, Genetic Algorithm, Potential Field Method, Car-Like Robot, Navigation

List of Symbols

α	Momentum constant
η	Learning rate
$\phi_{pj}(\cdot)$	Activation function associated with jth neuron lying on pth layer
θ_1	Deviation of the robot
ζ_{att}	Positive scaling factor for attractive potential
ζ_{rep}	Positive scaling factor for repulsive potential
a	Tangential acceleration of the robot, m/sec^2
b_{pj}	Bias applied to jth neuron lying on the pth layer
C_p	Constant of the pth layer activation function
d_{goal}	Distance between the robot and the goal, m
$d_{\min}$	Minimum distance required by the robot to reach the goal with zero velocity
$E_{pk}^{ms}(t)$	Half squared error at a distance step s of a training case m at tth iteration corresponding to kth neuron lying on the pth layer
$F(X)$	Potential force function
i	A neuron lying on first layer
$I_{pj}(t)$	Input of jth neuron lying on pth layer at iteration t
j	A neuron lying on hidden layer
J	Maximum number of neurons lying in hidden layer
k	A neuron lying on output layer
M	Maximum number of training scenarios considered during training
n	A particular GA-string
$O_{pj}(t)$	Output of jth neuron lying on pth layer at iteration t
S_m	Maximum number of distance steps taken by the robot in mth training scenario
T	Total traveling time, seconds
$T_{pk}(t)$	Desired response of kth output neuron lying on layer p at iteration t
$v_{ij}(t)$	Connecting weights between ith input neuron and jth hidden neuron at iteration t
$w_{jk}(t)$	Connecting weights between jth hidden neuron and kth output neuron at iteration t

List of Abbreviations

BPNN	Back-propagation neural network
CPU	Central processing unit
CG	Center of gravity
GA	Genetic algorithm
NN	Neural network
SOM	Self organizing map

1. Introduction

THERE is a growing interest in developing intelligent systems for mobile robots, to meet their increasing demand in manufacturing units, nuclear power plants, space applications and others. An intelligent robot should be able to take the decisions online, as the situation demands. To ensure this, a robot must be intelligent enough to behave quickly and flexibly. To meet these requirements, a robot must have a proper motion planning scheme. Motion planning of a car-like robot is a complicated task, due to the fact that its both nonholonomic and dynamic constraints are to be satisfied during navigation [1]. The problem becomes more difficult, when the environment is dynamic. Both analytical and graph-based techniques had been used by various investigators [2], to solve navigation problems of mobile robots. In this connection, the work of Fiorini and Shiller [3], Wang *et al*. [4] are important to mention, in which obstacle avoidance problems of a robot in the presence of some moving obstacles were tackled. Although po-

* Received on October 25, 2005; accepted on February 14, 2006; printed in June, 2006.

† Department of Mechanical Engineering, Indian Institute of Technology, Kharagpur-721 302, India.
E-mail: nirmal@mech.iitkgp.ernet.in, dkpra@mech.iitkgp.ernet.in

tential field method [5] appears to be the most popular one, in this context, the performance of this method depends on the chosen potential functions and it may have the local minima problem also. Moreover, most of the conventional motion planning techniques are not suitable for online implementations, due to their inherent computational complexity and limitations. Interested readers may refer to [6] for a more detailed information. Realizing the fact that it is difficult to develop an adaptive motion planner of a robot through explicit design, researchers working in this field of research started thinking, whether a suitable motion planner can be evolved by using evolutionary techniques. For this purpose, the field of evolutionary robotics (ER) [7] has been emerged as a thrust area in robotic research. Although ER includes evolution of both fuzzy logic-based and neural network (NN)-based controllers for solving some real-world problems, most of the investigators working in this field of research, concentrate on evolution of feedforward NNs. This may be due to the fact that NNs exhibit some important characteristics, such as generalizability, robustness to noise, adaptability and learning capability. For a proper study of such characteristics, one may refer to [8]–[10]. Quite a few investigators tried to develop a suitable robot controller based on NNs. Some of their works are mentioned below.

Yang and Meng [11] proposed an NN-based approach for solving navigation problems of a point robot. Moreover, Gu and Hu [12] modeled the kinematics of a car-like robot by using a three-layered feedforward NN, whose connecting weights were optimized by following a steepest descent algorithm. However, the developed NN-model representing the nonlinear kinematics of the robot may fail, when the robot is subjected to some dynamic constraints. Floreano and Mondada [13], [14] studied the evolution of an NN-controller by using a genetic algorithm (GA). The evolution of the controller was carried out on a physical robot — Khepara. A few more researchers had also attempted to evolve the NN-controllers by using the GAs. In this connection, work of Miglino *et al.* [15], Nolfi and Parsi [16], Husbands *et al.* [17], Noguchi and Terao [18] are important to mention. However, all such methods were unable to yield the best result, as the architecture of the NN was not optimized. Moreover, they assumed a two-layered (consisting of input and output layers) architecture of the NN, which might not be optimal in any sense. Later on, Baluja [19], Meeden [20] introduced a hidden layer in the NN-structure for solving the similar kind of problems. But, they did not perform the structural optimization of the NN. Cliff *et al.* [21], Harvey *et al.* [10] evolved the collision-avoidance behavior and Lee [22] evolved behaviors of autonomous robot in a pursuit system using a combined NN and a GA. The performances of their methods widely depend on the selection of fitness function of the GA. Thus, an improper choice of fitness function may deteriorate the proper functioning of the NN. Recently, Pratihar [23] has carried out a comprehensive review on various aspects of evolutionary robotics. More recently, Nelson *et al.* [24] have used an NN-controller for competitive game playing with teams of mobile robots. In their approach, a GA has been utilized to determine the optimal weights of the NN-controller, while evolving a proper behavior — **win**. Heero *et al.* [25] proposed a path selection mechanism by using an NN. They trained the controller in a partially-unknown environment, but no attempt was made to optimize the architecture of the controller. Kondo [26] suggested a neuro-modulatory NN model, for the development of a peg-pushing behavior of a mobile robot. Although they studied the robustness of the network against the specific environmental perturbations during optimization, no attempt was made to evolve the structure of the NN. Capi and Doya [27] used a parallel GA, to optimize a recurrent neural controller. In their approach, a sequential mode of training was considered. Thus, the evolved behavior of the network was far from the true optimal. Hagras and Sobh [28] explored some important points related to training of the autonomous agents working in an unstructured environment. They pointed out that neural controllers evolved using a GA could perform well for the behavior control of autonomous robots. Yamada [29] developed an evolutionary behavior-based learning approach for modeling an action-based environment of a mobile robot. In his approach, self organizing map (SOM) was applied to classify the training environment without a teacher and optimization of the classified model of training environment was carried out by using a GA. However, the combined GA-SOM approach was found to yield solutions, which were locally tuned and it was seen to be a bit computationally expensive. Thus, there is a chance of further improvement of the NN-based controller.

The design of an NN includes the selection of input-output parameters of the process to be controlled, number of hidden layer(s), number of neurons present in the hidden layer(s) and the connectivity among the neurons of the successive layers. The performance of an NN widely depends on the synaptic weights at different layers and its architecture. Initially, some efforts were made by various investigators [13], [14], [17], to design a fixed architecture NN. However, a designer should have a proper knowledge of the problem, to design a suitable NN-architecture for solving it. Moreover, a proper learning is required, so that the controller behaves in an optimal sense. In this context, two major learning methods, such as supervised and reinforcement learning are available in the literature. In supervised learning, updating proceeds through a teacher by following different learning rules, like Hebbian learning rule [30], Widrow-Hoff learning rule [31], back-propagation (BP) algorithm [32], and others. The BP algorithm has gained the maximum popularity, due to its ease of use. But, the main drawback of supervised learning lies in the fact that the teacher needs to have enough knowledge of the search space, which is often difficult to obtain beforehand. To overcome this problem, Barto and Anandan [33] proposed an associative reward-penalty learning rule, in which both the maximization of reward probability and the minimization of penalty factors were carried out simultaneously. Being a reinforcement learning rule, it will not require any teacher during training.

Besides the overwhelming majority in applications, there exist quite a few difficulties of both the supervised and reinforcement learning paradigms, for solving complex real-world problems. All such problems are explored and the probable solution methods available in the literature are

mentioned below.

1. They are all typically the local search algorithms and achieve the best solution in the region of their starting point. Moreover, obtaining a global solution is often dependent on a proper choice of initial parameters and selection of a search technique, like simulated annealing (SA) [34], genetic programming (GP) [35], GAs [36], and others.
2. A priori selection of an NN architecture is very much tedious. It is due to the fact that a large structure of NN provides better generalizability but learning becomes insufficient, whereas small structure learns well but has a less generalizing capability. Moreover, improper selection of NN structure may lead to either over training or under training [37]. Yao and Liu [38], Ritchie *et al.* [39] used GP for this purpose. However, the application of GP is restricted due to the fact that it takes a large amount of time for finding a solution. GAs have been widely used by various investigators for developing a suitable architecture of a fully-connected NN. In this connection, work of Yen and Lu [40], Takahama and Sakai [41], Sun *et al.* [42], Son *et al.* [43], Kim *et al.* [44] are worth mentioning. Islam *et al.* [45], Kang and Isik [46] also used GAs to optimize a partially-connected NN. All these approaches applied network growing and/or pruning methods to optimize the structure of NN. In network growing approach, either a single or a group of hidden neurons are added to the existing network in successive iterations, whereas the same is deleted in network pruning approach. As it is very difficult to arrange the hidden neurons based on their performance supremacy, a designer is unable to understand the basis of network growing/pruning techniques for the structural optimization of NNs. Thus, there is a possibility that the optimized architecture may contain some inefficient/redundant hidden neurons and/or connectivity.
3. Learning rates and momentum constants must be guessed heuristically or through systematic tests to ensure better convergence. To solve this problem, Castillo *et al.* [47] utilized a GA to obtain the best set of learning parameters. Though this combined GA-BP approach increased the search speed of the BP algorithm, the local minima problem was still not absent.

Understanding all such drawbacks of traditional learning methods and available global search algorithms, Gupta and Sexton [48], Sexton and Gupta [49] demonstrated that for a wide variety of complex functions, the GAs are able to achieve superior solutions for NN optimization, compared to those obtained by BP algorithm. However, a GA is basically a fitness function-driven search tool, which is blind for any other aspect that is not explicitly considered on fitness function. Thus, it may provide an optimal architecture of NN that is large in size for better generalizability, but it will extend the training time and increases the computation. A few investigators have added a network complexity term also in the fitness function, to penalize a large NN by loosing some amount of its generalizing capability. Thus, there is still a need to develop a suitable technique of NN optimization, which will evolve a proper NN-architecture having a meaningful generalizing quality in a reasonable training time. Interested readers may refer to [50], for a more detailed discussion on GA-based optimization of NN.

An attempt is made in this study, to design and develop an optimal/near-optimal NN-based motion planner, for solving navigation problems of a car-like robot. Simultaneous optimization of both topology and connecting weights of the NN are carried out offline, by using two different approaches. In Approach 1, a BP learning algorithm is used to tune the synaptic weights and the hidden neurons are optimized through a systematic study. However, in Approach 2, a sequential binary-coded GA with uniform crossover is applied to evolve the optimal controller, automatically. The performances of these two approaches are tested through computer simulations and their results are compared with that of a potential field method (i.e., Approach 3), for solving navigation problems of a car-like robot in dynamic environment.

The rest of the text is organized as follows: In Section 2, the problem is stated and a possible solution technique based on NN is identified. The proposed three approaches are explained in Section 3. Results of computer simulations are presented and discussed in Section 4. In Section 5, the performances of the proposed methods are compared with those of some other methods used for solving the similar kind of problems. Finally, some concluding remarks are made in Section 6 and the scope for future work is indicated in Section 7.

2. Statement of the Problem and a Proposed Method of Solution

The dynamic motion planning problem of a car-like robot is tackled in the present study. The problem has been defined and a feasible method of solution is identified in this section, as discussed below.

