Evolution of Cooperative Ensemble Neural Network Controller for Autonomous Mobile Robots

Mineki Okura*, Md. Shahjahan** and Kazuyuki Murase***

*Department of System Design Engineering, Graduate School of Engineering, University of Fukui
**Department of Electrical and Electronic Engineering, Khulna University of Engineering and Technology
***Department of Human and Artificial Intelligence Systems, Graduate School of Engineering, and Research and Education Program for Life Science, University of Fukui

Abstract: We here propose a new evolutionary approach with learning to create a variety of behavioral patterns in autonomous robots. The conventional evolution or learning is to optimize a cost function such as fitness function and error function. In practice, the robot encounters situations where exist multiple solutions having quite similar fitness or error values. The optimum solution is generally selected, while others are eliminated, even if the difference in the fitness or error is very little between the solutions. This causes an essential problem for behavior-based robots. Ideally, the robot should be able to select one of the behaviors by perceiving a slight difference in the sensory information, but the ability is lost. To overcome this problem, we introduced a structural learning during the evolution of neural network ensemble (NNE). Motor outputs were generated by summing outputs of component neural networks of an NNE, and they were trained to segregate each other by negative correlation learning between generations. In experiment, each component network exhibited different functionality, producing a variety of behaviors as a whole. The proposed evolution of NNE with negative-correlation learning thus can be a practical solution for the plasticity-stability problem in robotics.

Keywords Autonomous Robot, Genetic Evolution, Neural Network, Negative Correlation, Learning and Memory, Mobile Robot

1. Introduction

Artificial evolution has been widely studied in order to develop the control systems of autonomous mobile robots [1,2,3,4,5]. Control architecture, morphology, etc. of a population of robots are encoded on artificial chromosomes. The selected chromosomes computed by a fitness function are used to create a new population. The new population is then used in the following generation until a criterion is satisfied. A synergistic integration of evolutionary algorithms and Fuzzy theory was also studied in robotics [6,7].

The combination of artificial evolution and learning has recently attracted much interest [8,9,10,11]. Evolution and learning are two forms of biological adaptation that operate on different time scales. Evolution is capable of capturing slow environmental changes, while learning may produce adaptive changes in an individual [12]. It has been reported that learning boosts and speeds up the evolutionary process [13,14].

The attempts so far to evolve the control systems of autonomous mobile robots concern with many evolutionary approaches with a large variety of structures of neural networks. These include feed-forward networks [15] and dynamic time recurrent networks [16]. Although those network architectures are effective, they consist of single neural network (SNN) with and without a hidden layer.

Most of such works have concerned to find the optimum solution for the given task or in the given environment. If not, the network structure or functionality is determined based on the designer's knowledge in priori. As argued by Harvey [19], however, the search space of fitness in the real open environment is full of hills with similar fitness values. Autonomous robots working in open environment thus ought to have ability to distinguish situations and generate the best-suited behavior for a
particular situation. This may be considered, in another word, as the well-known plasticity-and-stability problem in learning and memory. When a robot has acquired an ability to generate a certain action in response to a particular sensing state, the sensing-action pattern should be memorized stably. If the following experience forces to learn a different sensing-action pattern in response to a slightly different sensing state, the robot ideally should be plastic enough to learn and memorize the new sensing-action pattern, not erasing the previous pattern. In the conventional evolution or learning, however, the robot often generalizes the past experience into the new one so that the past memory may be erased. The one with a higher evaluation becomes the winner and the loser is forced to diminish even if the difference in fitness is slight and both are important for the robot.

Recently ensemble neural networks (ENN) are developed and applied to solve many real world problems [17]. An ENN is a combination of several SNNs and proved its effectiveness against SNNs [18]. Although ENNs are applied in many real world problems, however, there are very few reports that deal with ENN in order to develop the control system of autonomous mobile robots as of our knowledge. A switching mechanism among several component networks (CNs) has been proposed [20]. However, it needs many IF THEN rules to meet a strategy and it is often difficult to write a suitable rules. Designing switching mechanism is often a difficult task. If cooperation is learned during evolution, it will be more reliable since it depends upon the output and input sensor values.

Therefore, we decided to apply the ensemble neural network controller (ENNC) to autonomous mobile robot, because we believe that ENN has potential to cope with the real world problems encountered by mobile robots, and show better performance than conventional control systems. We designed the controller by combining the evolution and cooperation among CNs. In the method, all individuals in a population are evolved according to a fitness function and each individual learns a cooperative function. The object of learning such a cooperative function is to produce new skills or abilities in the robots. The temporal correlation map which was computed from the collected outputs of CNs was used as an evidence for functional difference among networks. We also present the converged weights of the CNs after the evolution.

The advantages of adapting such a cooperative function are that one does not need to assume anything about the environmental structure to the robots and all CNs acquire new skills and information about the environment. Therefore, entire control system learns the environment effectively and completely. We observed characteristic movements and new skills in robots which were not found in robots with SNN. We also computed a robustness measure in order to validate the performance of the control system. We evaluate the proposed ENNC with the other control systems with ensemble network.

