Learning of the Way of Abstraction in Real Robots

Atsushi Ueno*, Hideaki Takeda* & Toyoaki Nishida* **

*Graduate School of Information Science, Nara Institute of Science and Technology, Ikoma, Nara 630-0101, Japan
{ueno, takeda, nishida}@is.aist-nara.ac.jp
**School of Engineering, The University of Tokyo, Tokyo 113-8656, Japan
nishida@kc.t.u-tokyo.ac.jp

ABSTRACT

Real robots should be able to adapt flexibly to various environments. The main problem is how to abstract useful information from a huge amount of information in the environment. This is called the *frame problem.* This paper proposes a new architecture which can learn how to perform abstraction while executing the task. We call the architecture *Situation Transition Network System (STNS)*. By this architecture, a robot can acquire a necessary and sufficient symbol system for the current task and environment. Furthermore, this symbol system is flexible enough to adapt to changes of the environment. STNS performs cognitive learning and behavior learning parallelly while executing the task. In cognitive learning, it extracts *situations* and maintains them dynamically in the continuous state space on the basis of rewards from the environment. A situation can be regarded as an empirically obtained symbol. In behavior learning, it constructs an MDP (Markov Decision Problem) model of the environment on the abstracted situation representation. This model is used for planning of behavior. The validity of STNS is shown in computer simulations.

1 INTRODUCTION

Real robots should be able to adapt flexibly to various environments. However, most existing robots cannot adapt. The main reason is that they have fixed symbol systems. A human-made symbol system is not always suitable for the robot. Therefore, in the environments which the designers cannot expect, such a robot is apt to fail to manage a huge amount of information and get into the *frame problem.* In this paper, the frame problem is defined as "the problem of how to deal with partiality of information" [3].

The frame problem cannot be solved intrinsically because no agent can have all information in the world. However, animals seem not to be troubled with the frame problem in the daily life. It can be considered as a practical solution of the frame problem. We think that "obtaining appropriate information frames empirically" is a key function for this solution. This function can be regarded as cognitive learning or learning how to perform abstraction. It is expected that an agent with this function can obtain a necessary and sufficient symbol system for the current environment. By incorporating this with behavior learning on the symbolic system, an agent can have flexibility necessary for adapting to unfamiliar environments. Although this parallel learning has demerits in inefficiency and complexity of information processing, we think that it is essential to the practical solution of the frame problem.

Such cognitive agents are realized only in the domain of reinforcement learning. In the agent, reinforcement learning of behavior policy is executed in the symbol space, and learning of the state representation is executed in the continuous input space. The latter is a kind of cognitive learning. Since the task of the agent is expressed all in the rewards from the environment in the general reinforcement learning problem, rewards can be used reasonably as the basis of articulation. In this point of view, reinforcement learning is suitable for cooperating with cognitive learning.

The following two properties are important for the cognitive learning to solve the frame problem practically. First, articulation should be based on rewards. By this property, highly abstracted state representation can be expected because the task is represented in rewards. Second, cognitive learning and behavior learning should be executed parallelly while executing the task. In this case, good state recognition can make good behavior, and good behavior can maintain well-shaped states. This on-line learning contributes to the following three advantages:

1. The system can adapt to changes of the environment.

2. The system can make a state representation from a few data since it can be adjusted afterward.

3. The robot can learn the area frequently where it goes frequently in task execution.

Advantage 1 means the flexibility of the symbol system. By this advantage, the robot need not wonder when it should start cognitive learning or how long it must continue learning. By advantage 2 and 3, it can be expected that learning converges immediately.

2 STNS

We have developed a new architecture which has the above two properties. We call the architecture Situation Transition Network System (STNS). This is a solution to the problem of how to adapt to various environments. By this architecture, a robot can acquire a necessary and sufficient symbol system for the current task and environment.

As shown in Fig. 1, STNS consists of a situation classifier, a situation transition network (STN), and several behavior modules. In each behavior step, the system identifies the current "situation" where the current input is included, makes a plan on the STN, and activates a behavior module according to the plan. And a list of the input, the corresponding situation, the selected behavior, and the acquired reward are put into a history database. It keeps the data for a fixed period, and always has fixed numbers of data. Cognitive learning and behavior learning are performed on the data in it.

