2007 IEEE International Conference on
Robotics and Automation
Roma, Italy, 10-14 April 2007

Enhancement of Self Organizing Network Elements for Supervised Learning

Chyon Hae Kim, Tetsuya Ogata, and Shigeki Sugano

Abstract— We have proposed self-organizing network elements (SONE) as a learning method for robots to meet the requirements of autonomous exploration of effective output, simple external parameters, and low calculation costs. SONE can be used as an algorithm for obtaining network topology by propagating reinforcement signals between the elements of a network. Traditionally, the analysis of fundamental features in SONE and their application to supervised learning tasks were difficult because the learning method of SONE was limited to reinforcement learning. Here the abilities of generalization, incremental learning, and temporal sequence learning were evaluated using a supervised learning method with SONE. Moreover, the proposed method enabled our SONE to be applied to a greater variety of tasks.

I. Introduction

The use of autonomous robots is expected to have application to the exploration of space and the deep sea. However, a robot's creator cannot determine in advance all appropriate behaviors, since the robot will encounter unknown situations. One solution to this problem would be the use of learning methods in robots.

The required conditions for such a learning system are very strict. First, the learning system requires robustness and autonomy to cope with various environments and tasks. To achieve this condition, the external parameters of the learning system must be simple in order to avoid hard tuning. Moreover, the calculation cost must be low enough that appropriate behaviors can be learnt within a time span suitable for the changing situation. Thus, the robot requires a learning system that meets the following three criteria:

1) Autonomous exploration of effective outputs
2) Simple external parameters
3) Low calculation costs

To meet the first condition of autonomous exploration of the effective outputs, traditionally many learning systems have been proposed based on Reinforcement Learning (RL) or Genetic Algorithms (GA). Several methods have also been proposed to meet the second condition, simple external parameters.

Approaches based on RL include Multi-Layered Reinforcement Learning (MLRL)[8] and Direct-Vision-Based Reinforcement Learning (DVB-RL)[9]. In the field of RL, the learning system requires the division of the state space for each task and environment. So, there is an external parameter used to choose a dividing method. MLRL is able to divide the state space automatically by the use of a network based on Q-learning modules. However, MLRL requires the choice of a transaction of the inputs for each task and environment. On the other hand, DVB-RL introduces a neural network as a reinforcement learning system, and it realizes a direct connection between the network and sensor-motor I/O. Thus, DVB-RL does not require the division of the state space or transaction of the inputs. However, the topology of the neural network in DVB-RL must be tuned for each task and environment. Therefore, the second condition is not met by these learning methods.

NeuroEvolution of Augmenting Topology (NEAT)[10] has been proposed as a learning method based on GA. NEAT does not require division of the state space, transaction of the inputs, or determination of the network topology, because the parameters in the neural network are determined evolutionarily using GA. However, the methods based on GA require a time span in which to evaluate phenotypes. When a long span is introduced into the system, the determination of the learning span for each task and environment is not required. However, learning costs a lot of time. On the other hand, when a short span is introduced, the learning span must be tuned according to the task and environment. Therefore, either the second or the third condition (parameter or time cost) is not met.

Real-time NEAT (rtNEAT) [11] and NEAT+Q[12] have been proposed as real-time or online learning methods based on NEAT, but their application to an autonomous robot is difficult. While rtNEAT realizes the online evolution of a group of agents in a video game, this method is useful only for groups, and its application to a single robot is difficult. NEAT+Q has also been proposed as a method for real-time learning by NEAT, but it requires a division of the state space for combining NEAT and Q-learning.

We have previously proposed Self-Organizing Network Elements (SONE) to overcome these problems. In an experiment with a simulated robot, the effectiveness of the method has been confirmed [19], [20]. The network is obtained by RL with SONE and it works effectively in a simple robot. However, when using SONE in a more complicated robot, it is preferable to prepare the network before the activation of the robot, so as to provide some

Chyon Hae Kim and Shigeki Sugano are with the Department of Mechanical Engineering, Waseda University, 3-4-1 Okubo, Shinjuku-ku, Tokyo, Japan. {tennkai/sugano} @ sugano.mech.waseda.ac.jp
Ogata Tetsuya is with the Graduate School of Informatics, Kyoto University, Yoshida-honmachi, Sakyo-ku, Kyoto, Japan. ogata @ i.kyoto-u.ac.jp

basic control rules before RL. As a method of obtaining such a network, we propose an enhancement of SONE to provide supervised learning.