The paper is organized as follows. Section 2 describes the evolutionary and learning processes. Experiments and experimental conditions are outlined in Section 3. Results for static and dynamic environment are discussed in Section 4. The effect of few facts, user specified parameter and comparisons are described in Section 5. We discuss on the results in Section 6. Finally, we conclude the paper in Section 7.

2. Evolution of Cooperative Ensemble Neural Network Controller (ENNC)

This section describes the evolutionary process together with cooperative learning. Evolution is a process of selective reproduction and substitution based on the existence of a population of individuals displaying some variability [21]. Learning, instead, is a set of modifications taking place within each single individual during its own life time [21]. We outline the steps of the evolutionary algorithm and cooperative learning.

2.1 Implementation of evolutionary algorithm

We follow conventional evolutionary algorithm which has the following steps.

1. Create an initial population of $N$ individuals where each of them represent a control system.
2. Decode each individual to a corresponding control system.
3. Allow each robot to perform evaluation task for a fixed life time.
4. Reproduce a number of children in the current generation based on fitness.
5. Apply genetic operators to the children and obtain the next generation.

2.2 Implementation of cooperative learning

The object of learning during evolution is to produce specific behaviors. It is well known that there is no reasonable benefit if all networks in an ensemble behave
alike. In order to remove similar behavior, one can incorporate decorrelation function. As a result, we can expect that a particular network is responsible for a particular job.

We use negative correlation learning in order to cooperate among the CNs in a negative sense. In order to implement the negative correlation learning at the individual level of entire population, we have to correlate the outputs of each network in the negative sense. The negative correlation learning in the ensemble network proposed by Liu & Yao (1999) [22] can not be directly applied to the robot controller, since we do not have training data and we are not interested here to produce training data. We instead minimize the cooperation function for each individual.

Let \( M \) and \( F_j(n) \) is total number of CNs and the output of the network \( i \) at the \( n \)-th input sample, respectively. We define the average of the outputs as \( F(n) \),

\[
F(n) = \frac{1}{M} \sum_{i=1}^{M} F_j(n) \tag{1}
\]

We minimize the same correlation function as the one proposed by Liu & Yao (1999) [22] and can be defined as \( p_i(n) \) for the \( i \)-th network at the \( n \)-th input sample.

\[
p_i(n) = \frac{1}{2}(F_i(n) - F(n)) \sum_{j=1, j \neq i}^{M} (F_j(n) - F(n)) \tag{2}
\]

The partial derivative of \( p_i(n) \) with respect to the output of network \( i \) on the \( n \)-th input sample is

\[
\frac{\partial p_i(n)}{\partial F_i(n)} = \sum_{j=1, j \neq i}^{M} (F_j(n) - F(n)) = -(F_i(n) - F(n)) \tag{3}
\]

\[
= - (F_i(n) - F(n)) \tag{4}
\]

Thus in this learning the weight update rule becomes,

\[
\Delta w_i(n) = - \frac{\partial p_i(n)}{\partial w_i} = - \frac{\partial p_i(n)}{\partial F_i(n)} \frac{\partial F_i(n)}{\partial w_i} \tag{5}
\]

where \( w_i(n) \) and \( \Delta w_i(n) \) are the weight of the \( i \)-th CN and the amount of weight update at the \( n \)-th input sample, respectively.

We assume that the output of the each CN be the weighted sum of \( L \)-dimensional sensory inputs \( x_i \). That is,

\[
F_i(n) = \sum_{j=1}^{L} w_j(n)x_j \tag{6}
\]

Then, the amount of weight update \( \Delta w_i \) and the weight of the \( i \)-th CN at the \( i \)-th iteration \( w_i \) become as follows.

\[
\Delta w_i(n) = (F_i(n) - F(n))x_i \tag{6}
\]

\[
w_i = w_i^{t-1} + \frac{\lambda}{N} \sum_{n=1}^{N} \Delta w_i(n) \tag{7}
\]

where the parameter \( \lambda > 0 \) is the strength of learning, and \( N \) is the number of input-output data sets used at every iteration. It is clear from Eq. (6) that we need output of network and sensory input to the network to update the weights of each individual of the population. In the series of experiments below, since sensory input \( x_i \) takes the values between 0-1023 and network output also more or less varies between 0-1023, we rescale the input values between 0-15 in order to facilitate computation. We used \( N=10 \) throughout experiments.

An ensemble may contain a large number of CNs, and the algorithm proposed in this study is valid for multiple CNs of any number. However, in the following experiments we mostly dealt with two two-layered neural networks, and several illustrative examples of three and four CNs. This is because the illustrative observation and explanation would be obtained, and what is actually going on in the real system could be understood easily.