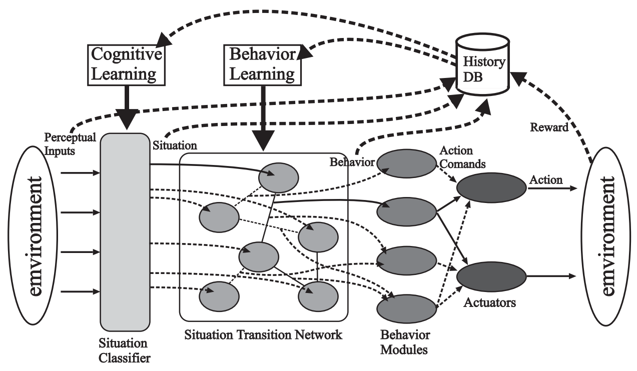


Figure 1: The structure of STNS

In cognitive learning, it extracts situations and maintains them dynamically in the continuous state space on the basis of rewards. A situation is defined as a set of input vectors from which the robot receives the same reward by the same behavior. It is a highly abstracted state and can be regarded as a empirically obtained symbol. In this way, the system can learn the appropriate way of abstraction from a huge amount of information of the real environment.

In behavior learning, it stores results of transitions between situations and constructs an MDP (Markov Decision Problem) model of the environment on the learned situation representation. This model is used for planning of behavior. The goal of the planning is to maximize discounted sum of rewards received over a period of time. Because the planning is performed on the abstracted situation representation, the robot with STNS does not have to deliberate details in planning. This process can be regarded as a kind of reinforcement learning of behavior policy.

The previous version of STNS [7] uses Interleave Planning-based Reinforcement Learning (IPRL) method for planning on the STN. Hereinafter, we call it STNS1. In this paper, we propose an improved version, STNS2, which uses the *policy iteration* method for planning on the STN. The policy iteration is one of generally used dynamic programming methods, and it is often the most efficient approach for small state spaces. The other modules in STNS2, that is a situation classifier, behavior modules, and a history database, are just the same as STNS1 because of the modularity of this architecture.

Cognitive learning and behavior learning are performed parallelly while executing the task. Therefore, the above-mentioned three advantages are realized.

In the next two sections, we explain the cognitive learning and the behavior learning in more detail.

3 COGNITIVE LEARNING IN STNS

In this section, we explain a new cognitive learning method which is used in STNS1 and STNS2 in common.

3.1 Segmentation Based on Similarity of Rewards

STNS segments the state space into some situations each of which has a specific meaning on the basis of similarity of rewards. The meaning of a situation is "the system can acquire the specific result by the specific behavior". The specific behavior is called the *condition behavior* of the situation. The specific results are divided into two types, i.e., *R-situation* and *T-situation*. In a situation based on immediate rewards called R-Situation, the result is to acquire a specific big reward. In a situation based on situation transitions called T-Situation, the result is to transit to a specific situation. If every chain of T-Situations is anchored to an R-Situation, every situation is guaranteed to lead to a specific big reward by the same sequence of behaviors.

3.2 Bitten Hyper-Ellipsoid Representation

In on-line learning, the system should be able to decide rough shapes of situations from a limited amount of data, and to decide finer shapes as data increase. For this purpose, we propose the *bitten hyper-ellipsoid* representation. In this representation, each situation is shaped by the positive instances and the negative instances that are decided based on the meaning of the situation.

As shown in Fig. 2, this representation is a mixture of macroscopic cognition and microscopic cognition. In macroscopic cognition, the boundary of a situation is a contour of Mahalanobis' distance from the population of the positive instances. This boundary forms a hyper-ellipsoid[1] . This cognition is quick and rough that can be formed even from very few data. Microscopic cognition is realized by the Nearest Neighbor methods[2] and grows finer as data increase. By mixing these two types of cognition, a fine and flexible cognition is realized.

3.3 Cognitive Learning in STNS

In STNS, the state space is divided into overlapped bitten hyper-ellipsoids (situation 1-7) and a margin space

[1] [1] uses the same type of representation.

[2] [6] uses the same type of representation.

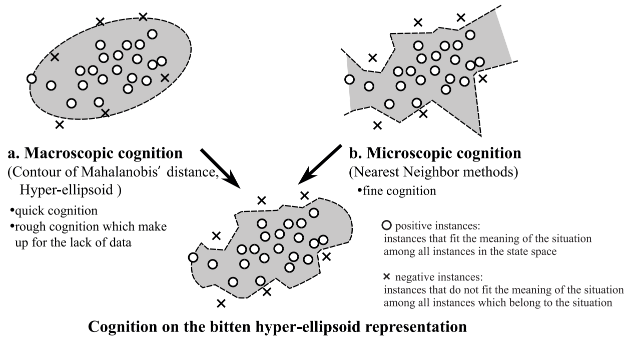

a. Macroscopic cognition
(Contour of Mahalanobis' distance, Hyper-ellipsoid)
•quick cognition
•rough cognition which make up for the lack of data
b. Microscopic cognition
(Nearest Neighbor methods)
•fine cognition
positive instances:
instances that fit the meaning of the situation among all instances in the state space
negative instances:
instances that do not fit the meaning of the situation among all instances which belong to the situation
Cognition on the bitten hyper-ellipsoid representation

Figure 2: The bitten hyper-ellipsoid representation

(situation 0) as shown in Fig. 3. A new perceptual input is ascertained whether it belongs to each situation from the top of the discrimination tree in Fig. 3.