The ability for generalization, incremental learning and temporal sequence learning were evaluated using the method proposed in this paper. The obtained results suggest that SONE can be used for various supervised learning tasks. In section two, we describe SONE. In section three, we describe the proposed implementation of supervised learning. In section four, we outline the experiments carried out in this research. In section five, we present our discussion, and finally in section six, we present our conclusions and outline our future work.

II. Self-Organizing Network Elements

In traditional RL methods, the division of the state space, transaction of the inputs, or the network topology must be tuned by the creator of a robot according to the task or environment. On the other hand, in traditional GAs, there is the problem of evaluation span. As a solution to these problems, we have proposed Self-Organizing Network Elements (SONE).

As with DVB-RL, the use of a network avoids the division of the state space and transaction of the inputs, so we introduced a network system, which can be directly connected to the robot I/O. Next, the learning method for the network topology is important to consider. In traditional GAs, there is a problem of evaluation span, because the whole network is evaluated. When the robot I/O is directly connected to the network, the evaluation of the whole network is related to the evaluation of complete behaviors for the robot in response to the whole environment. So, the evaluation span is determined by the span of the whole task that the robot has to learn. On the other hand, SONE are able to evaluate each element in the network. In general, the activation or deactivation of an element occurs for a particular situation encountered during the whole task. Thus the evaluation of an element is only related to behavior in a particular situation. As such, SONE do not require the duration of the whole task to evaluate the topology. Using this feature, real time and online learning of network topology were achieved with SONE.

The evaluation of elements is also encountered in the field of neural networks[3]. However, it is difficult to accomplish reinforcement learning because almost all of the evaluation methods are based on error and not on reinforcement signals. In our SONE, learning is realized by reinforcement signal propagation rules. The evaluation of any element in the network can be made using these rules. Then, the generation or self-destruction of any element in the network can be handled on the basis of the evaluation.

The one-step calculation cost of SONE is the order of the number of nodes (V) plus the number of Links (E), $O(V+E)$. In addition, SONE can allow online reinforcement learning and can create their own topology without the tuning of learning parameters. Therefore, all the functions previously described can be accomplished using SONE.

We applied SONE to four logic circuit elements: or-node, and-node, inverted link, and non-inverted link. Paired with each of these elements, we introduced four test elements. In this paper, we present only the implementation of the or-node and non-inverted link as examples.

A. Or-node

The or-node (Fig.1) has several (N) links and a test link, and each input is determined from $X(1)-X(N)$and X_T. Every element in a self organizing network has three phases in the calculation of its behavior: the action phase, the propagation phase, and the restructuring phase. In the action phase, an or-node only executes the operation of "OR" for its output $Y(=\bigcup_{i=1}^{N} X(i))$. In the propagation phase, an or-node propagates reinforcement signals to the links based on the propagation rules (Cases 1-5, TableI). Each case has its conditions described by $X(1)-X(N), Y, N$, and the number of True (T) signals from links, N_T. Based on these cases, the or-node generates reinforcement signals as follows: $\mathrm{R}_1(\mathrm{k})$ for a link and $\mathrm{R}_2(\mathrm{k})$ for an input-side node of the link. The or-node also propagates R_1 to the test link ($R_2 = 0$). However, in this case, calculation must be done under the assumption that the test-link is the same as other links. When Y is not the same for this assumption as it is in the real state, $R_1$is substituted by $-R_1$. In the restructuring phase, the or-node self-destructs when the number of its output-side links is 0, and promotes the test-link as a new link when the stored reinforcement signal in the test-link is higher than a threshold ($TH3$). Moreover, the or-node creates a new test-link between random nodes when the existing test-link self-destructs or is promoted as a new link.

B. Non-inverted link

A non-inverted link has a test node with two test links, TL1 and TL2 (Fig.2). TL1 is connected to the same node to which the non-inverted link is connected such that TL1 represents the same operation as a non-inverted link. TL2 is randomly connected to a node in the network.