2.3 The ENNC procedure

The process how ENNC was implemented in a real mobile robot can be realized from the flow chart given in Fig. 1. In the first phase, each individual has to move in the environment for a life time of 5 seconds and after finishing all individuals, fitness values are evaluated and then in the second phase each individual obtained in the first phase moves for the same life time. This time no genetic operator was applied, instead, we collect 10 sets of input-output data for adapting cooperation function. After completing this phase, it enters into the next generation. This process repeats until the end of desired number of generation. The detail experimental protocol will be discussed in the subsequent section.
3. Experiments

In order to study the evolution with correlation learning, we used ENNC to develop a real autonomous mobile robot. In this section, we describe experiment and its conditions on a real mobile robot.

3.1. Mobile robot

A real mobile robot Khepera is used in this study. The structure and function of the robot has been well described elsewhere [23]. In short, its size is small: 55 cm in diameter, 30 mm in height, and 70 gm in weight. It consists of two boards: CPU board and sensory motor board. The CPU board contains a microprocessor, (Motorola MC68311) with 128 Kbytes of EEPROM and 256 Kbytes of Static RAM, an A/D converter for the acquisition of analogue signals coming from the sensory-motor board, a proportional-integral-differential (PID) controller for the motor control and a RS232 serial port together with power supply terminals which is used for data transmission to and from an external computer system and for providing electricity from an external power supply unit. The microprocessor can execute programs downloaded from the external workstation.

The sensory motor system uses two lateral wheels and supporting pivots in front and back. Each wheel is controlled by a DC motor. The motor can rotate in both directions by the output from the CPU and PID controller. There are 8 infrared (IR) sensors: six in front and two on the back as shown in Fig. 2. These sensors can detect objects within 3 cm or so by emitting infrared light and measuring its reflection. The values can be utilized by the microprocessor through A/D converter for such functions as generating motor control signals and analyzing the scenery perceived by the robot. Several complete modules such as vision and gripper modules can be added to the basic structure.

3.2. Experimental set-up

As shown in Fig. 3 (top), an aerial cable from the RS232C port of the robot was attached to a serial port of a LINUX PC workstation via a miniature rotating contact. The neural network construction and executing genetic operators were managed by the external workstation, while other processes such as computation of motor outputs from sensor readings, and monitoring and storing sensor and motor values were performed by the on-board microprocessor. The environment used to accomplish the evolutionary processes consisted of a square area with four obstacles as shown on the top of Fig. 3. The size of the area was approximately 60×60 cm. White non-glossy card-board was pasted on straight wooden bars. They were used to make the walls of the path and obstacles in the environment. The height of obstacles was made of short so that proximity sensors of the robot can detect them. The environment was illuminated from above by two overhead fluorescent lights and a 60 watt bulb. The intensity of 60-watt bulb can be adjusted by a selector. We use this bulb to change the environmental condition.
3.3. Control network and encoding

A simple two-layered feed-forward neural network was used as a CN of the entire ensemble. Two to four CNs were used as a control system to produce motor control signals. The example of two CNs is shown in the bottom of Fig. 3. Two control signals for motors were produced by summing the values from IR and/or vision sensors. That is, each output was generated by

$$S'_m = S_b + G \sum_{j=1}^{L} w_j x_j$$

(8)

where, $S'_m$, $S'_b$, $S_b$, $G$, $w_j$ and $x_j$ represent the output value to the motor, the thresholded output value to the motor, the base navigation speed of the motor, global gain, connection strength and input sensor signals value, respectively. The value of $S_b$, $G$, and $L$ were set to 5, 1/1600, and 6, respectively. The global gain determines the sensitivity of the modulation signal from sensors. The final output signals to the motors were generated by averaging the output of the CNs. For example, two left outputs of two CNs are averaged and finally it was given to the left motor and so on for right motor.

A direct coding scheme was used to encode an entire ensemble network. Each weight was encoded on a gene of five bits where the first bit determines the sign of the weight and the remaining four bits its strength. We used front six sensors. So each CN has a total of 12 weights. Thus for two CNs a total of $12 \times 5 \times 2 = 120$ bits are necessary to represent a chromosome.

The robot can sense reflected ray from the environment. The range of input sensors lies between 0-1023 as shown in Fig. 4 for instance. The values were taken for only sensors 3 (top) and 4 (bottom) from a simple navigation with obstacle avoidance. Sampling was made at every 0.1 seconds.

3.4. The task and fitness function

The task given to the robot was to navigate in the environment by avoiding obstacles. The evolution and correlation learning were carried out in the consecutive steps. That is, after each generation each individual was taken and used to navigate in the environment for a certain time. The process was repeated for all individuals and then evolutionary process went into its 2nd generation. We used the most widely used fitness function as defined
below. This fitness function has been a standard for developing the obstacle avoidance and navigation capability [24].

\[
f = \sum_i V(t)(1 - \Delta v(t)) \left(1 - \sum_j s_j(t)\right)
\]  

(9)

here, \( V \) is the average rotation speed of two motors and is used to reward fast controllers. \( \Delta v \) is the absolute value of the algebraic difference between the signed speed values of the motors (one direction is positive and the other is negative) and is used to reward straight locomotion. \( s_j \) is the proximity measure of the \( i \)th sensor and is used to punish the robot each time it sensed obstacles. Here, \( \Delta v \) and \( \sum s_i \) were standardized so that the maximum values became 1.