2.1 Statement of the problem

A car-like robot has to find a collision-free, time-optimal path, while navigating among a few moving obstacles. It is to be noted that the obstacles are nothing but the moving objects (either regular or irregular shaped), whose boundaries have been represented by their corresponding bounding circles. Since the environment is dynamic, the robot has to move quickly, flexibly and cooperatively based on the collected information of the environment. Moreover, a car-like vehicle is subjected to both kinematic and dynamic constraints, which restrict its motion. Thus, it can only move backward or forward in a direction tangent to its trajectory and its turning radius is lower bounded because of the mathematical limits on the steering angle [51]. Hence, a car-like robot will have to find its collision-free path during navigation among moving obstacles, that will not only satisfy the kinematic and dynamic constraints of the robot, but also the planned path will have to be time-optimal one. Thus, the present problem can be treated as a constrained optimization problem. To reduce the complexity of the problem, all the moving obstacles are represented by their bounding circles and at a time, only one obstacle is treated to be critical by assuming that there is no overlap-

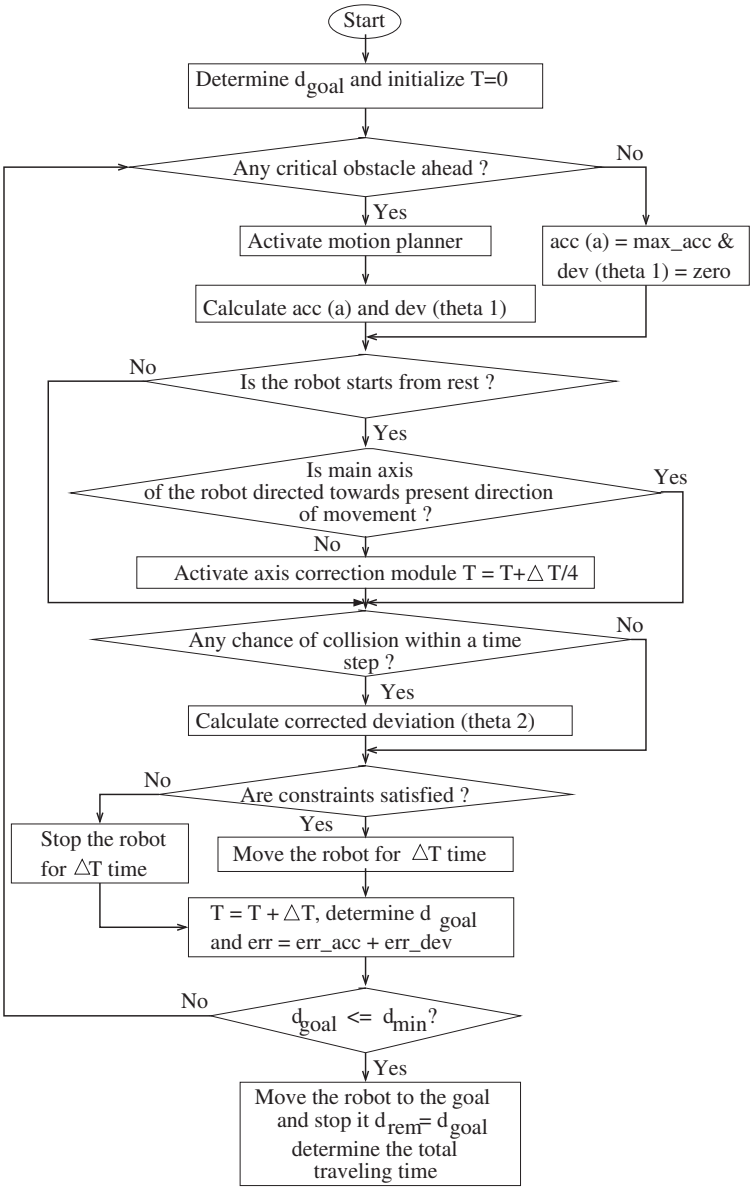


Fig. 1 Flowchart of the motion-planning scheme

ping between the two obstacles. Moreover, the wheels of the robot are considered to move due to pure rolling action only and back hitting of the robot by the obstacles is neglected, in the present work. It is also to be noted that the robot's motion is planned based on the predicted position ($P_{predicted}$) of the obstacles, which may be determined by linearly extrapolating from its present ($P_{present}$) and previous positions ($P_{previous}$) as given below:

$$P_{predicted} = P_{present} + (P_{present} - P_{previous}). \quad (1)$$

Our aim is to design a suitable controller of the robot that can plan and control its collision-free motion, while navigating among some moving obstacles in an optimal/near-optimal way, online.

2.2 Motion planning scheme

The developed motion planning scheme of the robot is explained with the help of **Fig. 1**. The total path (starting from a pre-defined position to a fixed goal) of the robot is assumed to be a collection of some small segments (either a straight one or a combination of straight and curved paths), each of which is traversed during a fixed time ΔT. The critical obstacle is identified by considering the relative velocity of the robot with respect to the obstacle and the direction of movement of the obstacle. If the robot finds any critical obstacle ahead of it, the motion planner is activated. Otherwise, the robot moves toward the goal in a straight path with a maximum possible velocity. The task of the motion planner is to determine the acceleration (a) and deviation (θ_1) of the robot based on the distance and angle inputs, to avoid collision with it. If required, the deviation as suggested by the motion planner, is to be corrected by using a collision-avoidance scheme [51], to ensure the robot's path to be completely collision-free. Moreover, if the robot starts its motion from rest, it may require an additional $\Delta T/4$ time to align its main axis towards its future direction of movement. Again, sometimes the robot's motion as provided by the motion planner may violate its kinematic and/or dynamic constraints. In such a situation, the robot is stopped for ΔT time at the present position itself.

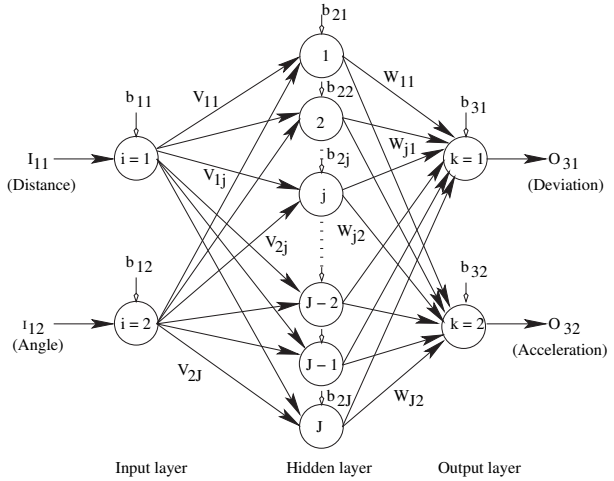


Fig. 2 A schematic diagram of the neural network structure

Thus, there will be an additional ΔT time per such occasion. This process will continue, until the robot reaches its destination and total traveling time T is calculated by adding all intermediate steps taken by the robot to reach it. It is important to mention that the last time step (T_{rem}) may not be a complete one and it depends on the distance left uncovered (d_{goal}) by the robot. If it (i.e., the goal distance d_{goal}) comes out to be less than or equal to a predefined minimum distance (d_{min}), it starts decelerating and stops at the goal. Our aim is to design a suitable adaptive motion planning algorithm, so that the robot will be able to reach its destination with the lowest possible traveling time by avoiding collision with the obstacles. Therefore, the present problem can be treated as a constrained traveling time (T) minimization problem as indicated below:

$$\text{Minimize } T = \sum \Delta T + \sum \frac{\Delta T}{4} + T_{rem} \tag{2}$$

subjected to the conditions,

- The path is collision-free,
- No constraint of the robot is violated.

It is important to mention that the minimum traveling time is possible to achieve, only when the robot traverses through a straight path with the maximum possible velocity of it. To ensure this, the present problem is solved indirectly by minimizing the error due to both deviation and acceleration outputs of the controller, simultaneously. Since the robot's path is planned based on the predicted position of the obstacles in the environment, there is a chance that the robot may collide with the most critical obstacle during its movement from the present position to the predicted position. Hence, a separate collision-avoidance scheme is developed to tackle such a situation, in which a geometry correction (in the direction opposite to the movement of the critical obstacle) is given to the deviation of the robot, as determined by the motion planner.

2.3 A possible solution using a neural network (NN)

NN has the capability of solving different complex real-world problems and it may also provide a feasible solution to the present problem. The steps involved in designing a suitable NN-controller are discussed below.

2.3.1 Selection of the topology of the NN: **Figure 2** shows the architectural graph of a three-layered feedforward NN with a single hidden layer. For simplicity, the structure of the NN is considered to be fully-connected, i.e., a neuron in any layer of the network is connected to all the neurons of the previous layer. In the first layer, two neurons representing the two inputs of the controller, such as *distance* of the robot from its most critical obstacle and their included *angle* with reference to the goal are considered, in the present work. There are two neurons at the output layer expressing two different outputs of the controller, namely *deviation* and *acceleration* of the robot required to avoid collisions with the moving obstacles and to reach the destination in minimum traveling time. The number of hidden layer neurons is varied in a reasonable range, to get the best result from the controller. For ease of implementation, we have assumed a fixed bias to each neuron of the architecture. A tangent hyperbolic function is taken as the activation function of the neurons in the NN-architecture.

2.3.2 Optimization of the synaptic weights: The performance of an NN-controller depends on its connecting (synaptic) weights and biases. Since the input and output layers contain two neurons each, there exist four connecting weights corresponding to one neuron lying on the hidden layer. Thus, for J number of hidden neurons, there are $4J$ synaptic weights, as it is a fully-connected network. These weights are to be optimized to get the best performance of the network and while doing that, a proper care should be taken, so that the weights do not come out of their ranges.

2.3.3 Tuning of the activation functions: The pattern of the input-output relationship is widely dependent on the activation functions of different layers. Several functions, such as linear, hard-limit, sigmoidal, hyperbolic and others have been used by various investigators [37]. A tangent hyperbolic function is utilized in the present study, as shown below:

$$\phi(I_{pj}) = \frac{e^{C_p I_{pj}} - e^{-C_p I_{pj}}}{e^{C_p I_{pj}} + e^{-C_p I_{pj}}} \tag{3}$$

where C_p is a constant to be optimized to get the better performance of the controller and I_{pj} denotes the input to a neuron j lying on the pth layer.

2.3.4 Optimization of the NN architecture: Generalizing power of an NN can be improved by optimizing both the number of hidden layers and the number of neurons present in a hidden layer. However, optimizing the number of hidden neurons is found to be more biologically plausible and has a greater effect on the performance of NN.

In the present work, an attempt is made to optimize both the architecture of an NN and its synaptic weights by following two different approaches. In Approach 1, the synaptic weights of the network are tuned by using a BP learning algorithm, whereas in Approach 2, the whole task of designing an NN-controller is given to a GA.

3. Proposed Algorithm

Navigation problem of a car-like robot is solved by using an NN, for the development of which two different approaches, namely BPNN approach (i.e., Approach 1) and a GA-NN approach (i.e., Approach 2) have been developed,

in this paper. Moreover, the said navigation problem has also been solved by utilizing a potential field approach (i.e., Approach 3). The performances of these three approaches have been compared, in the present work. All these approaches are discussed below, in detail.

3.1 Approach 1: Back-propagation neural network (BPNN)

The architecture of the NN is shown in Fig. 2. Signal-flow through the network progresses in the forward direction (from left to right) following a layer-by-layer basis. The formulation of this approach is explained below.

3.1.1 Layer 1: Two variables, namely distance (i.e., $I_{11}(t)$) and angle (i.e., $I_{12}(t)$) are fed as the inputs to the network and the outputs are calculated as follows:

$$O_{1i}(t) = \phi_{1i}(I_{1i}(t) + b_{1i}) \tag{4}$$

where $i = 1$ indicates the distance input and angle input is represented by $i = 2$, b_{1i} are the bias values.

3.1.2 Layer 2: Let us consider that there exists a total of J number of hidden neurons at this layer. The input to a neuron j lying on this layer may be determined as follows:

$$I_{2j}(t) = \sum_{i=1}^{2} (O_{1i}(t) \times v_{ij}(t)) \tag{5}$$

where $j = 1, 2, \cdots, J$. Therefore, the output of jth neuron at iteration t is calculated as shown below:

$$O_{2j}(t) = \phi_{2j}(I_{2j}(t) + b_{2j}) \tag{6}$$

where b_{2j} are the bias values.

3.1.3 Layer 3: The input to a neuron k lying on the third layer is calculated as follows:

$$I_{3k}(t) = \sum_{j=1}^{J} (O_{2j}(t) \times w_{jk}(t)) \tag{7}$$

where $k = 1$ represents the deviation output and $k = 2$ indicates the neuron corresponding to acceleration output. Hence, the output from this layer is obtained as follows:

$$O_{3k}(t) = \phi_{3k}(I_{3k}(t) + b_{3k}) \tag{8}$$

where b_{3k} represent the bias values.