In the action phase, a non-inverted link only outputs as Y its input X ($Y = X$). In the propagation phase, the non-inverted link propagates a reinforcement signal to the test-node based on the propagation rules (Cases 1-4 TableII). Each case has its conditions described by Y, the output of the test node Y_T, and the reinforcement signal from a node R_1. Based on these cases, the non-inverted link generates a reinforcement signal R_T to the test node. In the restructuring phase, the non-inverted link stores R_1 as its evaluation value R ($R \leftarrow R + R_1$). If R is less than 0, the non-inverted link self-destructs. Moreover, when the reinforcement signal stored in the non-inverted link is higher than a threshold (TH1), and the reinforcement signal stored in TL1 and TL2 is higher

TABLE I
The generation rule for the reinforcement signal in an or-node

$Case1:$	$(Y = T) \wedge (X(k) = F)$
	$R_1(k) = 0, R_2(k) = 0$
$Case2:$	$Y = F$
	$R_1(k) = R/N, R_2(k) = R/N$
$Case3:$	$(Y = T) \wedge (N_T = 1) \wedge (X(k) = T)$
	$R_1(k) = R, R_2(k) = R$
$Case4:$	$(Y = T) \wedge (N_T \neq 1) \wedge (R \geq 0) \wedge (X(k) = T)$
	$R_1(k) = -R \times (N_T - 2)/N, R_2(k) = 0$
$Case5:$	$(Y = T) \wedge (N_T \neq 1) \wedge (R \geq 0) \wedge (X(k) = T)$
	$R_1(k) = R \times N_T/N, R_2(k) = 0$

TABLE II
The generation rule for the reinforcement signal in a non-inverted link

$Case1:$	$(R > 0) \wedge (Y_T = Y)$
	$R_T = 0$
$Case2:$	$(R > 0) \wedge (Y_T \neq Y)$
	Reconstructing TL2
$Case3:$	$(R \leq 0) \wedge (Y_T = Y)$
	$R_T = R_1$
$Case4:$	$(R \leq 0) \wedge (Y_T \neq Y)$
	$Y_T = -R_1$

than a second threshold (TH2), the non-inverted link promotes the test node to the real structure and destroys itself. If the stored reinforcement signal in TL2 is less than 0, the test node generates a new TL2.

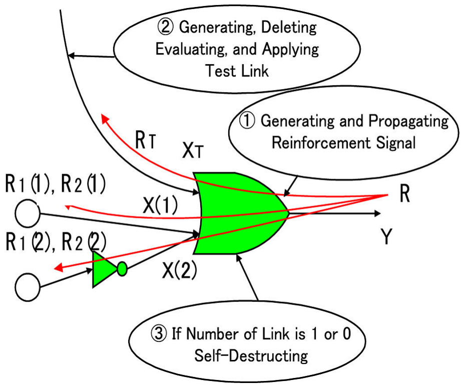

② Generating, Deleting Evaluating, and Applying Test Link
① Generating and Propagating Reinforcement Signal
RT
XT
R1(1), R2(1)
X(1)
R
Y
R1(2), R2(2)
X(2)
③ If Number of Link is 1 or 0 Self-Destructing

Fig. 1. Or-node

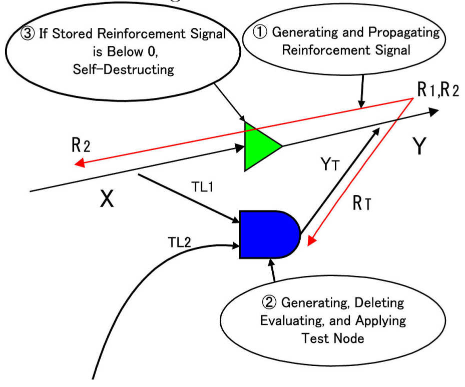

③ If Stored Reinforcement Signal is Below 0, Self-Destructing
① Generating and Propagating Reinforcement Signal
R1,R2
R2
YT
Y
X
TL1
RT
TL2
② Generating, Deleting Evaluating, and Applying Test Node

Fig. 2. Non-inverted link

C. Network

A network can be self-organized by the phases of its elements; the action phase, the propagation phase, and the restructuring phase. In the initial state of the network, input-nodes and output-nodes are prepared according to the number of the robot I/O elements. Input-nodes are dummy nodes and only receive as input from the robot its output Y. Output-nodes are or-nodes which do not have the ability to self-destruct. These elements are managed in a list, which registers input nodes and output nodes in this order. The list registers newly created nodes (middle nodes) before the node, which is the output side of this new node in the network.