3.5. Genetic parameters

A simple genetic algorithm was used to determine the values of connection weights in the neural network. Genetic parameters used in the experiment are listed in Table 1. During the evolution, the robot changes its position randomly according to Braitenberg algorithm [25] in order to avoid the effect of previous individual’s action.

<table>
<thead>
<tr>
<th>Table 1. Genetic parameters used for evolution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Population size</td>
</tr>
<tr>
<td>Number of generation</td>
</tr>
<tr>
<td>Cross-over probability</td>
</tr>
<tr>
<td>Mutation probability</td>
</tr>
<tr>
<td>Elite preservation</td>
</tr>
<tr>
<td>Bit per weight</td>
</tr>
<tr>
<td>Chromosome size</td>
</tr>
<tr>
<td>Final weight range</td>
</tr>
<tr>
<td>Life time (indiv)</td>
</tr>
<tr>
<td>Sample rate</td>
</tr>
</tbody>
</table>

3.6. Experimental protocol

The interaction between robot and environment is an important issue during navigation. A robot collects information about its environment with the help of its sensors, which is then fed as input to the robot controller and the output of the controller is realized through motor action. Depending on the output of the controller, the motor runs to move the robot starting from a initial position.

We implemented evolution and cooperative learning in the subsequent two stages. In the first stage, evolution occurred on the entire population and in the 2nd stage cooperative learning was performed in each individual.

(a) 1st stage: In this stage, each chromosome corresponding to an individual in a population was generated by the external workstation, and loaded down to the on-board microprocessor. After decoding into the corresponding neural network, the robot was allowed to move in the environment with the neural network’s input and output values as well as encoder values sampled at every 0.1 s. These values were then uploaded to the workstation for fitness evaluation. This procedure was repeated for all individuals. When all individuals had been tested, three genetic operators - selective reproduction, cross-over, and mutation - were applied to create a completely new population of the same size.

(b) 2nd stage: After the 1st stage, the same procedure was started for the sake of cooperative function adaptation. Due to computational complexity, we separated this stage for collecting input-output data from the environment. Each chromosome of the population obtained in the 1st stage now loaded down to the on-board microprocessor and robot was allowed to move in the environment. No genetic operators were applied this time, rather the robot only moves and collects 10 sets of input and corresponding output data for each chromosome corresponding to an individual. After finishing all individuals, all these values were uploaded to the workstation in order to adapt the cooperative function. Then, weights of all individuals were updated according to Eq. (7). After these two stages, the evolutionary process started for the next generation.

4. Results

This section reports the results and analysis of applying the cooperative controller architecture in the robot. Experiments were done in static and dynamic environments. The static environment is meant to be an area where environmental conditions such as temperature, light intensity, number of illuminating light, etc., were fixed, whereas environmental conditions changed in a dynamic environment. We investigated the responses of robot controller for these two cases.
4.1. Evolutionary and learning process in a static environment

4.1.1. Evolutionary process

Fig 5 shows the evolutionary process of the proposed architecture in a static environment. A total of 10 individuals were used in the population. The average fitness of the population was improving in the course of evolution. Visual inspection suggests that the fitness function was not influenced by correlation adaptation. The robot became able to navigate in the environment avoiding obstacles within 30-40 generations.

![Fig 5. Average fitness values as a function of generation in a static environment](image)

4.1.2. Temporal correlation map

Temporal correlation map (TCP) is defined as the cross correlation between two left outputs for left motor and between two right outputs for right motor. We show TCP in Fig. 6 for all individuals of the population after each generation. For each individual, we collected a series of 10 outputs for both CNs and computed their correlation. Thus, at the end of each generation, there were 10 correlations for 10 individuals and averaged over all individuals during every generation, as plotted in the figure as a function of generation. In the beginning, the correlations were roughly random, that is, mixture of positive and negative values. We can clearly observe that correlations gradually became negative at the latter generation due to the correlation term we used. We updated the weights by a small amount, thus it takes a little time to be negatively correlated each other. We computed one pair of cross-correlation, one for two left outputs for left motor and the other for two right outputs for right motor, by

\[
C_{AB} = \frac{1}{100} \sqrt{\frac{\sum_t (F_A(t) - \bar{F}_A) (F_B(t) - \bar{F}_B)}{\sum_t (F_A(t) - \bar{F}_A)^2 \sum_t (F_B(t) - \bar{F}_B)^2}}
\]

where \( C_{AB} \) is the cross-correlation between network A and B. \( F_A(t) \) and \( \bar{F}_A \) are the output of network A for \( t \) input sample or pattern and average over all outputs of A, respectively. We summed up all correlations over all individuals of the population after each generation by,

\[
C_{sum} = \frac{1}{100} \sum_{i=1}^{10} C_{AB}^i
\]

![Fig 6. Temporal correlation map for both outputs for both motors in a static environment](image)

4.1.3. Functional weight

In order to see the influence of the cooperative function, we recorded the weight distribution. In Fig. 7, weights values are plotted when two CNs were used. The numbers 1-6 and 7-12 correspond to weights from sensors 0-5 to left and right output units, respectively. The weights of the two component networks clearly exhibited functional difference. Three plots are shown in the course of evolution. From generation 0 to 50, weights were not functional. After 75 generations, both of the networks achieved functionally different weights. In fact, they are negatively correlated.