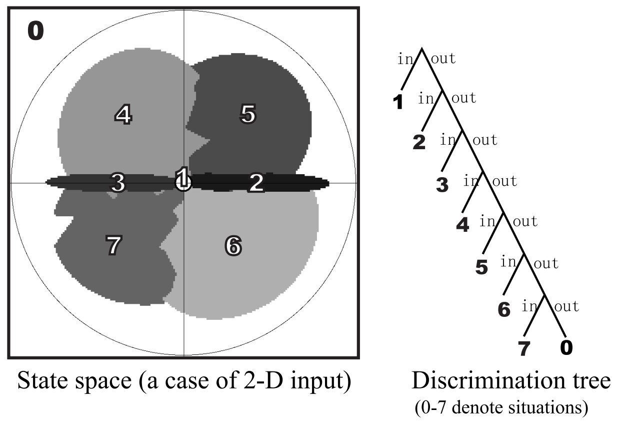


Figure 3: The state space and the discrimination tree

Cognitive learning is realized by extraction and maintenance of situations. There are two conditions for situation extraction.

1. **R-Situation** In the whole state space, there are enough (more than N^R_{init}) data in which the system acquired a specific reward larger than a threshold (r^R_{min}) by a specific behavior.

   T-Situation In situation 0, there are enough (more than N^T_{init}) data in which the system transited to a specific situation other than situation 0 by a specific behavior and did not acquire a large (more than r^R_{min}) reward.

2. There is no situation yet which has the specific behavior and the specific result in condition 1 as its meaning.

When both of these conditions are fulfilled, a new situation is extracted. In extraction, the meaning of the new situation is defined referring to the condition 1, and positive instances are collected from the history database.

The extracted situations are maintained by these five methods.

- Replacement of instances in the population of the positive instances and the population of the negative instances of each situation
- Adjustment of the boundary of each hyper-ellipsoid (The boundary is settled just on the farthest positive instance.)
- Adjustment of the order of situations in the discrimination tree (The nearer situation to rewards is put upper.)
- Changing the meaning of deformed situations
- Elimination of deformed situations

4 BEHAVIOR LEARNING IN STNS2

In STNS, each situation has a condition behavior. However, it decides a behavior policy by reinforcement learning. The main reason is that the cognitive learning in STNS is on-line learning: it extracts a situation from a few data and it is assumed that the environment can be changed, therefore the initial condition behavior is not always the optimal behavior.

Behavior learning is realized by combination of learning of the STN and behavior planning on it. There is no difference in components of STN between STNS1 and STNS2. The only difference is the planning method: STNS1 uses IPRL and STNS2 uses the policy iteration method. In this section, we explain the STN first, and then explain a planning method with policy iteration.

4.1 STN

An STN is an MDP model which consists of the transition probabilities between situations and the expectations of immediate rewards accompanying transitions. The transition probability from situation i to situation j by behavior b is called $p(i,j;b)$, and the expectation of the immediate reward which are acquired by the transition is called $r(i,j;b)$.

If Marcov property is assumed, the problem of finding the optimal policy on an STN is a Markov decision problem. Accordingly, the optimal policy is determined by solving the optimal equation of dynamic programming:

$$U(i) = \max_{b \in B} \sum_{j \in X} p(i,j;b)\{r(i,j;b) + \gamma U(j)\} \qquad (1)$$

where X is the state space, B is the behavior space, i ($i \in X$) is a situation, and γ ($0 \le \gamma < 1$) is a discounting factor. $U(i)$ is the utility of situation i, which is the expectation of the discounted sum of the rewards received over time by the optimal policy.

In STNS, the MDP model cannot be given in advance since the situation representation is changed dynamically. Therefore, the model are estimated by the maximum likelihood estimation from the data in the history database at the same time as the learning of a behavior policy. This problem is in the domain of reinforcement learning.