When the robot calculates the output of the network, the action phase is activated for every node in the list. In this calculation, nodes are activated from the head to the tail of the list. When the robot learns, all the output nodes take the reinforcement signal to be the same amount as the signal calculated from the robot's states. Then the reinforcement signals are propagated during the propagation phase. In this calculation, nodes are activated from the tail to the head of the list. After this propagation, the network enters the restructuring phase for all of its elements. In this calculation, any activation order is applicable. These calculations are repeatedly done in this order by the robot so that it learns in real-time and online.

III. Proposed implementations

Traditional experiments involve 3-bit operation tasks and collision avoidance tasks in simulated robots using SONE. However, the value of reinforcement signals in such experiments is set equal among output nodes because it is very difficult to compare the value of each network's output in one step. In this system, complicated control of the outputs using supervised signals cannot be treated because the output nodes must learn against different targets to achieve supervised learning. Therefore, we introduced a supervised learning phase to the SONE.

A. Implementation of supervised learning method

In addition to the traditional learning phase (the reinforcement learning phase), we created a new learning phase which we call the supervised learning phase. This phase is where the network learns using a target vector given from outside of the network. It is reasonable to propagate reinforcement signals to the output nodes after converting target signals because each node is evaluated using reinforcement signals from the other nodes and the network changes its topology to get more reward signals (positive reinforcement signal). So in this phase, when the output of an output-node is the same as target value, a reward signal (1) is set to the output-node. On the other hand, when the output of the output-node is a counter, a penalty signal (-1) is set to the output-node. In this implementation, the reward system in the network can be shared by both learning phases. Therefore, both learning phases can coexist in a single network.

IV. Experiments

We introduced tracking experiments for a circular track (C-track) and an infinite shaped track (I-track) in a real numbered two-dimensional space. These experiments started from a specific point on a track. At first, the network received input from the x-y coordinate point on the track. The network had to output the next point to allow movement on the track. If the point to which the agent moved was not the same as teaching data, the network learnt the correct point from the teaching data. Then the correct point was inputted to the network. The I/O interface for the SONE was a 16-bit binary A/D and D/A transformer which was used to correspond with real numbers during tasks.

The experiment involving the C-track was a static task for the SONE because the network can only calculate the next point from the input. However, in the experiment involving the I-track, the network had to store the input in the memory and calculate the output from the memory because the network cannot calculate the output from only the input at the two points overlapping at the centre of the I-track. So, the I-track is generally classified as a temporal sequence problem.

Teaching data was created for each track by choosing eight points on the track (the two data points at the centre of the I-track were included). We determined "one step" to the calculation of an output vector in previous work[19], [20], since the SONE can learn by online. However in the present experiment, we use the term "one step" to describe one cycle of the dataset as occurs with a large number of traditional experiments in neural networks. Each experiment was stopped when the error calculated by a one-dimensional norm had not been renewed in 10,000 steps. We defined the point next to the last renewed point as the convergence point in this paper.

A. Noise-free environment

The results shown in TableIII indicate the average data for ten trials. In the experimental results listed in TableIII, the average error of bit (AEB) was within one bit. The outputs within one step was $8 \times 32 = 256$ bits (C-track or I-track) and $16 \times 32 = 512$ bits (C&I-track). These results suggest that learning was successful. The average error of real number (AER) was measured by using the standard radius of a circle. This error converged to zero. Figure 3 shows the convergence of the error in the experiment for the C&I-track. The results shown in this figure indicate that AER converges with AEB.