The essence of this learning is that the movement of the robot could be realized with the two complementary functioning weights. This may lead the robot different movement or different path following, which is impossible with SNN. The detail will be discussed in section 5.4.

The converged weights of the two CNs were of the form \( W_\alpha = (w_1, -w_2) \) and \( W_\beta = (-w_1, w_2) \) approximately, although several weights of two CNs had identical sign.
This form of the weight representation is reasonable, since we used cooperation in the negative sense. We noticed that few weights of the two networks had identical sign. We suggest that this is due to the evolutionary pressure, not only cooperative learning itself, because there were two driving force in ENNC; evolution and learning. Therefore, correlation among the two sets of weights may not be 100% complementary each other due to evolutionary pressure.

Figure 8 shows the weights when three CNs were used. Similar to the case of two CNs, weights of CNs became complementary each other after generations. Figure 9 shows an example of weights with four CNs. Here we exhibited only the weights of best individual at generation 100. As apparent from the figure, weights of CNs became complimentary each other.

Fig 7. Converged weights during the course of evolution in a static environment for two component networks (CNs). There were ten individuals at every generation.

Fig 8. Converged weights for three CNs. 10 individuals at every generation.
4.1.4. Divergence of Weights

When three CNs are used, for example, the divergence of weights among CNs can be measured by the following equation.

\[ D = \sqrt{d^2 + e^2 + f^2} \]

where \( d, e \) and \( f \) are the weight vectors as such,

\[ W_A = (w_{A1}, w_{A2}, ..., w_{A4}) \]
\[ W_B = (w_{B1}, w_{B2}, ..., w_{B4}) \]
\[ W_C = (w_{C1}, w_{C2}, ..., w_{C4}) \]

Figure 10 shows how the divergence was increased during the evolution for the cases when two to four CNs were used. Each trace was the average of 6 independent trials. As evident from the figure, the divergence grows after generations.

4.2. Evolutionary and learning process in a dynamic environment

4.2.1. Evolutionary process

A dynamic environment was setup by changing the intensity and position of a 60 Watt bulb. We needed approximately 13.5 hours for 300 generation. After each 30 minutes, we changed one intensity out of three positioned on a side of the square area. After finishing all intensity options, we moved the light to the next right hand side of the square area as shown in Fig. 11(a). In addition a 2 x 3 cm rectangular obstacle was suddenly placed in the environment three times for each 100 generation; each time the obstacles stayed for 30 minutes (11(b)).

The fitness values are plotted as a function of generation as shown in Fig. 12(a) for the environment of Fig. 11(a). Fitness increased in the course of evolution. However, fitness decreased between 60-70 and 160-170 generation and again increases for the rest of generation. The reason is the dynamics of environment. The robot received much noise during the navigation due to light adjustment. Still the robot navigated in the environment by avoiding obstacles.

Fig 10. The divergence among component networks.

Fig 11. Model of dynamic environment

Fig 12. Average fitness as a function of generation in a dynamic environment
4.2.2. Temporal correlation map  

The temporal correlation map is shown in Fig. 13. The outputs for left and right motors are shown. Initially the correlation was random, gradually it converged to negative, although not for all individuals in the population. Because the real robot received much noise from the dynamic environment and we used a small amount of cooperation strength, $\lambda = 1/18$ for the evolution. This was the reason why learning took long time for converging negatively correlated networks. We will discuss the influence of parameter $\lambda$ on evolution in the subsequent section.

![Fig 13. Temporal correlation map for dynamic environment](image)

4.2.3. Functional weight  

The converged weights for two networks are shown in Fig. 14 at generation 10, 50 and 100. It is clear from the figure that the network learned different set for each network. At generation 10, the weights were rather random, at generation 50, they tended to be functionally different from each other. We can clearly see that weight set for network A became approximately opposite to that of network B after 100 generation. Therefore, we can conclude that if network A is responsible for left turn, network B should be responsible for right turn or vice versa. Moreover, each network could be specialized on a particular aspect of the environment.

![Fig 14. Learned functional weight for two networks A and B in a dynamic environment](image)

5. Analysis  

In this section, we discuss the influence of user specified parameter, $\lambda$, the responses of the controller subject to external disturbance, and comparisons with other controllers.