3.1.4 Backward calculation: During training, the connecting weights between the input layer and hidden layer (i.e., $v_{ij}(t)$) and those between the hidden layer and output layer (i.e., $w_{jk}(t)$) are to be tuned to minimize the error in prediction. A BP algorithm along with a batch mode of training is followed for minimizing the mean squared error (MSE). Let us consider that there are M training scenarios and in a particular scenario (say, mth), the robot takes S_m number of steps to reach the goal, starting from a fixed initial position. Thus, at a particular time step s for mth training scenario, half squared error corresponding to a neuron k lying on the third layer may be calculated as follows:

$$E_{3k}^{ms}(t) = \frac{1}{2}\left[T_{3k}^{ms} - O_{3k}^{ms}(t)\right]^2 \tag{9}$$

where T_{3k}^{ms} and $O_{3k}^{ms}(t)$ indicate the target output and calculated output of the controller at a time step s for mth training scenario, respectively. Target output for deviation is considered to be equal to zero and that for acceleration is taken as the maximum permissible acceleration of the robot. The updated weights between the hidden and output layers are then calculated by using the following expression:

$$w_{jk}(t+1) = w_{jk}(t) - \eta \frac{\partial \overline{E}_{3k}(t)}{\partial w_{jk}(t)} + \alpha \Delta w_{jk}(t-1) \tag{10}$$

where η is learning rate and α is usually a positive number called momentum constant. Now, $\partial \overline{E}_{3k}(t)/\partial w_{jk}(t)$ is determined as follows:

$$\frac{\partial \overline{E}_{3k}(t)}{\partial w_{jk}(t)} = \frac{\partial \overline{E}_{3k}(t)}{\partial \overline{E}_{3k}^{m}(t)} \frac{\partial \overline{E}_{3k}^{m}(t)}{\partial E_{3k}^{ms}(t)} \times \frac{\partial E_{3k}^{ms}(t)}{\partial O_{3k}^{ms}(t)} \frac{\partial O_{3k}^{ms}(t)}{\partial I_{3k}^{ms}(t)} \frac{\partial I_{3k}^{ms}(t)}{\partial w_{jk}(t)} \tag{11}$$

where MSE during training per iteration, i.e. $\overline{E}_{3k}(t)$ may be obtained as follows:

$$\overline{E}_{3k}(t) = \frac{1}{M}\sum_{m=1}^{M} \overline{E}_{3k}^{m}(t) = \frac{1}{M}\sum_{m=1}^{M} \frac{1}{S_m}\sum_{s=1}^{S_m} E_{3k}^{ms}(t) = \frac{1}{M}\sum_{m=1}^{M} \frac{1}{2S_m}\sum_{s=1}^{S_m} \left[T_{3k}^{ms} - O_{3k}^{ms}(t)\right]^2. \tag{12}$$

To calculate the change in $v_{ij}(t)$ weight (i.e., $\Delta v_{ij}(t)$), the contributions of both the outputs are to be taken into account, i.e., errors due to both the outputs are to be combined together during the BP processing. Thus, it is essential to find out the effect of individual output in the change in error and to do so; four possible combinations are to be dealt with, as discussed below:

- Change in weights due to deviation output, associated with distance input,
- Change in weights due to acceleration output, associated with distance input,
- Change in weights due to deviation output, associated with angle input,
- Change in weights due to acceleration output, associated with angle input.

Therefore, the weights between input and hidden layers are updated by using the following expression:

$$v_{ij}(t+1) = v_{ij}(t) - \eta \frac{\partial \overline{E}_{3}(t)}{\partial v_{ij}(t)} + \alpha \Delta v_{ij}(t-1) \tag{13}$$

where

$$\frac{\partial \overline{E}_{3}(t)}{\partial v_{ij}(t)} = \frac{1}{2}\sum_{k=1}^{2} \frac{\partial \overline{E}_{3k}(t)}{\partial v_{ij}(t)} \tag{14}$$

where

$$\frac{\partial \overline{E}_{3k}(t)}{\partial v_{ij}(t)} = \frac{\partial \overline{E}_{3k}(t)}{\partial \overline{E}_{3k}^{m}(t)} \frac{\partial \overline{E}_{3k}^{m}(t)}{\partial E_{3k}^{ms}(t)} \frac{\partial E_{3k}^{ms}(t)}{\partial O_{3k}^{ms}(t)} \frac{\partial O_{3k}^{ms}(t)}{\partial I_{3k}^{ms}(t)} \times \frac{\partial I_{3k}^{ms}(t)}{\partial O_{2j}^{ms}(t)} \frac{\partial O_{2j}^{ms}(t)}{\partial I_{2j}^{ms}(t)} \frac{\partial I_{2j}^{ms}(t)}{\partial v_{ij}(t)}. \tag{15}$$

It is important to mention that the bias values of all the neurons are assumed to be constant throughout the study

and training is provided for a fixed number of hidden neurons at a time. A set of 200 training data are created at random, in which initial position, size, velocity, and direction of movement of the obstacles are varied. During training, the learning rate η and momentum constant α are varied in the range of (0.01, 0.1) and (0.1, 1.0), respectively. BP algorithm is initiated by randomly generating the weights in a range of (0, 1) and is terminated when the difference in error in two successive iterations becomes equal to or less than a user-specified value of 0.00001. It is important to mention that there is a chance that the updated weights may come out of their ranges. In such a situation, those particular weights are modified by using the expression given below:

$$V_{ij}(t+1) = \frac{1 - V_{ij}^2(t+1)}{1 + V_{ij}^2(t+1)}. \tag{16}$$

3.2 Approach 2: Genetic-neural (GA-NN) system

Realizing the fact that it is difficult to develop a neural controller through explicit design, researchers working in this field started thinking, whether it can be evolved by using an evolutionary technique. Simultaneous optimization of weights and the architecture of an NN is addressed in this section. To select proper magnitudes of the constant of activation functions (C_1, C_2, C_3) and to optimize the weights of the network, we need to deal with a few continuous variables, whereas tuning of the architecture involves the problem dealing with discrete variables. Thus, the present problem can be treated as a mixed-integer optimization problem, involving both the integer and real variables. A binary-coded GA with 850-bits long string is used for this purpose. The first 30 bits will carry information of three continuous variables — C_1, C_2, C_3 (10 bits for each variable), representing the constants of hyperbolic functions at three different layers. Out of the remaining 820 bits, every 41 bits (starting from 31st bit location of 850-bits long string) are used to indicate the existence of a hidden neuron (1 for presence and 0 for absence) and its corresponding four synaptic weights (10 bits for each weight). Therefore, a GA-string will look as follows:

$$\underbrace{1\cdots1}_{C_1}\ \underbrace{0\cdots1}_{C_2}\ \underbrace{1\cdots0}_{C_3}\cdots$$

$$\cdots\ \underbrace{\underbrace{1}_{j\text{th hidden neuron}}\ \underbrace{1\cdots1}_{v_{1j}}\ \underbrace{0\cdots1}_{v_{2j}}\ \underbrace{1\cdots0}_{w_{j1}}\ \underbrace{0\cdots0}_{w_{j2}}\cdots}_{\text{Architecture of NN}}$$

in which 41-bits are shown to indicate the presence of jth neuron and its connecting weights, such as $v_{1j}, v_{2j}, w_{j1}, w_{j2}$.

It is important to mention that we have restricted our search up to a maximum of twenty neurons lying in the hidden layer. During optimization, the constants of activation function for three layers are varied in a range of (0.1 to 15.0) and the weights are allowed to vary from 0.0 to 1.0. The ranges of variation of different variables are selected after a careful study.

Figure 3 shows the working principle of the combined GA-NN approach. The GA begins its search by randomly creating a number of solutions (equals to the population size) represented by the binary strings and each string indicates a typical NN-based controller. A particular NN-controller differs from other, in terms of the number of hidden neurons, connecting weights and constants of activation function at different layers. Each solution in the population is then evaluated, to assign a fitness value. After the fitness is assigned to each solution in the population, they are modified by using three operators — reproduction, uniform crossover and bit-wise mutation. One iteration involving these three operators followed by the fitness evaluation, is called a generation. Generations proceed until a termination criterion is satisfied. In this approach, the GA is allowed to run for a pre-specified number of generations. The fitness evaluation criterion, as followed in the present work, is discussed below.

3.2.1 Fitness calculation Navigation problem of a car-like robot is considered in the present work. It is to be noted that the same set of 200 training data used in Approach 1, has also been utilized in Approach 2. For each training scenario, the robot will initiate its motion from a pre-defined starting position and reaches the goal in 2-D environment, by avoiding collisions with the obstacles and to complete the path, it may take several time steps (ΔT). Thus, in each time step ΔT, the NN-controller will have to determine the two outputs, namely *deviation* and *acceleration*, based on the two input conditions, such as *distance* and *angle*. Our aim is to design the controller, in such a manner that the robot is able to reach the goal in the lowest possible traveling time, by avoiding collisions with the obstacles and not violating any constraint. Hence, the present problem can be treated as a constrained traveling time minimization problem. The minimum traveling time is possible to achieve, only when the robot travels with zero deviation and maximum acceleration. Therefore, the fitness of a GA-string (say, n) is considered to be the average of cumulative *deviation* and *acceleration* errors, determined after the whole training set is passed and it is given as follows:

$$Fitness = \frac{1}{M}\sum_{m=1}^{M}\frac{1}{S_m}\sum_{s=1}^{S_m}\sum_{k=1}^{2}\left|T_{3k}^{ms} - O_{3k}^{ms}(n)\right| \tag{17}$$

where all the symbols have their usual meaning as explained in Section 3.1.

It is important to mention that the error in deviation could be either positive or negative. Thus, an absolute value of error is taken for the fitness determination. Moreover, if the output of the controller in the predicted distance step is such, that the robot may collide with the most critical obstacle during its movement from the present position to the predicted position, a fixed penalty equals to 200 (selected at random) is added to the fitness. Again, sometimes the robot's motion as suggested by the NN-based controller, is not possible to implement due to its kinematic and/or dynamic constraints. In such a situation, the robot is stopped for that particular time step and a fixed penalty equals to 2000 (selected at random) is added to the fitness, to avoid such incidences. As there are two inputs of the NN-architecture and there is some coupling effect among them, topology of the NN having less than three hidden neurons may affect its generalizing capability. To avoid such a situation, another fixed penalty (equals to 2000,

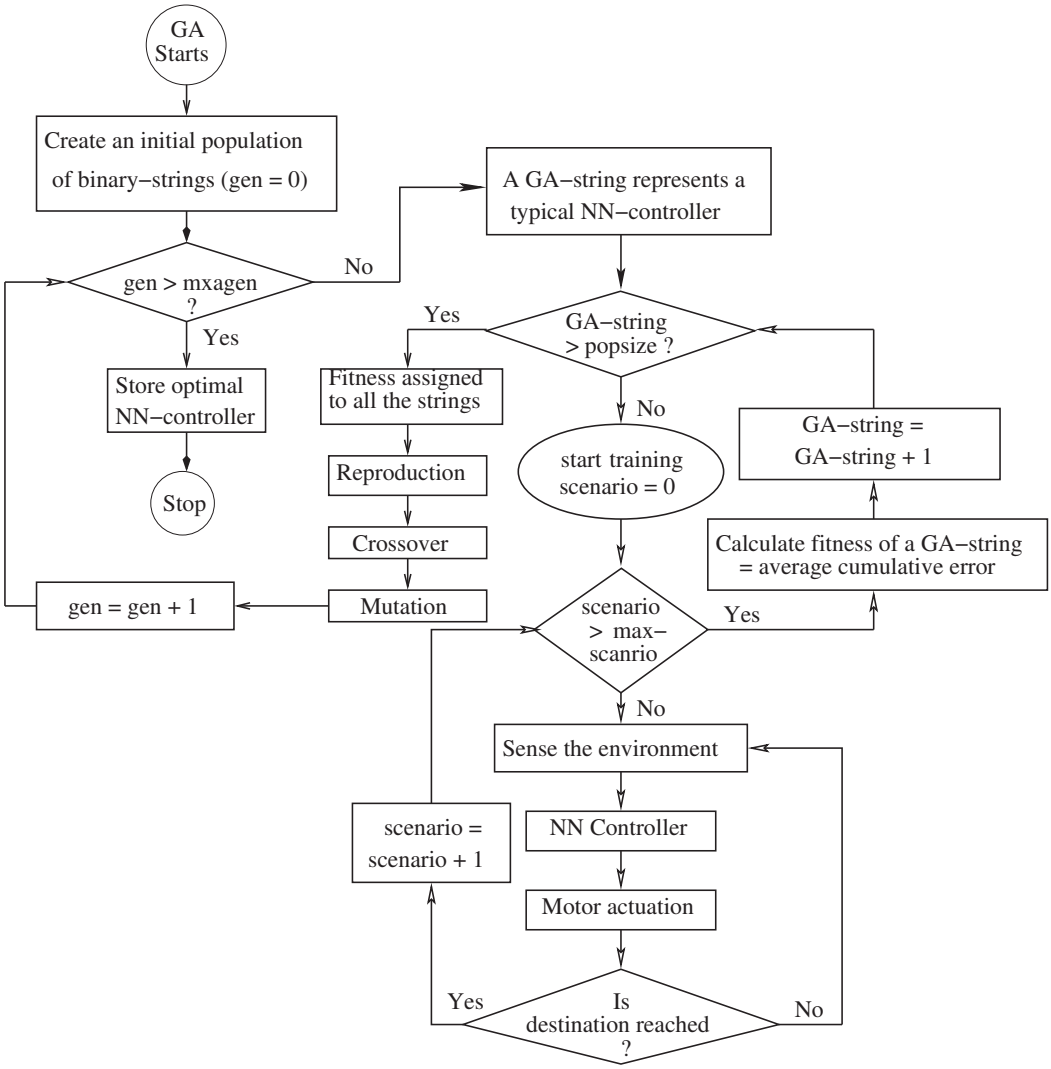


Fig. 3 A schematic diagram showing the working principle of the genetic-neural system

which is selected at random) is added to the fitness of the GA.