B. Noisy environment

In this experiment, the signals inputted to the network had 5% noise in real numbers. The effect of learning is

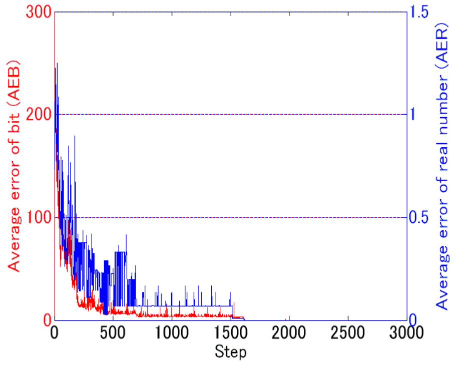


Fig. 3. Error history (noise-free)

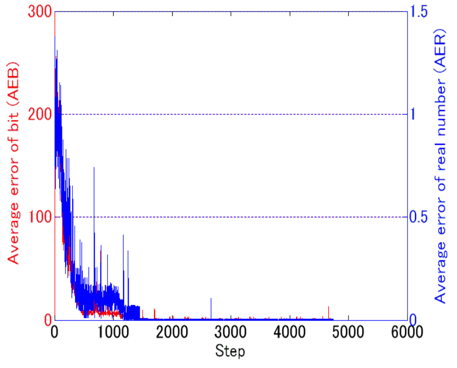


Fig. 4. Error history (noisy environment)

shown in TableIV. Compared to the noise-free experiments, both AEB and AER decreased, and the number of nodes and links at the CP increased.

The convergence of the error for the C&I-track is shown in Fig.4. The results shown in this figure indicate that the AER converges with the AEB, as it does in the noise-free case shown in Fig.3. Figs. 6 and 5 show the history of tracking and the structure obtained using the C&I-track. The nodes lying to the left in Fig.5 are the input-nodes, and the nodes lying to right are the output-nodes. The network formed in the middle is the result of the self-organization of network elements.

C. Incremental learning

To investigate the incremental learning feature of SONE, we performed the following experiment. In the previous experiment, C&I-tracks were switched at each step. Therefore, the C-track and I-track were learnt almost simultaneously. In this new experiment, we initially taught the C-track to completion and then taught the I-track. We changed the track back to the C-track when the I-track was completely learnt by the network. This experiment was then repeated.

The error history is indicated in Fig. 7. In this figure the step at which the error increases sharply is the step in which the tracks were switched. Fig.7 also shows that

TABLE III

Conclusion of learning (Noise-free)

Data	Average steps	Average error (bit)	Average error (real number)	Number of nodes at CP	Number of links at CP
C-track	88.3	0.3	6.25×10^{-7}	70.8	244.0
I-track	1064.9	0.3	4.93×10^{-5}	82.2	306.3
C&I-track	923.1	1.0	2.78×10^{-4}	149.6	536.7

TABLE IV

Conclusion of learning (noisy environment)

Data	Average steps	Average error (bit)	Average error (real number)	Number of nodes at CP	Number of links at CP
C-track	196.3	0	0	91.5	318.2
I-track	1068.7	0	0	153.9	537.5
C&I-track	2779.8	1	2.08×10^{-6}	448.1	1725.3

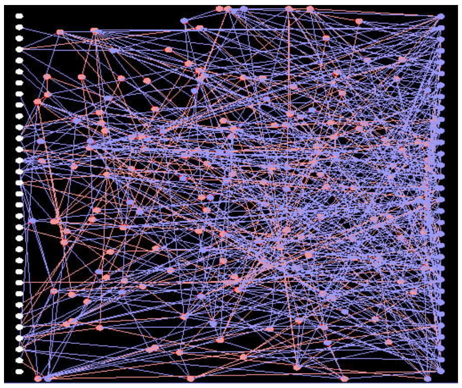


Fig. 5. Generated circuit

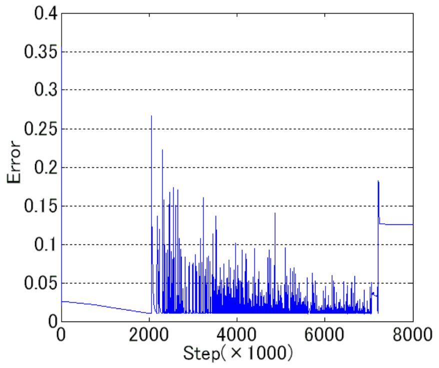


Fig. 8. Error history in incremental learning with RNN

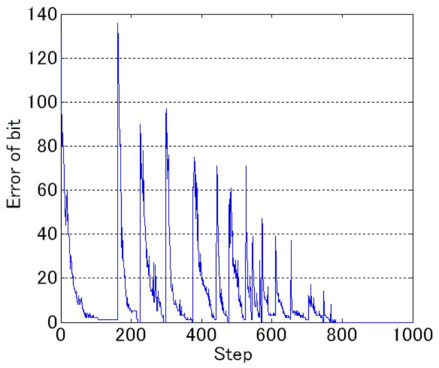


Fig. 7. Error history in incremental learning

overshoot peaks decreases.