5.1. Effect of strength of cooperation  

The parameter $\lambda$ controls the strength of cooperation between CNs. The correlation between the networks becomes negative at the increase of $\lambda$. The response of the controller is influenced by this value. In order to show how the controller behaves, we run the program six times with six different values of $\lambda$ as shown in Fig. 15.

In the figure, we observe that the fitnesses were not increasing for $\lambda = 1/2$ to $\lambda = 1/8$. In this range,
however, the correlation of two motors become mostly negative. This indicates that maintaining always 100% negative correlation of both motors is harmful for the controller if the controller is designed with this fitness function. However, if we relax the value of $\lambda$, the fitness was increasing as shown in the figure for $\lambda = 1/12$ to $\lambda = 1/20$. Therefore, we can conclude that suitable value for $\lambda$ lies at $\lambda < 1/8$. We used $\lambda = 1/18$ in our experiments.

Fig 16 shows the total strength of correlation of two motors as a function of $\lambda$. The total strength of correlation was computed by summing all correlations of two motors and then divided by $2 \times 10$ to fit the correlation between 0 to ±1. The figure shows that the correlation became strongly negative at higher values of $\lambda$. This is also harmful for overall performance of the controller. However, we noticed two situations during the experiments.

(a) Robot could navigate in the environment avoiding obstacles reliably for $\lambda = 1/12$ to $\lambda = 1/20$ or more smaller values. The achievement of negative correlation in these values periodically changed, i.e., once both correlations for both motors (left and right) became positively correlated, again one correlation (left or right) became negatively correlated, then both correlations for both motors became negatively correlation and then somehow negativeness was lost and the process was repeated throughout the evolutionary process. We suggest that there were two possible reasons behind this. Firstly, the extracted input-output data are highly dynamic and incomplete. Secondly, input sensory signals are chaotic even in a static environment [26]. Therefore, correlation might be affected periodically since it is a function of motor output signals.

(b) Robot could navigate in the environment with many intermediate delay for $\lambda > 1/8$. Intermediate delay means a short delay subject to an obstacle. When it found any obstacle, then it waited for a short period (2-3 seconds) to decide where to move. Most importantly the fitness was not increasing, since the fitness function incorporated the distance covered by it and the distance covered by robot was automatically reduced since it stayed as many times as it found obstacles. Therefore, a suitable value of $\lambda$ should be less than 1/8.

5.2. Responses of controller due to external disturbance

One of the requirements of robust controller is that it should be unaffected by the external disturbance. In order to prove the effectiveness of the controller against external disturbance, we setup an experiment that consists of a sudden intentional human intervention during the navigation. That is, we place an white object in front of robot after every one minute. We observed that all correlations became negative after 21 generations. We plot the fitness values of these 21 generation as shown in Fig. 17. It is clear that the responses were quite robust against disturbances. In addition, a good navigation capability was achieved within 21 generation as well.
of robustness are used. A SNN controller means a controller with one network with front six sensors. Therefore, a SNN contains 12 connections. All other parameters and GA operations are similar to ones as mentioned earlier. An uncooperative controller is meant to be simple ensemble with no cooperation among the CNs.

5.4.1. Trajectory

In this section, we describe the trajectories of the robot movements obtained with various conditions. We draw the trajectory of the evolved robot for a fixed time between 50 and 51 generation for SNN and the proposed cooperative controller. This is because we need enough time (generation) to be elapsed for developing the actual behavior.

Fig 19(A) shows a schematic drawing when SNN was used. The robot follows a path abcedfijkl with three collisions with obstacles and walls indicated by arrow. The behavior of obstacle avoidance is almost achieved with no special characteristic movements.

Fig 19(B) exhibits the case where ENNC was used. The robot follows abcedfijkl without any collision with walls and obstacles. The important properties that the robot exhibited are few special skills which are discussed in section 5.4.2.

The computation time for ENN controller is almost two times than SNN controller. This is because ENN controller requires additional 2nd stage for the collection of input output data. However, the convergence time for ENN controller to achieve obstacle avoidance behavior is less than the SNN controller in terms of number of generation.

The ENN controller achieves obstacle avoidance behavior within 20-25 generation, while SNN requires more than 40-45 generation as evident in the Fig. 20 for average speed at each generation. This shows that ENN produces regular and smooth speed after about 25 generation, while SNN controller does not give us such behavior. Most importantly, even though SNN achieves obstacle avoidance behavior, it hits the obstacle quite often.

5.4. Comparison

In this section, we compare our proposed controller with SNN controller and uncooperative controller. For comparison, trajectory followed by a robot and a measure
5.4.2. Special skills from cooperative learning

The object of learning in evolution is to achieve new skills in robots. In this section, we describe some skills exhibited from robots such as path repetition, right angle turning.