3. 3 Approach 3: Potential field method

Potential field method is one of the most popular conventional methods for solving the motion-planning problems of a mobile robot. In this approach, the robot in the configuration space is represented as a particle under the influence of an artificial potential field. The potential field function can be defined over free surface as the sum of an attractive potential, pulling the robot towards the goal configuration and a repulsive potential pushing the robot away from the obstacle. However, the performance of the potential field method depends on the chosen artificial potential function. Several potential functions, such as parabolic-well, conic-well, hyperbolic function, rotational field function, quadratic, exponential function, are tried by various investigators [2], [5], out of which, parabolic and hyperbolic functions are widely used for solving the similar problem [52], due to their nonlinear approximation capability about the system. The attractive potential field $U_{att}(X)$ can be defined as a parabolic-well as follows:

$$U_{att}(X) = \frac{1}{2}\xi_{att}\, d^2_{goal}(X) \tag{18}$$

where ξ_{att} is a positive scaling factor of attractive potential and $d_{goal}(X)$ denotes the Euclidean distance of the robot from its current position to the goal.

The repulsive potential field $U_{rep}(X)$ can be expressed as follows:

$$U_{rep}(X) = \frac{1}{2}\xi_{rep}\left[\frac{1}{d_{obs}(X)} - \frac{1}{d_{obs}(0)}\right]^2 \tag{19}$$

where ξ_{rep} is a positive scaling factor of repulsive potential and $d_{obs}(X)$ indicates the Euclidean distance of the robot from the obstacle and $d_{obs}(0)$ represents the distance of influence of the obstacle and is made equal to the distance between the center of the robot's bounding circle and that of the obstacle.

Attractive potential force is then determined by differentiating the attractive potential with respect to $d_{goal}(X)$, as given below:

$$F_{att}(X) = \xi_{att}\, d_{goal}(X). \tag{20}$$

Figure 4(a) shows a graph of attractive force $F_{att}(X)$ vs. distance $d_{goal}(X)$, from which, it can be observed that when the distance between the robot and its goal becomes equal to zero, there will not be any attractive force and it increases in a linear fashion with the increase of $d_{goal}(X)$.

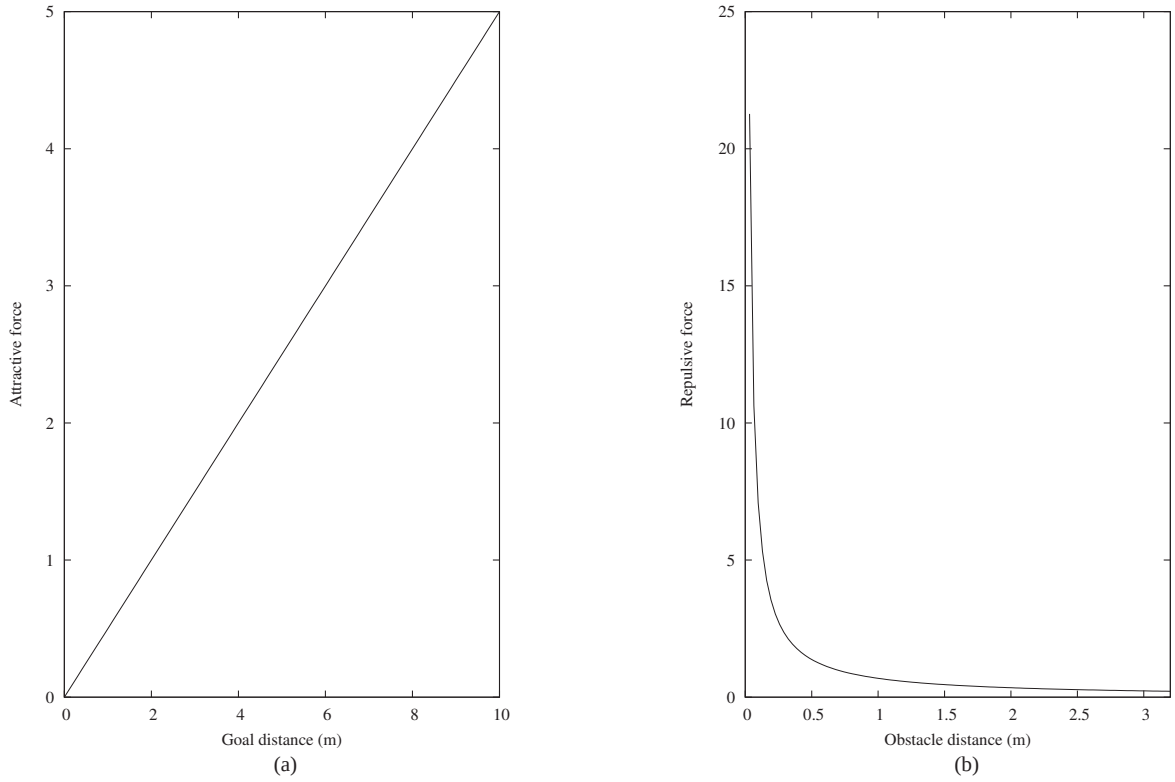


Fig. 4 Variations of the forces with goal distance and obstacle distance — (a) attractive force vs. goal distance and (b) repulsive force vs. obstacle distance

The above pattern of the attractive force is chosen after a careful study, to ensure that the robot reaches the goal with zero velocity.

Similarly, the repulsive potential force $F_{rep}(X)$ can be determined as follows:

$$F_{rep}(X) = -\xi_{rep}\ \frac{1}{d_{obs}^2(X)}\left[\frac{1}{d_{obs}(X)} - \frac{1}{d_{obs}(0)}\right]. \quad (21)$$

A graph of repulsive force $F_{rep}(X)$ vs. obstacle distance $d_{obs}(X)$ is shown in Fig. 4(b). From this figure, it is evident that the obstacles closer to the robot, exert more repulsive force compared to those which are far away from it. The resultant potential force $F(X)$ is then calculated by adding $F_{att}(X)$ with $F_{rep}(X)$ vectorially. In this approach, the acceleration output is taken to be proportional to the magnitude of the resultant force $F(X)$ and deviation output is considered as the angle made between the direction of the resultant potential force and the new reference line joining the CG of the robot at the present time step and the goal position.

4. Results and Discussion

The navigation problems of a car-like mobile robot, in the presence of some moving obstacles, are solved with the help of an NN-based controller, which is optimized with the help of a set of training scenarios, to get the best result. Two different approaches are proposed to optimize the NN controller. In Approach 1 (i.e., BPNN Approach), the connecting weights of the network are updated with the help of a BP algorithm as explained in the Section 3.1, whereas in Approach 2 (i.e., GA-NN Approach), both the connecting weights and the architecture of the network are optimized by using a binary-coded GA. Two hundred training scenarios are generated at random for the training purpose. A particular training scenario is different from the other, in terms of the initial position of the obstacles and their size, speed and direction of movement. The time interval (ΔT) is taken to be equal to sixteen seconds. The robot is assumed to have a maximum and minimum acceleration of 0.05 [m/s^2] and 0.005 [m/s^2], respectively. It is to be noted that the robot is assumed to be a cube of 63 [mm] × 63 [mm] × 63 [mm] and the maximum and minimum velocities of the robot are considered to be equal to 0.2 [m/sec] and 0.02 [m/sec], respectively. Results of computer simulations are presented for three different cases. In simulations, the radii of the obstacle boundaries are varied between 0.1 [m] to 0.5 [m] and their velocities are assumed to vary within a range of 0.1 [m/sec] to 0.4 [m/sec]. In Case 1, the robot is allowed to navigate among eight moving obstacles, whereas more complex environment having twelve and sixteen moving obstacles are considered in Cases 2 and 3, respectively. The performances of Approach 1 and Approach 2 are compared among themselves and to that of a potential field method (i.e., Approach 3), for solving the same problem.

4.1 Case-1: Navigation among eight moving obstacles

The navigation problem of a car-like robot among eight moving obstacles is studied in a grid of 19.95 × 19.95 [m^2]. In Approach 1, several combinations of learning rate parameter η and momentum constant α are simulated, to ob-

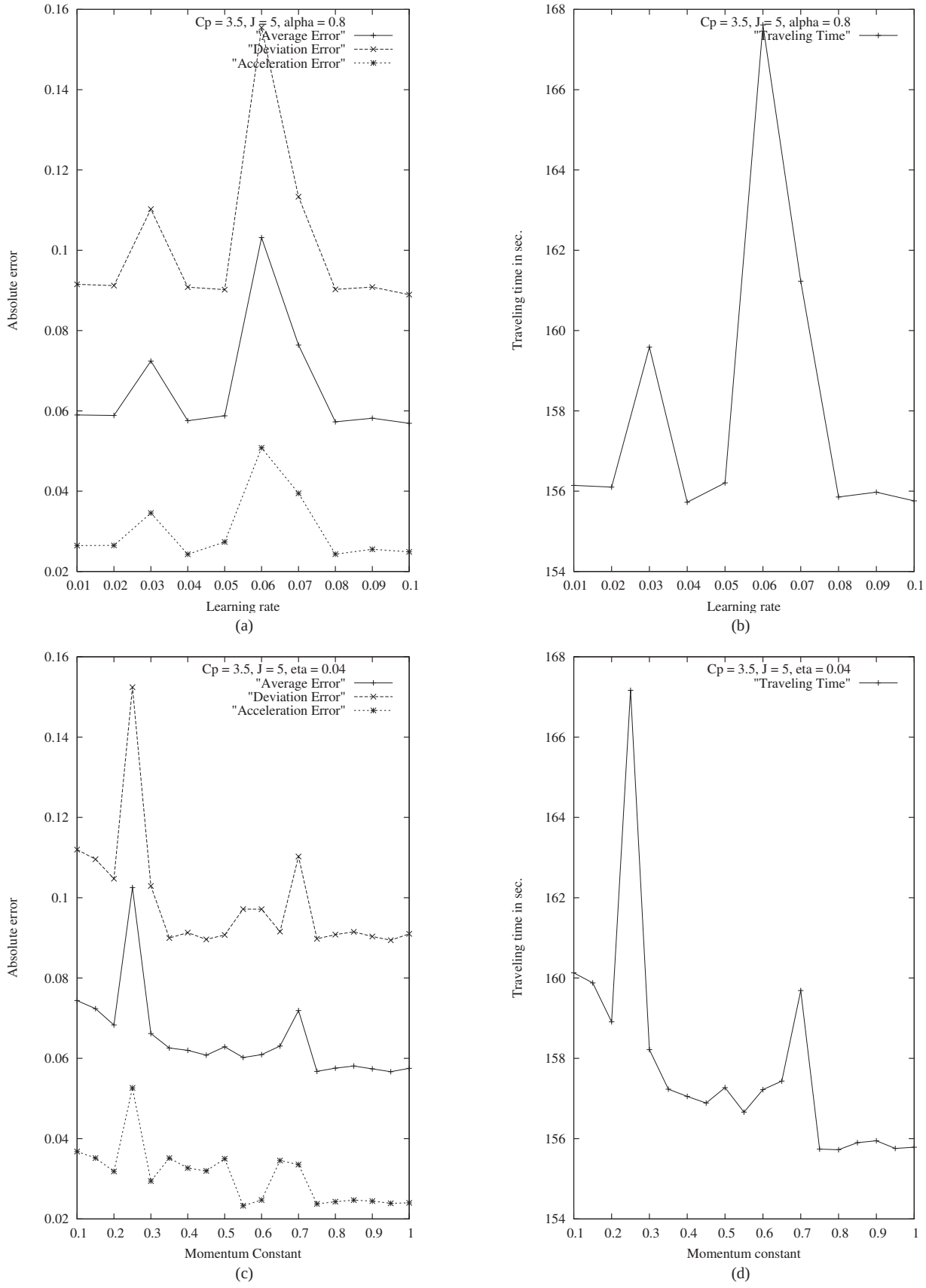


Fig. 5 Simulation results to obtain an optimal set of learning rate and momentum constant for eight obstacles case: (a) error vs. learning rate (η), (b) traveling time vs. learning rate (η), (c) error vs. momentum constant (α), (d) traveling time vs. momentum constant (α)