This experiment was also performed using a recurrent neural network (RNN). The topology of the RNN was 2-20-2 layers with five nodes as the context layer. The learning ratio of the RNN was 0.01. The error history is indicated in Fig.8.

V. Discussions

A. Influence of binary on convergence

Before the experiment, we considered that convergence in the errors of the bits and the errors of the real numbers would be very different because the flags in the higher section of the output can change as a result of noise or for some other reason. In our experiments, however, both graphs converge in the same manner.

To explain this effect, we propose the following hypothesis. Learning higher sections of the outputs may be an easier task for SONE and convergence may occur fast in higher sections because the higher sections of teaching data do not generally change so often in the tracking task. If this hypothesis is true, the learning of SONE is very stable and reliable even in a system in which there is no guarantee of convergence, because, even when the lower section does not converge, the higher section can converge.

B. Generalization

Generalization is a key topic of research in the field of neural networks, and experiments with noise have confirmed that SONE have some type of generalization ability. In this experiment, SONE must generalize the dataset and respond to inputs that are always different because of noise. However, SONE could learn all tracks with high accuracy and the error converged to zero sometimes. Therefore, SONE also have this ability to generalize.

C. Incremental learning

The errors of the C-track and I-track both converged at the same time in the incremental learning experiments. So, new data is assimilated by the SONE without the complete removal of data previously learnt.

When compared to the RNN, learning speed was very fast (about 1/10,000 in terms of the number of steps) and the stability of learning was better. Unfortunately, RNN has the problem of excessive learning. So, the error overshot and did not come back, as shown in Fig.8. To minimize this effect, a low learning ratio is more effective. However, if we minimize the learning ratio, the number of steps required for learning increases. Therefore, it is difficult for a standard RNN to solve this type of problem as fast as SONE.

D. Temporal sequence tasks

The I-track experiment required a network to store the input in the memory and to calculate the output from the memory because the network could not calculate the output only from the input at the two points overlapping at the centre of the I-track. So in general, the I-track is classified as a temporal sequence problem. The SONE must create an accurate feedback loop to solve temporal sequence problems because the nodes do not store signals.

The accomplishment of the I-track experiment indicated that the SONE could obtain the structure of memory by using a feedback loop in a way similar to RNN.

VI. Conclusion and future work

We have proposed a supervised learning method using SONE. The system can create the network topology by itself. Also, the time cost for calculation was determined to be $O(V+E)$.

SONE can handle noisy data using the ability of generalization. Also, if the data are divided to parts, SONE can learn these parts at different times, and can combine these data through the ability of incremental learning. Moreover, when the data include hidden states, temporal sequence learning can be available. These results indicate that supervised learning with SONE is very robust and helpful for preparing a network before the activation of a robot.

In addition, the evaluation value R can be calculated for every element using this proposed supervised learning procedure. So, the use of the evaluation value for reinforcement learning is anticipated.

In the future, the comparison to other learning methods should be done. Also, tasks involving longer temporal sequences should be investigated. Moreover, creating SONE with real numbered elements is a great challenge.

References

[1] Jun Tani and Jun Yamamoto:"On the dynamics of robot exploration learning", Cognitive Systems Research, Vol.3, pp.459-470, 2002.

[2] A.Robins and S.McCallum:"The consolidation of learning during sleep: comparing the pseudorehearsal and unlearning accounts", Neural Networks, Vol.12, pp.1191-1206, 1999.

[3] Masumi Ishikawa:"Structural Learning with Forgetting" , Neural Networks, Vol.9, No.3, pp.509-521, 1996.

[4] Jie Ni and Qing Song:"Dynamic pruning algorithm for multilayer perception based neural control systems", Neurocomputing, 2005.

[5] Pengfei Xu and Chip-Hong Chang:"Self-Organizing Topological Tree", ISCAS V 732-735, 2004.