4.2 Planning with Policy Iteration

Many reinforcement learning methods keep the estimated utilities of situations, and improve them little by little after every behavior to convergent to the optimal values. However, STNS does not keep them but calculate them as the need arises in order to adapt to the changes of situation representation immediately.

STNS2 uses the policy iteration method for calculating the utilities of situations. This method works by picking a policy, then calculating the utility of each situation. Given a fixed policy, a set of optimal equations (Equation 1) are simultaneous linear equations and can be solved directly by linear algebra methods such as Gaussian elimination to find the utilities of situations. It then updates the policy at each situation to maximize its utility using the utilities of the successor situations, and repeats until the policy stabilizes.

By this method, optimal behaviors in all situations are determined. It is expected that a simple hill-climbing method which picks the optimal behavior reactively in each situation leads the robot to a big reward. However, STNS2 makes a plan before activate a behavior module. A plan P in STNS is a list of pairs of a behavior and a *target situation* of the behavior:

$$P = ((b_1, d_1), (b_2, d_2), ..., (b_n, d_n))$$

where n is the length of the plan, b_i is the ith behavior in the plan, and d_i is the target situation of the behavior. The target situation can be used as a stopping condition of a continuous behavior. As shown in Fig. 3, the situations made by STNS has rugged shapes. Therefore, this sort of stopping condition is useful in order not to be caught in a projection of an undesirable situation.

The procedures for planning are as follows:

1. Find the utilities of all situations by the policy iteration method.
2. Develop a plan forward from the current situation. Set the optimal behavior of the situation as the behavior b_i, and the most promising destination as the target situation d_i:
$$d_i = \arg\max_{j \in X} p(d_{i-1}, j; b_i)\{r(d_{i-1}, j; b_i) + \gamma U(j)\}$$
3. Continue plan developing until the peak of the utility or a terminal situation which has no successor situations.

Then, the system starts to execute the plan in the environment. Plan execution is stopped and planning is started again after the last behavior in the plan is executed, or another situation than the target situation is reached.

5 EXPERIMENTS

We conducted three types of experiments on computer simulation in order to examine the validity of STNS2. The results of the same sorts of experiments of STNS1 are shown in [7].

5.1 Experiment 1: Navigation on 2-D Input

To show the validity of STNS2, we show a simple experiment on computer simulation. Fig. 4 shows the navigation problem on 2-dimensional input. Every trial starts after the goal is settled at an arbitrary position and the rover is set at an arbitrary position outside of the goal in an arbitrary direction. The trial ends when the rover arrives at the goal, and the next trial starts immediately. At the beginning of learning, there are only the goal situation and situation 0 (the margin space) in the state space. We set parameters as follows: $N^R_{init} = 3, N^T_{init} = 15, r^R_{min} = 5.0, \gamma = 0.7$. Note that the number of positive instances for R-Situation extraction, $N^R_{init} = 3$, is the minimum in order to construct an ellipsoid in the 2-dimensional state space. We repeated this experiment 20 times.

Task:
To navigate a 16x12 rectangle rover from an arbitrary position to the only goal (a small circle whose radius is 5) in a 100x100 square continuous plane.
The goal is settled at an arbitrary position in the 72x72 square area at the center of the room.
There is no obstacle but the wall around the room.

Perceptual Input (2-dimensinal):
The real-valued (x, y) coordinates of the goal on the coordinate system fixed to the rover (the goal sensor).

Rewards:
1. Arrival at the goal: +10
2. Collision with the wall: -1
3. Trying to rotate over 90 degrees: -1
(The rover can look any direction by at most 90 degrees rotation because of the symmetry of the behaviors.)

Behaviors:
forward movement, backward movement, clockwise rotation, counterclockwise rotation.
(The rover is assumed to be able to make collision-free rotation.)

Stopping Conditions of Behaviors:
1. Getting some reward.
2. Arrival at the target situation in the master plan.

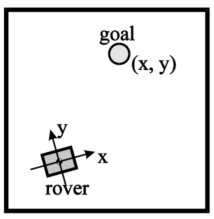


Figure 4: The navigation problem on 2-D input

Learning has converged after about 1300 behaviors are executed on average. Fig. 5 shows a typical state space after leaning has converged and the optimum state space. The small circle at the center of each space denotes states in which the rover arrives at the goal. As shown in this figure, good situation representation and good behavior policy were acquired.