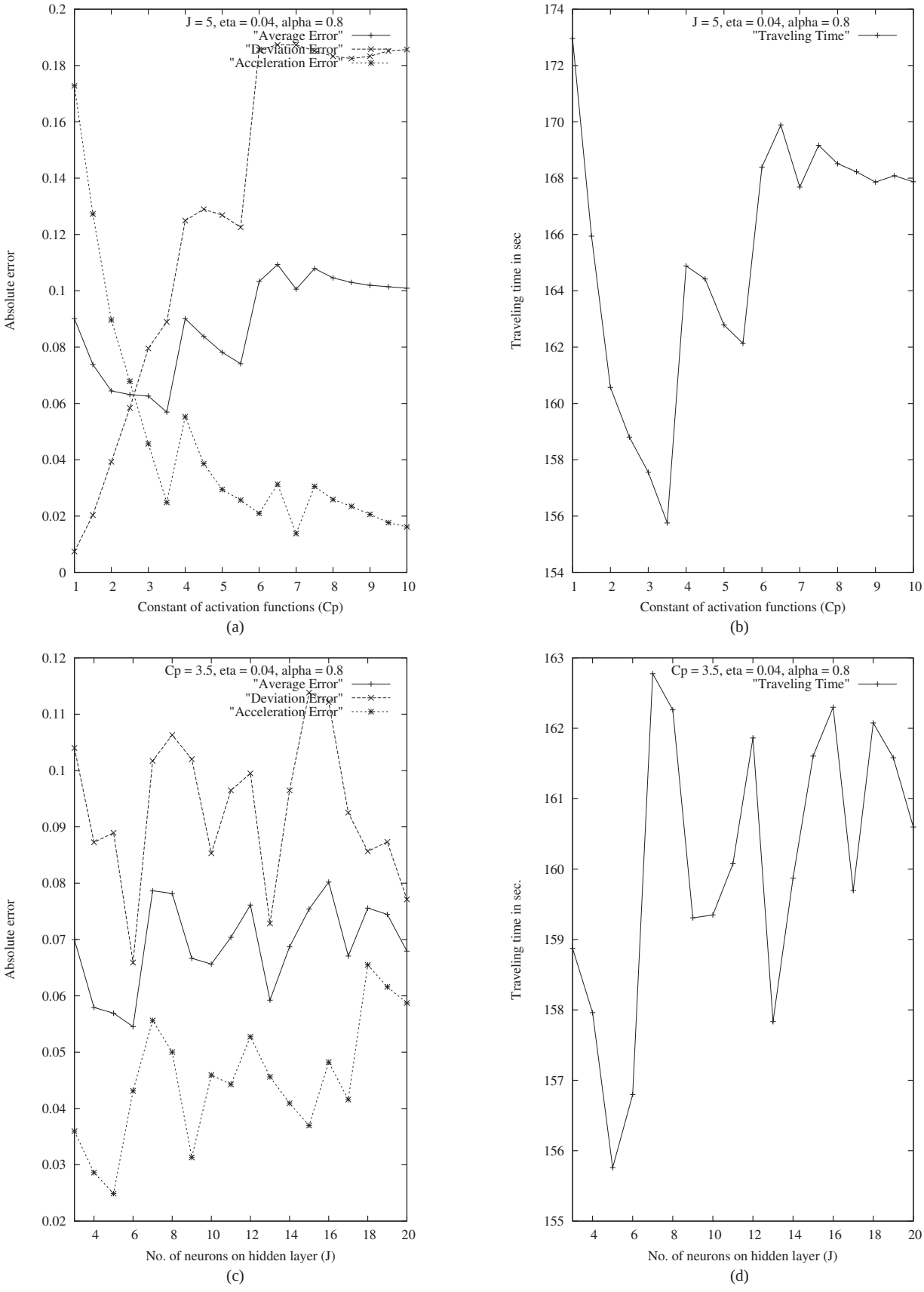


Fig. 6 Results of computer simulations to select the optimal constant value for activation functions and number of hidden neurons for eight obstacles case: (a) error vs. constant of activation functions (C_p)), (b) traveling time vs. constant of activation functions (C_p)), (c) error vs. number of hidden neurons (J), (d) traveling time vs. number of hidden neurons (J)

serve their effect on the network convergence. When the momentum constant is made equal to zero, the best convergence is achieved for $\eta = 0.1$, whereas the best convergence of the same network is achieved for a combination of $\eta = 0.04$ and $\alpha = 0.8$. The simulation results to determine the best combination of learning rate and momentum constant are shown in **Fig. 5**. It is important to note that although it is a traveling time minimization problem, it has been solved indirectly by minimizing the average of deviation and acceleration errors, for ease of implementation. Both the average error and traveling time have shown the similar trends with the learning rate and momentum constant, considered separately (refer to Fig. 5). As the performance of a BP learning algorithm depends on the constant of activation functions (C_p) and a proper choice of the number of hidden neurons also, experiments are carried out to select the optimal set of those parameters, for which the best performance of the NN is achieved. The experimental results for determining the optimal values of these two parameters of NN (Approach 1 with momentum), during the training are shown in **Fig. 6**. Nature of variations of the average error and that of traveling time with number of hidden neurons and constant of activation functions considered separately, are seen to be dissimilar at a few points (refer to Fig. 6). It could be due to the fact that the average error (in normalized scale) is calculated by giving equal weightage to both deviation and acceleration errors, whereas in practice, deviation error may have a slightly different effect on traveling time from that an acceleration error has. Moreover, it is to be noted that deviation error has been calculated based on the reference (the line joining the current position of the robot and the fixed goal), which itself is varying, whereas acceleration error is determined considering a fixed reference, i.e., the maximum permissible acceleration. The optimal parameters obtained in this way, are listed in **Table 1**. The performance of a GA is dependent on its parameter setting, experiments are carried out with different sets of parameters, to find the most suitable one. Results of the parametric study are shown in **Fig. 7**. The best results are obtained with the following GA-parameters: uniform crossover with probability $p_c = 0.5$, mutation probability $p_m = 0.00084$, population size $Y = 130$, maximum number of generation $Maxgen = 180$. Table 1 shows the optimal numbers of hidden neuron, activation function constants at different layers, training accuracy and training time required to converge to an error equal to 0.074456, during optimization. Moreover, Approach 2 has obtained a less average absolute error during training as compared to that of Approach 1 (refer to Table 1). It could be due to the fact that a BP algorithm may converge to a local minimum, whereas a GA may be called a global optimizer. It is to be noted that in Approach 1, the constants of activation functions of the different layers are assumed to be the same, to make the analysis simpler.

The performances of three approaches are studied for forty test scenarios (selected at random), which are not included among the training scenarios. In most of the scenarios, traveling time taken by the robot in Approach 2, comes out to be less than that of other two approaches, as shown in **Table 2**. It may be due to the fact that the potential field method does not have any in-built optimization module and as the BP algorithm works based on a steepest descent method, there is a chance of its solutions for getting stuck at local minima. A particular test scenario (say 4th of Table 2) is shown in **Fig. 8**, where the movements of both the obstacles and the robot are shown. An interesting fact may be noticed that when the direction of goal is just opposite to the direction of movement of the most critical obstacle, the robot by following the potential field method moves with a very low speed, as a result of which, the robot takes more time to reach the goal. Thus, the shortest distance path obtained by potential field method may not always be the optimal path, in terms of traveling time. However, the NN-based controllers are seen to tackle such situations effectively. It may be due to the fact that the NN-based approaches may have some adaptability.

Table 1 Optimized parameters related to the architecture of NN obtained by two approaches for eight obstacles case — (Case 1)

	Approach 1		Approach 2
	with momentum	without momentum	
Number of hidden neurons (J)	5	5	5
Constant of activation function Layer 1 (C_1)	3.5	12	14.479
Layer 2 (C_2)	3.5	12	9.964
Layer 3 (C_3)	3.5	12	8.582
Training error	0.057553	0.074456	0.006427
Training time (sec.) required to converge to an error of 0.74456	8.150	8.710	3.138
Test error	0.070466	0.125490	0.055416

To check the feasibility of these approaches, for online implementations, their CPU times are compared. It is to be noted that experiments are conducted on a Pentium - IV Intel PC. The CPU times of the BPNN with momentum, GA-NN approach and potential field method are found to be 0.015, 0.016 and 0.013 seconds, respectively. Thus, potential field method is seen to be the fastest of all, although the performance of GA-NN approach is found to be better than that of others, in terms of traveling time.

4. 2 Case 2: Navigation among twelve moving obstacles

In the present case, a car-like robot is allowed to navigate among twelve moving obstacles. In Approach 1, the best network convergence is noticed with the following combinations of learning parameters: $\eta = 0.09$ with zero momentum and $\eta = 0.09$, $\alpha = 0.9$. During training using a GA, the best result is obtained with the following GA parameters: $p_c = 0.5$, $p_m = 0.00028$, $Y = 50$, $Maxgen = 140$. The optimal number of neurons lying in the hidden layer and constants of activation functions, for which the best result is obtained by following the first two approaches are shown in **Table 3**. It is to be noted that Approach 2 has converged to an error value less than that obtained by Approach 1 during training (refer to Table 3). It could be due to the fact that the solutions provided by Approach 1 are locally optimal. It is also to be noted that Approach 2 has

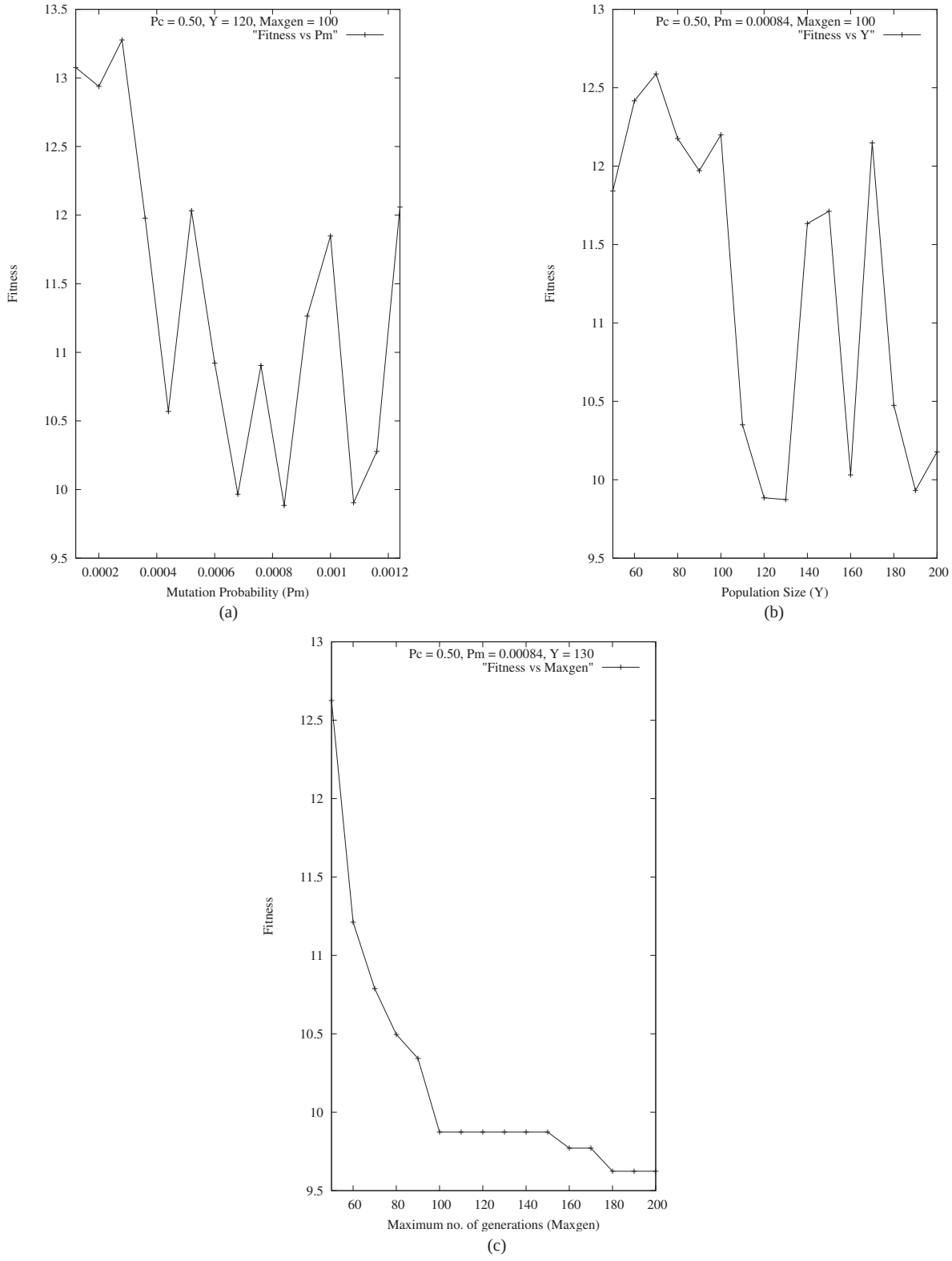
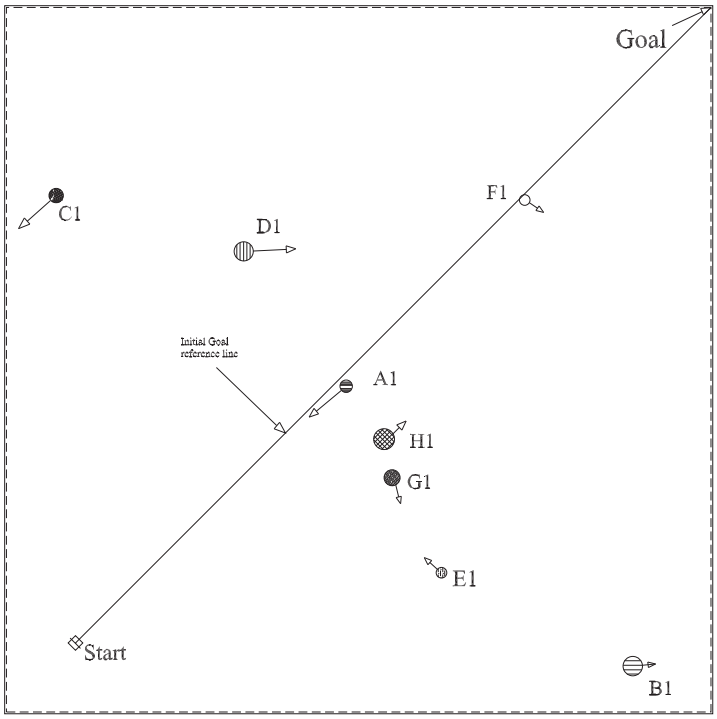