- **Path repetition**: Path repetition means the robot repeats a path as shown by a schematic drawing in Fig. 21(A). The robot R starts to walk along the path \( ab \) and arrives the wall W3 as shown. Then again starts for following the same path from which it came, i.e., \( ba \). We can interpret this round trip as follows. There is a competition occurs among CNs due to cooperative learning. This competition usually happens in ensemble network trained by negative correlation learning as evidenced by one of the authors and others [27]. This competition may lead the robot controller to follow a round trip. Since every CN wants to know about the environment.

- **Repeated path escaping**: When it makes a complete round trip, it moves to a new place automatically as shown in Fig. 21(C). This may be due to the evolutionary pressure, since the fitness function involves a penalty when facing obstacles.

- **Right angle turning**: What happen if the robot comes to an obstacle with right angle. It shows an intelligent turning. First, it moves back after facing obstacle and then faces the obstacle again and turns on its right or left side as clearly shown in Fig. 21(D).

5.4.3. Robustness of the controllers

This section compares the performance of two controllers, SNN and ENN. To evaluate and compare the behaviors produced by by different controllers, it is important to find dimensions that indicate the quality of the performance on pertinent aspects of the task and show up the difference between controllers.

In this study, three dimensions were chosen to compare the performance of the robot controller. They were (i) the speed of completing the task; (ii) collision rate; (iii) robustness. The following three quantitative measures were employed in order to compare robot control systems.

- **Sensory-motor loop values (SLV)**: The time taken for producing the robot behavior is partially dependent on a host computer and the frequency of data transfer from a host computer to a robot. In order to make a fair assessment, a measure in time step of robot was used.
each time step of a robot can be characterized by a sense-process-act step. In this step, incoming sensory inputs are transferred into motor outputs. This is called sensory-motor loop. In this study, each sensory motor loop took about 0.1 seconds. The number of sensory motor loops is used to measure how long the robot took, in robot time, to complete the task.

- **Collision rate (CR):** A physical parameter that indicates the number of collision with obstacles and walls was recorded over a fixed period of time for SNN and ENN controller. The controller with less number of collision is said to be good as a whole.

- **Robustness (R):** A robot controller may be said to be robust when a robot needs a minimum number of sensory motor loops with less collision rate. That means,

\[
R = \frac{1000}{SLV \times CR}
\]  

(12)

These three parameters were not used in the fitness function. This measure is a good criteria in order to evaluate the robustness of the control system. This kind of robustness measure can be found in [28]. A larger value of R indicates that the controller is better than smaller value.

The best individual of SNN of a run and the best individual of ENN of a run were taken in the experiment and tested for 15 minutes. The CR was taken collision/15 minutes. The experiment was repeated for five times. The average results were listed in Table 2. The score for robustness is higher with ENN than SNN controller. Therefore, the controller with ENN is more robust than controller with SNN. It is important to note that SNN controller requires less number of SLV, where as ENN needs more number. The reason is that the speed of ENN controller was averaged of two CNs. So the robot observes an object slightly longer than SNN controller. But the robot with ENN controller does not hit the obstacles and walls as frequently as SNN controller. Therefore, ENN controller is seemed to be better than SNN controller as a whole.

### Table 2. Performance of SNN and ENN controller in terms of SLV and CR

<table>
<thead>
<tr>
<th></th>
<th>SLV</th>
<th>CR</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>SNN</td>
<td>153.57</td>
<td>2.33</td>
<td>2.79</td>
</tr>
<tr>
<td>ENN</td>
<td>188.43</td>
<td>0.33</td>
<td>16.08</td>
</tr>
</tbody>
</table>

### 5.4.4. Relationship with other methods

There are several methods that use collaborative neural network as control systems [20,29,30]. Hartono et al. [20] implemented a neural network ensemble model, consisting of a number of independent neural networks. The ensemble can automatically train each of its members to acquire a strategy e.g., a function that maps sensory input into a certain action of the robot, and executes strategy switching to deal with unpredictable environment.

On the other hand, our method implements two networks in the ensemble which are functionally different. Therefore, it does not need the switching mechanism or mapping function. It automatically adapt by minimizing a cooperative function during evolution. We use the evolution and learning together, while their method [20] used only learning.

Goldberg implemented a ‘collaborative control’ where two ensembles of input sources (sensors) share control of the path [29]. An output from one component member network is used to move the robot to the right and other output from the remaining network is used to move the robot to the left. On the other hand, the sum of outputs from all member networks are used to control the movement of the robot as a whole in our proposed method.

### 6. Discussion

Designing a suitable controller for an autonomous and intelligent mobile robot in an unknown and changing environment is a great challenge to the researchers. Since ENN is a recent technique and there are few reports that addressed the issue of designing control systems using ENN, we develop control system by combining two processes; evolution of ENN and cooperation among CNs. The novelties of the proposed method are to incorporate the ensemble architecture and cooperative function that play a central role to make entire control system expert on the real world environment.