[6] Gang Leng, Girijesh Prasad, and Thomas Martin McGinnity: "An on-line algorithm for creating self-organizing fuzzy neural networks", Neural Networks 17 1477-1493, 2004.

[7] Yuichi Kobayashi, Hideo Yuasa, and Shigeyuki Hosoe: "Hyper-cubic Function Approximation for Reinforcement Learning Based on Autonomous-Decentralized Algorithm", SICE Vol.40 No.8 849-858,2004.

[8] Yasutake Takahashi and Minoru Asada: "Multi-Controller Fusion in Multi-Layered Reinforcement Learning", International Conference on Multi-sensor Fusion and Integration for Intelligent Systems ,2001.

[9] Katsunari Shibata, Yoichi Okabe, and Koji Ito:"Direct-Vision-Based Reinforcement Learning using Neural Network(in Japanese)", Transaction of the Society of Instrument and Control Engineering, Vol.37, No.2, pp.168-177, 2001.2 (in Japanese)", Proceeding of SANDI

[10] Kenneth O. Stanley and Ristio Miikkulainen: "Efficient Reinforcement Learning Through Evolving Neural Network Topologies", In Proceedings of the Genetic and Evolutionary Computation Conference, 2002.

[11] Kenneth O. Stanley, Bobby D. Bryant, and Ristio Miikkulainen:"Real-Time Neuroevolution in the NERO Video Game", IEEE Transactions on Evolutionary Computation, 2005.

[12] Shimon Whiteson and Peter Stone:"Evolutionary Function Approximation for Reinforcement Learning", Machine Learning, 2006.

[13] Akihide Utani, Gen Kobayashi, Yuji Yamazaki, and Nobuyoshi Tosaka: "Self-Designing Neural Network with Integrated Learning Algorithm for Structure and Weight Parameters", Transaction of JSCES Paper NO.20010043, 2001.

[14] Bertrand Mesot and Christof Terscher: "Deducing local rules for solving global tasks with random Boolean networks", Physica D 211 88-106, 2005.

[15] Naoko Shibata, Teruhiko Kitagawa, and Tetsuya Fukunaga: "Control of Autonomous Mobile Robot with Self-Organizational Neural Network used Reinforcement Learning", the Japan Society of Mechanical Engineers Robotics and Mechatronics Conference, 2003.

[16] Robert E. Schapire:"The strength of weak learnability", Machine Learning, 1990.

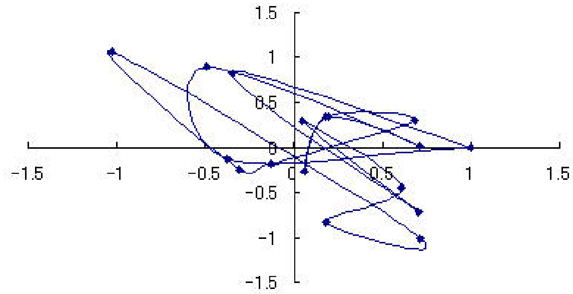
[17] Yoav Freund and Robert E. Schapire:"A Short Introduction to Boosting", Journal of Japanese Society for Artificial Intelligence, 1999.

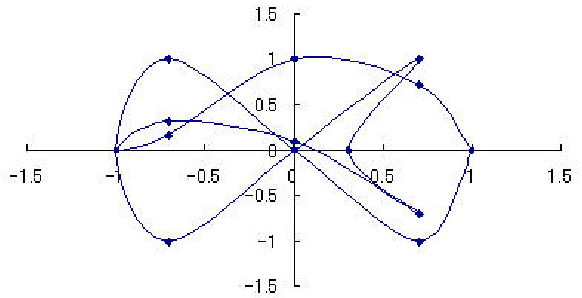
[18] Richard S.Sutton and Andrew G.Barto: "Reinforcement Learning An Introduction", A Bradford Book, The MIT Press, 2000.