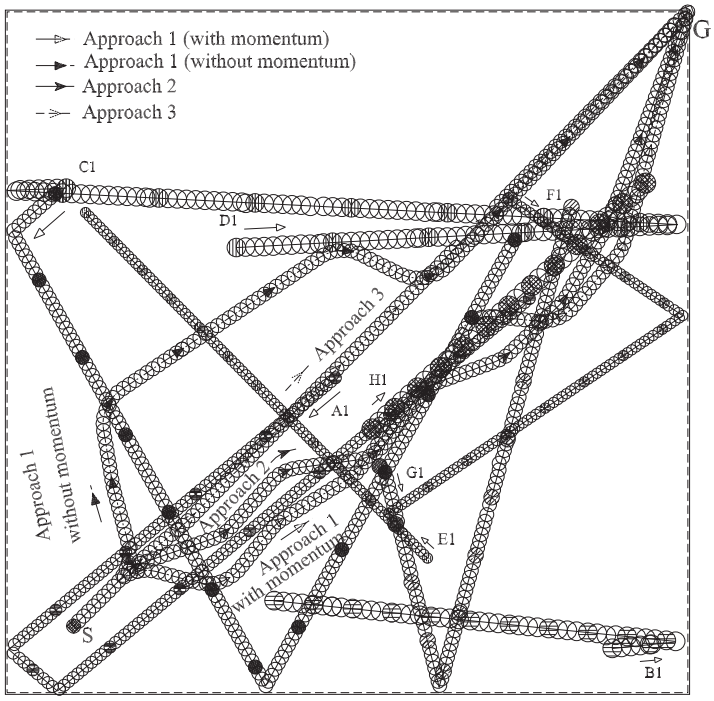
Fig. 7 Results of the parametric study to obtain the optimal GA-parameters for eight obstacles case: (a) fitness vs. mutation probability, (b) fitness vs. population size, (c) fitness vs. maximum no. of generations.

taken less CPU time to converge to a training error equals to 0.117783, compared to Approach 1.

After the training is over, effectiveness of these two NN-based approaches is compared to that of the potential field method, for forty randomly generated test scenarios (refer to **Table 4**). The traveling time taken by the robot by following Approach 2, has come out to be less compared to that of the other approaches, in most of the test scenarios. This could be due to the fact that both the Approaches 1 and 3 may suffer from local minima problem. A particu-



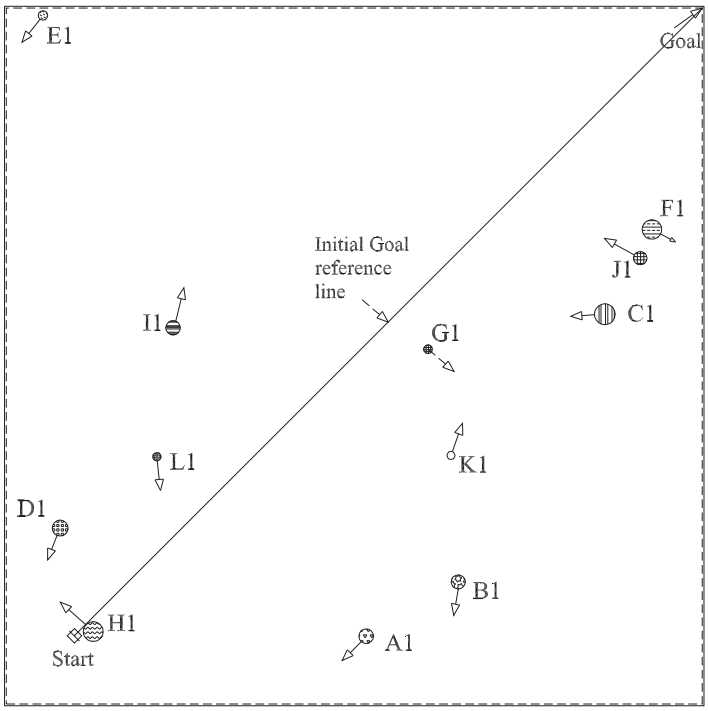
(a)



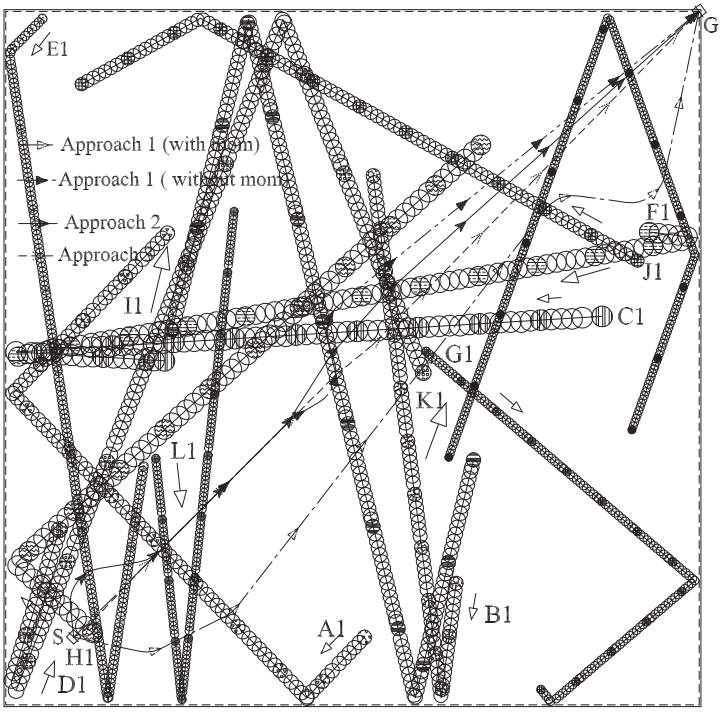
(b)

Fig. 8 Navigation problem of the robot in the presence of eight moving obstacles (Case 1, 4th test scenario of Table 2) — (a) initial position of the robot and obstacles, (b) collision-free paths obtained by the robot along with its positions using three different approaches

lar test scenario (say, 3rd of Table 4) is shown in **Fig. 9**, in which the complete path of the robot determined by following all the approaches are shown. CPU time values of all the approaches are found to lie within 0.01 to 0.043 seconds, thus making them suitable for online implementations.



(a)



(b)

Fig. 9 Navigation problem of the robot in the presence of twelve moving obstacles (Case 2, 3rd test scenario of Table 4) — (a) initial position of the robot and obstacles, (b) collision-free paths obtained by the robot using three different approaches

4. 3 Case 3: Navigation among sixteen moving obstacles

A car-like robot will have to find its collision-free, time-optimal path, while navigating among sixteen moving obstacles. Learning through BP algorithm showed the best convergence for a learning rate of $\eta = 0.1$ with zero momentum and for a learning rate of $\eta = 0.01$ with a momentum constant equal to 0.2. The following GA-parameters have provided the best result during training: $p_c = 0.5$, $p_m = 0.00068$, $Y = 120$, $Maxgen = 200$. **Table 5** shows the

Table 2 Comparison of three approaches in terms of traveling time in seconds — (Case 1)

	Approach 1		Approach 2	Approach 3
	with momentum	without momentum		
1	175.963730	178.164581	159.550446	178.524124
2	142.870316	146.946960	143.863022	146.425659
3	171.570801	205.301224	165.427750	209.450928
4	174.888504	178.143600	173.776031	184.135132
5	157.882721	172.881149	157.953186	178.245010
6	160.797668	181.496536	159.347763	194.534424
7	156.013824	149.744980	148.014725	161.899689
8	164.310211	188.803574	152.000000	189.441681
9	173.727585	179.673218	167.921829	190.962875
10	152.000000	155.528442	157.960449	176.404892
11	141.425659	141.425659	141.425659	141.425659
12	172.516312	241.048401	172.825790	289.274567
13	161.961761	155.759018	156.694977	300.836670
14	145.868912	156.379013	151.969116	162.882812
15	141.425659	141.425659	141.425659	141.425659
16	141.425659	141.425659	141.425659	141.425659
17	178.068741	223.757202	157.646286	236.722610
18	175.333572	187.712997	158.815842	226.126068
19	175.986343	157.300674	152.000000	189.769791
20	149.913300	197.349823	151.946030	223.653763
21	149.601776	156.764252	148.982162	161.598373
22	161.714920	157.345657	151.912170	160.582855
23	191.464325	171.854111	174.333206	176.487411
24	155.882629	168.000000	159.852829	189.633835
25	172.057816	207.908585	144.392242	231.962524
26	176.717545	205.273743	173.620514	204.485992
27	160.284363	162.642044	159.116898	252.080582
28	188.355911	228.945175	180.017624	252.810272
29	162.026566	179.627914	163.034134	188.768692
30	141.425659	141.425659	141.425659	141.425659
31	156.844055	156.941940	153.122543	162.793091
32	164.440552	173.797974	163.181793	188.707077
33	188.885849	197.947662	178.671295	223.671066
34	176.371811	188.577820	183.850067	176.367493
35	183.944870	209.801590	157.915100	223.711395
36	161.968826	162.064590	160.398346	199.937103
37	156.285263	165.808502	162.890671	189.224640
38	157.417282	157.720490	151.991440	163.232864
39	176.236145	236.130905	177.890533	190.068359
40	162.870514	164.450073	160.315414	192.058929

optimal number of hidden neurons and constants of activation functions, for which the best result is obtained by the Approaches 1 and 2. During training, Approach 2 is found to yield the less error compared to Approach 1. Moreover, Approach 2 has shown faster convergence than Approach 1, during the training. It may be due to the fact that Approach 1 requires dealing with a large number of parameters, selection of which plays an important role in the convergence criterion. As the BP algorithm, which works based on the steepest descent search, is replaced by a GA-based search in Approach 2, the chance of its solutions for getting trapped into the local minima is less.

Table 3 Optimized parameters related to the architecture of NN obtained by two approaches for twelve obstacles case — (Case 2)

	Approach 1		Approach 2
	with momentum	without momentum	
Number of hidden neurons (J)	4	5	9
Constant of activation function Layer 1 (C_1)	3	10	10.867
Layer 2 (C_2)	3	10	11.263
Layer 3 (C_3)	3	10	8.499
Training error	0.074412	0.117783	0.009430
Training time (sec.) required to converge to an error of 0.117783	1.30	5.20	0.85
Test error	0.091710	0.231172	0.082622

Traveling time values of three different approaches for forty test scenarios, created at random, are shown in **Table 6** and Approach 2 is found to perform better than other approaches, in most of the test scenarios. This could be due to the fact that the solutions provided by both potential field method and BPNN approach may be locally optimal. Results of these approaches are shown in detail for a particular test scenario (say, 4th of Table 6) in **Fig. 10**. Although Approach 3 has generated the shortest path (in terms of distance), it may not be the time-optimal one. It could be due to the fact that the potential field method is unable to provide with a sufficiently large value of acceleration to the robot. The CPU time values of all the three approaches are found to be less and thus, these are suitable for online implementations.

Results of all the three approaches have been compared to solve navigation problems of a car-like robot in a dynamic environment. Although Approach 3 has generated the shortest distance path in most of the scenarios, its performance in terms of traveling time is seen to be the worst. It could be due to the following reasons.