The ENNC is the first attempt in which we utilize evolution and cooperative learning together in a real mobile robot, as of our knowledge, although there are few reports with only ensemble networks. The ENNC explores the cooperation based learning during the course of evolution. This kind of expert learning is not possible for two or three layer SNNs. Unlike other attempts, ENNC implements evolution with cooperative learning which
promotes the interaction of the CNs and exhibits special
skills on its movement.

The evolutionary process and learning process were
separated in the subsequent two stages in order to avoid
computational complexity. We needed input and output
data to update the cooperation learning. In the second
stage, a robot navigate in the environment for a life time
and we collected input output data during its life time and
updated the weights of the robot after finishing its life
time in a batch mode.

Experiments are done extensively in static and
changing environments. Although the sensory informa-
tion received through sensory organs is dynamic in a static
environment because of the behavior of the robot itself,
we have shown the responses of the controller in a static
and dynamic environments. The dynamics of the envi-
ronment were maintained by placing a time dependent
obstacle and intensity adjustable bulb in the environment.
The controller was also behaved well in response to the
external disturbaes.

The cooperative controller is better than uncoopera-
tive controller and SNN controller in the sense that such a
cooperation makes the controller with less collision rate.
Therefore, the controller shows intelligent motion subject
to sensory information. Thus a robot knows the envi-
ronment completely and moves in it carefully.

Since the evolution is known to be a very slow
process in adaptation, we use relatively smaller number
of individuals (ten individuals). We found those individuals
to effectively capture the environment and robot can na-
vigate in the environment by avoiding obstacles without
any human intervention. This reduces much computa-
tional effort.

The robots achieve good movement in the envi-
ronment. One important issue of evolutionary robotics is that
how well the robot moves smoothly in the environment.
The evolution captures the spatial movements of the robot
and the cooperative learning captures temporal sequence
of sensory information. Therefore, we can say that the
robot controller becomes familiar on the environmental
structure effectively.

The ENN controller is seen to be robust in terms of
collision probability. The SNN controller may be faster
than ENN. But it moves with many collision with walls
and obstacles. There is little benefit if the robot hits the
obstacles and walls even after evolution. The ENN con-
troller may be little slower than SNN, the speed is the
average of the CNs. Therefore the robots treat the envi-
ronment carefully and moves intelligently almost without
hitting obstacles and walls.

It is, however, unclear that which specific job was
done by a CN during navigation. This is impossible to
search specially where the environmental structure is
constant and sensors values are noisy and chaotic. We
found competition among CNs during navigation. This
shows that both CNs want to know a particular job care-
fully and completely. This will be helpful when a SNN
controller acquires insufficient information about the
environment. However, we believe that round trip is an
indication of typical outcome of functionally different
CNs, where forward trip should possibly be governed
from a CN and backward trip by other one.

The performance of the controller was influenced by
the parameter $\lambda$ as discussed before. Users need to
select a suitable value of it. The robustness of the con-
troller was evaluated with the fitness function, since it
contains two parameters which show its robustness. We
use two CNs because it is easy to visualize the repre-
sentative weights and correlation map for smaller number
of CNs. Using many CNs is an aspect that restricts repre-
sentative visualization.

7. Conclusions

We describe a method to design control system of
autonomous mobile robots using ensemble neural net-
work. A cooperative function was used in order to coo-
perate between the CNs in the negative sense. Experi-
ments were carried out on a real autonomous mobile robot
in a real environment. The effectiveness of the controller
was tested in a static and dynamic environment. Signifi-
cant results were obtained during the evolution and after
the evolution. Functional representation of weights and
correlations were appeared in the controller. The left and
right output pair were approximately functionally different.
As a result, each networks wanted to become expert
on the environment in a different manner by competition.
Specific skills were observed during navigation such as
path repeating, careful right angle turning etc. The tra-
jectories exhibited the way how the controller drives the
robots in the environment without the knowledge about it.
The influence of the user specified parameter was also
well described. The controller was compared with SNN
and the uncooperative controller. The performance of the
proposed controller was better than both of them. The use
of more complex cooperative function with the evolution
may be left for future study.
To produce autonomy, it is essential to form saddle points where slight difference in the sensory information makes significant change in the behavior. The example is that the robot faces an obstacle in front where both right and left turns are sufficient to avoid collision. Losing one of the other results in less adaptation ability to the environment. Although the Subsumption Architecture can form such saddle points, the designer of the robot has to fabricate the structure with his prior knowledge. By machine learning, either with bayesian methods or with evolutionary methods, it is difficult to articulate such saddle points. It rather erases them during the course of learning and/or evolution. Less frequently-occurred sensory-motor patterns are subsided by more frequently occurred events if the epochs are resembled each other. By the proposed method with negative correlation among component networks, such less-occurred epochs can be stored in separate components and thus saddle points could be formed. The experiments with a real mobile robot, we proved that complimentary networks were formed during the course of evolution. This in turn exhibits the ability to form saddle points during evolution. Further study including on the efficacy of the proposed method in comparison to others, such as embedding nonlinear dynamics into recurrent neural networks, is needed for the wider application.

References