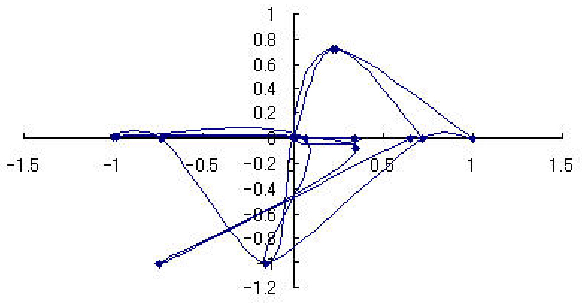
[19] Chyon Hae Kim, Tetsuya Ogata, and Shigeki Sugano:"Self-Organizing Algorithm for Logic Circuit based on Local Rules"(in Japanese), Transaction of the Society of Instrument and Control Engineering Vol. 42 No.4, 2006.

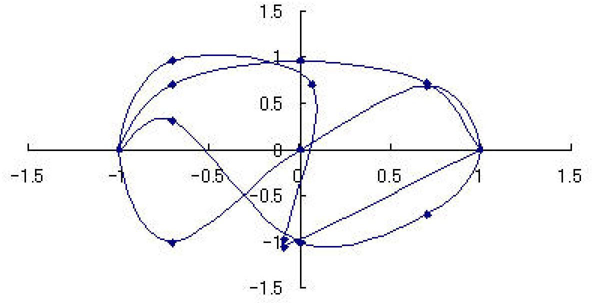
[20] Chyon Hae Kim, Tetsuya Ogata, and Shigeki Sugano:"Efficient Organization of Network Topology based on Reinforcement Signals", IEEE/RSJ Proceeding of the International Conference on Intelligent Robotics and Systems, 2006.

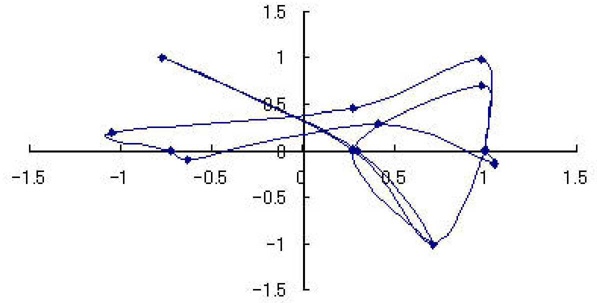
Fig. 6. Tracking history (C&I-track)

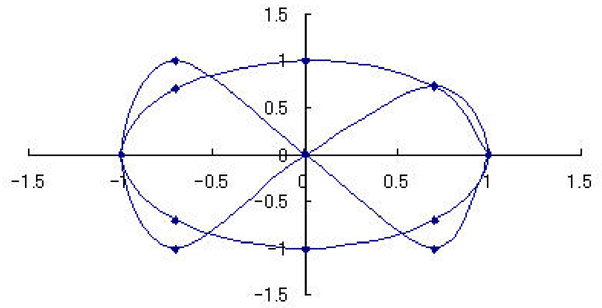

(1) 0 step

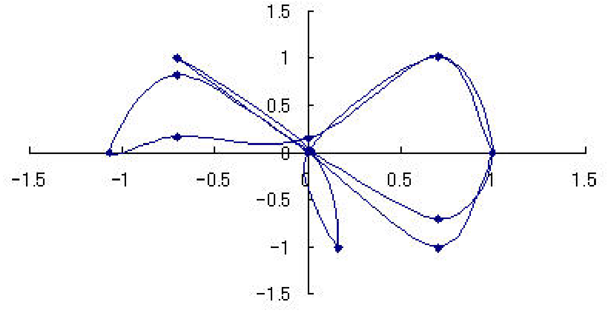

(5) 400 step

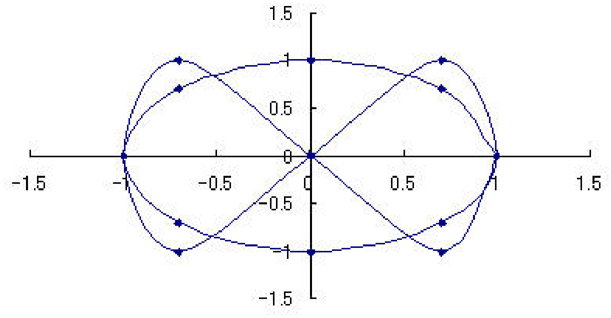

(2) 100 step


(6) 500 step


(3) 200 step


(7) 600 step


(4) 300 step


(8) 772 step