- The motion planner based on Approach 3 provides with low values of both the acceleration and deviation of the robot, when the most critical obstacle, in the predicted time step, comes closer to the line joining the robot and the goal.
- If the direction of movement of the most critical obstacle is almost perpendicular to the line joining the robot and the goal and the robot tries to deviate towards the predicted position of the most critical obstacle, a geometric correction is given to ensure a collision-free movement of the robot. In such a situation, there is a possibility that the traversed path may unnecessarily be a lengthy one, which is dependent on the direction of movement of the most critical obstacle.
- As the attractive potential force is linearly decreasing, when the robot comes closer to the goal (refer

Table 4 Comparison of three approaches in terms of traveling time in seconds — (Case 2)

	Approach 1		Approach 2	Approach 3
	with momentum	without momentum		
1	157.892914	284.952362	162.531097	203.579483
2	188.766449	196.521027	188.319489	199.779617
3	165.637451	187.521637	168.000000	173.367035
4	158.485672	184.000000	159.093124	189.957352
5	152.000000	231.757294	146.092575	241.238922
6	143.290436	174.537598	145.361938	162.882874
7	147.606812	179.412582	165.836731	236.976517
8	168.000000	259.632629	162.161011	173.478409
9	211.599670	245.099670	176.199188	175.036514
10	176.307892	165.103409	164.255417	204.778137
11	156.946335	205.076782	161.701904	199.237152
12	200.000000	200.000000	148.192230	161.052414
13	155.809418	295.377991	175.248550	199.831451
14	162.190140	236.847519	165.183197	238.406296
15	146.880386	173.849548	172.342392	199.878616
16	195.865067	183.758224	158.980438	226.737091
17	165.286484	293.490295	162.439362	308.092560
18	157.622467	223.836258	172.057587	237.052505
19	147.918060	174.961060	162.789490	183.599960
20	205.372238	210.784241	147.076416	161.401062
21	159.205566	207.230698	151.928879	253.271362
22	165.622375	195.524353	171.511841	215.655441
23	192.811371	193.002289	147.696457	160.232849
24	178.913452	244.769196	172.018219	252.794327
25	189.892044	171.534317	164.453400	211.038132
26	177.156967	188.773636	157.935944	210.623123
27	155.816284	155.811111	146.721817	162.530411
28	208.934158	220.654816	193.182022	220.655014
29	145.134613	173.446457	151.625595	175.004211
30	168.000000	175.360733	163.161438	175.906174
31	183.998703	164.295731	172.893967	177.974045
32	156.009918	192.230408	155.796539	204.643311
33	191.337112	194.832550	164.078369	199.695084
34	149.125092	168.000000	156.408417	173.168961
35	204.841568	173.776169	188.081970	224.593079
36	216.000000	216.000000	176.623459	199.715302
37	204.750305	223.005646	179.241913	227.459625
38	172.689072	192.667725	168.000000	191.577164
39	211.859238	221.553421	163.993881	221.600525
40	157.869400	157.261429	148.530991	157.709869

to Fig. 4(a)), the motion planner is unable to yield a higher value of acceleration, irrespective of the obstacle's position in the environment.

- The performance of Approach 3 depends on the chosen potential function.
- There is a chance of the solutions of potential field method to get trapped into local minima.

Table 5 Optimized parameters related to the architecture of NN obtained by two approaches for sixteen obstacles case — (Case 3)

	Approach 1		Approach 2
	with momentum	without momentum	
Number of hidden neurons (J)	6	5	8
Constant of activation function Layer 1 (C_1)	3.5	9.5	13.823
Layer 2 (C_2)	3.5	9.5	11.729
Layer 3 (C_3)	3.5	9.5	8.650
Training error	0.093850	0.136371	0.001369
Training time (sec.) required to converge to an error of 0.136371	7.160	7.310	4.728
Test error	0.156841	0.220330	0.117518

The effectiveness of the GA-NN approach is found to be better than that of other approaches. The potential field method is seen to be the fastest of all the approaches but it lacks adaptability. On the other hand, both BPNN and GA-NN approaches are able to provide with more adaptive solutions compared to those of Approach 3.

5. Comparison of the Developed Approaches with Some Other Approaches

The prime aim of this research is to design and develop an adaptive robot controller that can plan and control the motion of a car-like mobile robot, navigating among several moving obstacles. In the past, several attempts were made by various investigators to develop an adaptive NN-based controller. Some of these are mentioned below for the purpose of comparison with the present approaches.

Son *et al.* [43] used a GA to determine the optimal number of hidden neurons and the number of epochs up to which the learning of the network by following the Levenberg-Marquardt BP algorithm is to be continued. In the similar manner, Kim *et al.* [44] applied GAs to obtain the BPNN's parameters, such as number of neurons in the hidden layers, momentum constant, and learning rate. But, the main drawback of these two approaches lies in the fact that the actual optimization of the network parameters was performed with the help of a BP algorithm, which may suffer from so-called local minima problem and thus, resulting into a slow convergence rate. Sexton and Gupta [49] made a comparison of the potential of GA with that of the BP algorithm for NN learning. However, they restricted their search only with three NN-structures having two, four and six neurons in the hidden layer, separately. Again, none of the above mentioned approaches tried to evolve the NN-controller completely.

Yang and Meng [11] proposed an approach for dynamic robot motion planning, based on NN. But, their approach suffers from the following drawbacks. The structure of the NN was not optimized. Moreover, the performance of the controller was tested only on a point robot and a manipulator, whereas the motion planning of a car-like robot is much more difficult, as it is subjected to both nonholonomic and dynamic constraints. Gu and Hu [12] developed

a path tracking scheme for a car-like robot based on an NN. However, their model may fail to perform well in a situation, where the robot is subjected to some dynamic constraints, as it was not taken into account in their model.

In the present study, a suitable controller based on a feed-forward NN is designed to solve navigation problems of a car-like robot in a dynamic environment. Two separate approaches based on NN are developed for this purpose. In Approach 1, connecting weights of a fixed architecture NN are updated by using a BP learning algorithm and a parametric study is carried out, to determine the optimal parameters of the network. In the next approach, both the topology and connecting weights are evolved by using an evolutionary technique, such as GA. The main advantage of this approach lies in the fact that the designer plays a passive role and the desired behaviors emerge automatically through evolution. It is interesting to note that the developed NN-based controller is capable of tackling both kinematic and dynamic constraints of the robot.

6. Concluding Remarks

The approach of evolutionary robotics appears to be promising for the development of intelligent and autonomous robots. Evolution of a suitable NN-based controller has become a thrust area in robotic research, due to its important characteristics, such as generalizability, adaptability and learning capability. The performance of an NN depends on its both architecture and the free parameters like, synaptic weights, biases values and others. Several methods are developed by various investigators to optimize all such parameters as well as the topology of the NN, but each of these methods has its inherent limitations and/or application restrictions.

In the present study, an attempt is made to update the connecting weights and the structure of the NN by following two different approaches. In Approach 1, the weights are optimized by following a BP algorithm and some experiments are carried out to select the best set of learning parameters. The optimal values related to the number of neurons in the hidden layer and the constants of activation functions are obtained through conducting experiments with different sets of parameters, to get the best accuracy. In Approach 2, the whole task of designing an NN, is given to a binary-coded GA. Through search, it has developed a suitable topology of the network along with its synaptic weights, due to the interactions between the robot and its environment. The performances of these two NN-based approaches are compared to those of a potential field-based approach (i.e., Approach 3), through computer simulations.

Once NN-based controller is optimized, it can be used to solve the navigation problems of a car-like robot. The performance of all the approaches is compared for forty randomly generated test scenarios. In most of the scenarios, Approach 2 is found to perform better than Approaches 1 and 3. It could be due to the fact that the potential field method (i.e., Approach 3) does not have any in-built optimization module and in Approach 1, optimization of the NN is carried out by using a BP algorithm, which may have the local minima problem. The supremacy of Approach 2 over Approach 1 could be due to the fact that a GA has a wider search space compared to that of a steepest descent method.

Table 6 Comparison of three approaches in terms of traveling time in seconds — (Case 3)

	Approach 1		Approach 2	Approach 3
	with momentum	without momentum		
1	163.341782	188.071014	171.554153	197.110733
2	161.304855	211.339539	175.014465	177.132309
3	167.997696	196.672714	161.692078	188.237930
4	226.383804	268.111298	189.368195	240.551376
5	178.026932	204.123886	181.034958	226.416428
6	205.177536	268.625305	199.817917	269.983582
7	240.712158	164.335464	157.133667	188.798767
8	148.731491	177.649124	159.946655	173.117950
9	200.165787	231.925690	179.818863	255.768692
10	180.635849	188.171219	175.021042	309.344269
11	178.281082	268.079529	157.933731	222.356277
12	155.678619	164.714600	172.410095	188.612152
13	197.509872	258.555695	205.939774	284.528534
14	196.360428	206.361374	198.719345	331.945282
15	235.943085	161.401688	161.616028	283.958923
16	242.037170	247.981537	208.894714	317.982727
17	234.919418	236.205399	196.219070	261.396179
18	208.846039	221.679184	204.030457	252.236649
19	210.784866	237.188995	172.259628	222.016479
20	211.734558	223.036972	208.850037	235.736603
21	152.000000	149.066010	148.281784	215.753647
22	173.227646	204.195328	163.642578	208.244797
23	212.352478	180.289124	158.908966	191.382004
24	245.694885	340.608582	176.271820	342.417969
25	181.428558	190.504181	167.863159	190.119141
26	165.421600	165.701263	162.287842	183.400391
27	205.916870	222.703369	179.912155	260.493011
28	268.862152	268.775360	200.000000	302.918793
29	152.000000	179.431900	184.000000	206.895340
30	156.217514	204.372940	151.655670	209.636627
31	156.752350	190.393906	159.843781	212.924362
32	188.619858	199.146484	188.056213	204.380264
33	172.829163	237.693588	172.376266	240.304871
34	141.425659	141.425659	141.425659	141.425659
35	190.089096	209.220108	193.235748	255.053772
36	237.669479	190.280869	189.019592	160.811920
37	189.629410	193.022324	167.918121	199.867569
38	195.674774	218.199936	167.896637	204.271500
39	223.440338	179.302765	152.000000	236.599747
40	162.642838	161.155991	155.699799	160.941269

The entire optimization/evolution of an NN is conducted offline. Once the best controller is obtained, it is used to solve the test scenarios. As the values of CPU time of all the approaches are seen to lie within an acceptable limit, these algorithms might be suitable for online implementa-

tions.

It has been observed through computer simulations that the shortest distance path determined by the potential field approach might not be always the time-optimal path. On the other hand, the developed NN-based approaches have the tendency to find time-optimal, collision-free path of the robot. Moreover, potential field approach has failed to yield feasible solutions in a number of occasions but those have been effectively tackled by using the NN-based approaches. Thus, NN-approaches are found to be more adaptive to the environment compared to the potential field approach.

7. Scope for Future Work

The present work can be extended in a number of ways. Some of these are mentioned below, on which the authors are working at present.

- Designing a suitable NN architecture needs to deal with a number of parameters, selection of which plays an important role on its effectiveness. An attempt will be made to model the fuzziness that exists in the NN-parameters, to get the best accuracy during training.
- The effectiveness of the developed NN controller is tested on computer simulations. It will be interesting to download the optimal algorithm on a real robot and examine its performance.
- In the present work, navigation problems of a single robot have been tackled, in the presence of some moving objects. However, it will be more interesting to replace the moving objects by some mobile robots. Thus, it will constitute a more complex problem involving coordination, cooperation and communication of multiple robots navigating in a common dynamic environment.

Acknowledgment

This work is supported by the Department of Science and Technology, Govt. of India (Sanction No. SR/S3/RM/28/2003 dt. 12.12.2003). The authors gratefully acknowledge the cooperation of Dr. A. Roy Chowdhury of the Department of Mechanical Engineering, IIT–Kharagpur, India.

References

[1] D. Feng, H. Krog and H. Bruce, "Dynamic steering control of conventionally steered mobile robots," in *Proc. IEEE Conf. Robotics and Automation*, 1990, pp. 390–395.

[2] J. C. Latombe, *Robot motion planning*. Dordrecht, The Netherlands: Kluwer Academic Publishers, 1991.

[3] P. Fiorini and Z. Shiller, "Motion planning in dynamic environments using velocity obstacles," *International Journal of Robotics Research*, vol. 17, no. 7, pp. 760–772, 1998.

[4] J. Wang, Z. Qu, Y. Guo, and J. Yang, "A reduced-order analytical solution to robot trajectory generation in the presence of moving obstacles," in *Proc. IEEE Conf. Robotics and Automation*, 2004, pp. 4301–4307.

[5] A. Bemporad, A. D. Luca, and G. Orilo, "Local incremental planning for a car-like robot navigating among obstacles," in *Proc. IEEE Conf. Robotics and Automation*, Minneapolis, Minnesota, April 1996, pp. 1205–1211.

[6] D. K. Pratihar, "Algorithmic and soft computing approaches to robot motion planning," *Machine Intelligence and Robotic Control*, vol. 5, no. 1, pp. 1–16, 2003.

[7] J. A. Meyer, "Evolutionary approaches to neural control in mobile robots," in *Proc. IEEE Conf. on Systems Man & Cybernetics*, October 11–14 1998, vol. 3, pp. 2418–2423.

[8] C. Torras, "Robot adaptivity," *Robotics and Autonomous Systems*, vol. 15, pp. 11–23, 1995.

[9] T. Gomi and A. Griffith, "Evolutionary robotics — An overview," in *Proc. IEEE Intl. Conf. Evolutionary Computation*, IEEE Society Press, 1996, pp. 40–49.

[10] I. Harvey, P. Husbands, D. Cliff, A. Thompson, and N. Jakobi, "Evolutionary robotics: The sussex approach," *Robotics and Autonomous Systems*, vol. 20, pp. 205–224, 1997.

[11] S. X. Yang and M. Meng, "An efficient neural network approach to dynamic robot motion planning," *Neural Networks*, vol. 14, pp. 143–148, 2000.

[12] D. Gu and H. Hu, "Neural predictive control for a car-like mobile robot," *Robotics and Autonomous Syst.*, vol. 39, pp. 73–86, 2002.

[13] F. Mondada and D. Floreano, "Evolution of neural control structures: Some experiments on mobile robots," *Robotics and Autonomous Systems*, vol. 16, pp. 183–195, 1995.

[14] D. Floreano and F. Mondada, "Evolutionary neuro controllers for autonomous mobile robots," *Neural Networks*, vol. 11, pp. 1461–1478, 1998.

[15] O. Miglino, H. H. Lund, and S. Nolfi, "Evolving mobile robots in simulated and real environments," *Artificial Life*, vol. 2, pp. 417–434, 1995.

[16] S. Nolfi and D. Parsi, "Learning to adapt to changing environments in evolving neural networks," *Adaptive Behavior*, vol. 5, no. 1, pp. 75–98, 1997.

[17] P. Husbands, I. Harvey, D. Cliff, and G. Miller, "The use of genetic algorithms for the development of sensorimotor control systems," in *From Perception to Action*, Nicoud and Gaussier Eds. Los Alamitos, CA: IEEE computer society press, 1994, pp. 110–121.

[18] N. Noguchi and H. Terao, "Path planning of an agricultural mobile robot by neural network and genetic algorithm," *Computers and Electronics in Agriculture*, vol. 18, pp. 187–204, 1997.

[19] S. Baluja, "Evolution of an artificial neural network based autonomous land vehicle controller," *IEEE Trans. Syst. Man Cybern. - Part B: Cybern*, vol. 26, no. 3, pp. 450–463, 1996.

[20] L. A. Meeden, "An incremental approach to developing intelligent neural controllers for robots," *IEEE Trans. Syst. Man Cybern. - Part B: Cybern*, vol. 26, no. 3, pp. 474–485, 1996.

[21] D. Cliff, I. Harvey, and P. Husbands, "Exploration in evolutionary robotics," *Adaptive Behavior*, vol. 2, no. 1, pp. 73–110, 1993.

[22] M. Lee, "Evolution of behaviors in autonomous robot using artificial neural network and genetic algorithm," *Information Sciences*, vol. 155, pp. 43–60, 2003.

[23] D. K. Pratihar, "Evolutionary robotics — A review," *Sadhana*, vol. 28, no. 6, pp. 999–1003, 2003.

[24] A. L. Nelson, E. Grant, and T. C. Henderson, "Evolution of neural controllers for competitive game playing with teams of mobile robots," *Robotics and Autonomous Syst.*, vol. 46, pp. 135–150, 2004.

[25] K. Heero, J. Willemson, A. Aabloo, and M. Kruusmaa, "Robots find a way: A learning method for mobile robot navigation in partially unknown environments," *Intelligent Autonomous Systems*, F. Groen et al. Eds. Amsterdam, The Netherlands: IOS Press, 2004, vol. 8, pp. 559–566.

[26] T. Kondo, "Evolutionary design and behavior analysis of neuromodulatory neural networks for mobile robots control," *Applied Soft Computing*, available online 10 August 2005 (in press).

[27] G. Capi and K. Doya, "Evolution of neural controllers using an extended parallel genetic algorithm," *Robotics and Autonomous Systems*, vol. 52, pp. 148–159, 2005.

[28] H. Hagras and T. Sobh, "Intelligent learning and control of autonomous robotic agents operating in unstructured environments," *Information Sciences*, vol. 145, pp. 1–12, 2002.

[29] S. Yamada, "Evolutionary behavior learning for action based environment modeling by a mobile robot," *Applied Soft Computing*, vol. 5, pp. 245–257, 2005.

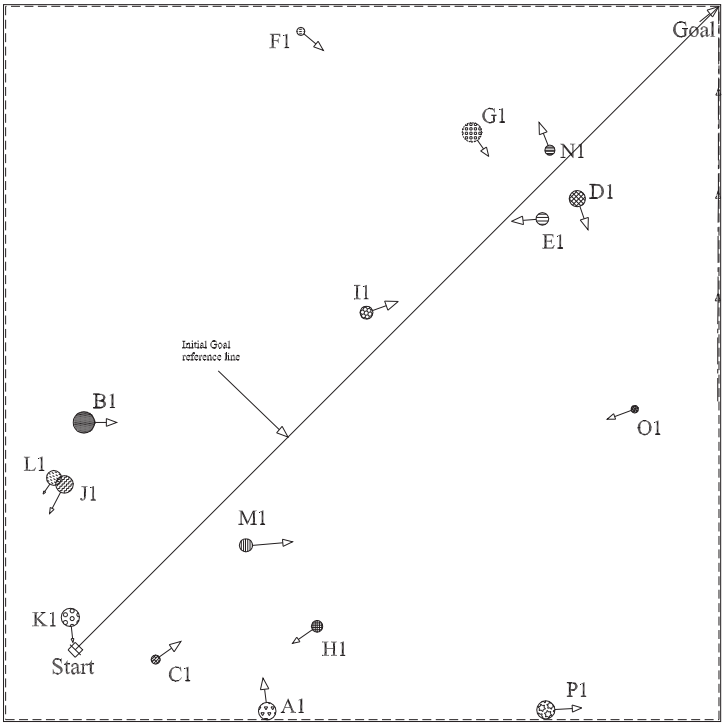
[30] D. O. Hebb, *The Organization of Behavior*. New York, NY: Wiley, 1949.

[31] B. Widrow and M. E. Hoff, "Adaptive switching capability and its relation to the mechanism of association," *Kybernetik*, vol. 12, pp. 204–215, 1960.

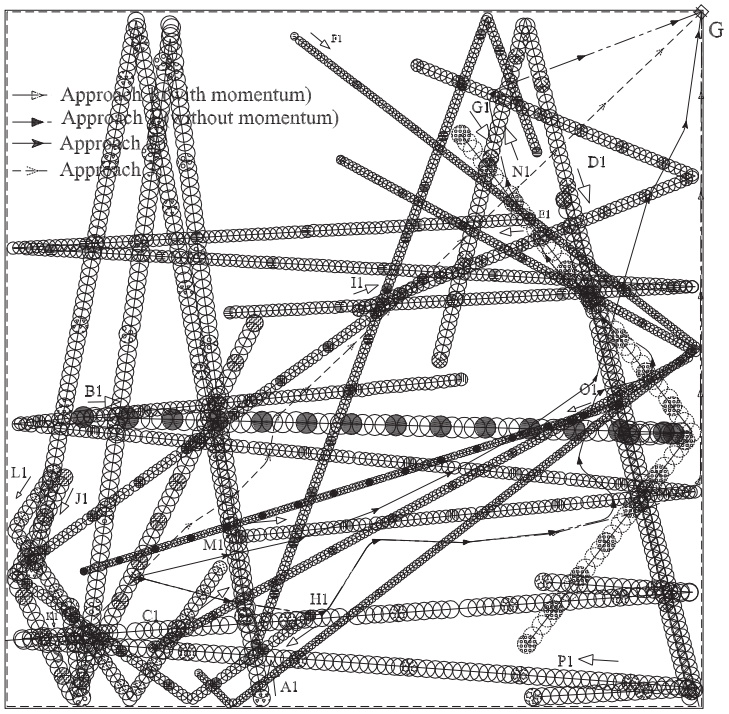
[32] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagation errors," *Letters in Nature*, vol. 323, pp. 533–535, 1986.

[33] A. G. Barto and P. Anandan, "Pattern-recognizing stochastic learning automata," *IEEE Trans. Syst. Man Cybern.*, vol. 15, no. 3, pp. 360–375, 1985.

Please update the information, if you have



(a)



(b)

Fig. 10 Navigation problem of the robot in the presence of sixteen moving obstacles (Case 3, 4th test scenario of Table 6) — (a) initial position of the robot and obstacles, (b) collision-free paths obtained by the robot using three different approaches

[34] W. L. Goffe, G. D. Ferrier, and J. Rogers, "Global optimization of statistical functions with simulated annealing," *J. Econometrics*, vol. 60, pp. 65–99, 1994.
[35] J. Koza, *Genetic Programming*. Cambridge, MA: The MIT Press, 1992.
[36] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, Machine Learning*. Reading, MA: Addison-Wesley, 1989.
[37] S. Haykin, *Neural Networks*. New Delhi, India: Pearson Education, 2001.
[38] X. Yao and Y. Liu, "A new evolutionary system for evolving artificial neural networks," *IEEE Trans. Neural Nets.*, vol. 8, no. 3, pp. 694–713, 1997.

[39] M. D. Ritchie, B. C. White, J. S. Parker, L. W. Hahn, and J. S. Moore, "Optimization of neural network architecture using genetic programming improves detection and modeling of gene-gene interactions in studies of human diseases," *BMC Bioinformatics*, vol. 4, no. 28, pp. 1–14, 2003.
[40] G. G. Yen and H. Lu, "Hierarchical genetic algorithm based neural network design," in *Proc. IEEE Symposium on Combinations of Evolutionary Computation and Neural Networks*, 2000, pp. 168–175.
[41] T. Takahama and S. Sakai, "Structural optimization of neural network by genetic algorithm with damaged genes," in *Proc. the 9th Int. Conf. on Neural Information Processing (ICONIP'02)*, 2002, vol. 3, pp. 1211–1215.
[42] X. M. Sun, C. M. Ren, Y. W. Wu, L. G. Ning, and J. R. Wang, "The design and application of neural network controller based on genetic algorithms," in *Proc. IEEE Int. Conf. on M/c learning and Cybern.*, Xian, 2003, pp. 1318–1321.
[43] J. S. Son, D. M. Lee, I. S. Kim, and S. K. Choi, "A study on genetic algorithm to select architecture of a optimal neural network in the hot rolling process," *J. Material Processing*, vol. 153–154, pp. 643–648, 2004.
[44] G. H. Kim, J. E. Yoon, S. H. An, H. H. Cho, and K. I. Kang, "Neural network model incorporating a genetic algorithm in estimating construction costs," *Building and Environment*, vol. 39, pp. 1333–1340, 2004.
[45] Md. M. Islam, X. Yao, and K. Murase, "A constructive algorithm for training cooperative neural network ensembles," *IEEE Trans. Neural Nets.*, vol. 14, no. 4, pp. 820–834, 2003.
[46] S. Kang and C. Isik, "Partially connected feedforward neural networks structured by input types," *IEEE Trans. Neural Nets.*, vol. 16, no. 1, pp. 175–184, 2005.
[47] P. A. Castillo, J. J. Merelo, A. Prieto, V. Rivas, and G. Romero, "G-prop: Global optimization of multilayer perceptrons using GAs," *Neurocomputing*, vol. 35, pp. 149–163, 2000.
[48] J. N. D. Gupta and R. S. Sexton, "Comparing back-propagation with a genetic algorithm for neural network training," *The Intl. J. Management Science*, vol. 27, pp. 679–684, 1999.
[49] R. S. Sexton and J. N. D. Gupta, "Comparative evaluation of genetic algorithm and back-propagation for training neural networks," *Information Sciences*, vol. 129, pp. 45–59, 2000.
[50] X. Yao, "A review of evolutionary neural networks," *Int. J. Intell. Syst.*, vol. 8, no. 4, pp. 539-567, 1993.
[51] N. B. Hui, V. Mahendar, and D. K. Pratihar, "Time-optimal, collision-free navigation of a car-like mobile robot using a neuro-fuzzy approach," *Fuzzy Sets Syst.*, (under review), 2004.
[52] D. K. Biswas, "Path planning of multiple robot working in the same workspace — Potential field approach," M.Tech thesis, REC Durgapur, India, 2003.

Biographies

Nirmal Baran Hui completed B.E. and M. Tech Degrees in Mechanical Engineering from Regional Engineering College, Durgapur–713209, India, in 2001 and 2003, respectively. He received University Gold Medal for securing the highest marks in the M. Tech program from the University of Burdwan, India. He is presently pursuing Ph.D. program at the Department of Mechanical Engineering, Indian Institute of Technology, Kharagpur–721302, India. His research interests include robot navigation, soft computing, image processing, dynamics and control of mechanical systems.

Dilip Kumar Pratihar completed B.E.(Hons.) and M. Tech Degrees in Mechanical Engineering from Regional Engineering College, Durgapur–713209, India, in 1988 and 1994, respectively. He was awarded the University Gold Medal and A. M. Das Memorial Medal for securing the highest marks in the University. He received Ph.D. in Mechanical Engineering from Indian Institute of Technology, Kanpur, India, in 2000. He visited Kyushu Institute of Design, Fukuoka, Japan, in 2000 and Darmstadt University of Technology, Germany, in 2001 (under the Alexander Von Humboldt Fellowship Programme), for his post-doctoral study.

He is working, at present, as an Associate Professor in the Department of Mechanical Engineering, Indian Institute of Technology, Kharagpur. His research interests include robotics, manufacturing science, genetic algorithms, fuzzy logic control, neural network control, and others. He has published around 60 technical papers. He received the Institution of Engineers (India) Medal for one of his technical papers, in 2002. He is an active reviewer of many international journals and conference proceedings.

He is a member of the Institution of Engineers (India).

Please update the information, if you have