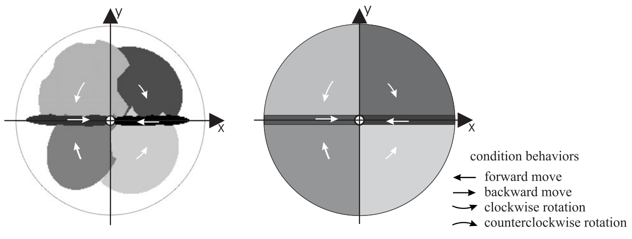


Figure 5: A state space after convergence and the optimum state space

5.2 Experiment 2: Flexibility Test

To test the flexibility of STNS2, we conducted two experiments on computer simulation. One is a flexibility test to sensor trouble and the other is a flexibility test to actuator trouble. In both experiments, we picked 10 systems after 5000 behavior steps in Experiment 1, changed the environment, and then let them continue learning in the new environment. As changes of environment, we rotated the direction of the goal sensor (10, 15, 20, 25, 90, and 180 degrees rotation) in the former test, and made the revolution rate of the left wheel lower (5%, 10%, and 15% lower speed) in the latter test. Fig. 6 shows a typical result in the case of 15 degrees rotation of the goal sensor and a typical result in the case of 15% lower speed of the left wheel. As shown in this figure, passably good situation representation and good behavior policy were acquired.

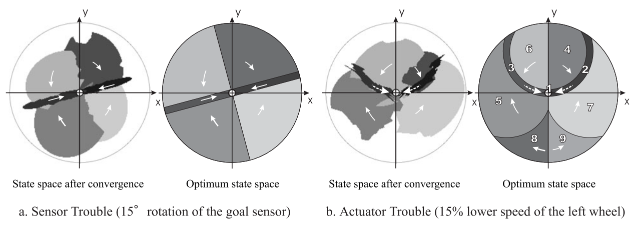
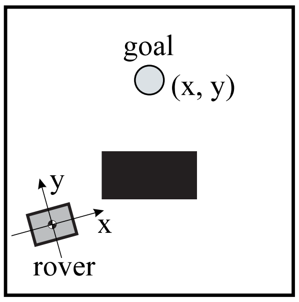

State space after convergence
Optimum state space
State space after convergence
Optimum state space
a. Sensor Trouble (15° rotation of the goal sensor)
b. Actuator Trouble (15% lower speed of the left wheel)

Figure 6: Results of the flexibility test

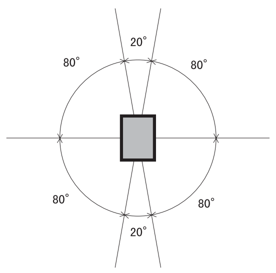
In both experiments, STNS2 can flexibly adapt to small changes while keeping the high performance by transforming the shapes of situations, and it can stably adapt to big changes by eliminating deformed and obstructive situations and starting again from the blank state space. These two types of adaptation are switched to each other autonomously. Even in the case of big changes, learning converged to the same performance after a delay of at most 1000 behavior steps compared with learning from the blank state space. So it can be said that learning of STNS2 is stable.

5.3 Experiment 3: Navigation on 8-D Input

The last experiment is a navigation problem on 8-dimensional input. The setting is almost the same with Experiment 1. But one obstacle was settled in the room and the goal was fixed to the position shown in Fig. 7a. And the rover got a 6-dimensional obstacle sensor shown in Fig. 7b in addition to the 2-dimensional goal sensor. We set parameters as follows: $N^R_{init} = 9, N^T_{init} = 20, r^R_{min} = 5.0, \gamma = 0.7$. Note that the number of positive instances for R-Situation extraction, $N^R_{init} = 9$, is the minimum in order to construct a hyper-ellipsoid in the 8-dimensional state space. This problem is difficult because the dimension of the state space is high and the paths through the state space is not continuous. We repeated this experiment 10 times.



a. Work space



b. 6-dimensional obstacle sensors

Figure 7: The navigation problem on 8-D input

Fig. 8 shows a typical segmentation after enough leaning (29000 behavior steps) and the optimum segmentation. The 8-D state space was mapped onto the 2-D work space by fixing the attitude of the rover. As shown in this figure, not bad situation classification and not bad behavior policy was obtained in the areas near to big rewards. However, in the areas far from big rewards, no useful situation was extracted. The main reason is that the ability for cognitive learning is not sufficient. For compensating for this weak point, I think hierarchical learning or empirical creation of new axes of the state space is effective.

5.4 Comparison between STNS1 and STNS2

When comparing STNS2 with STNS1, they have similar performance on learning: similar convergence rates, similar performance after convergence, similar flexibility to changes of environment, and so on. As an example, Fig. 9 shows the changes of the average numbers of behaviors to the goal in Experiment 1 comparing STNS1 and STNS2.

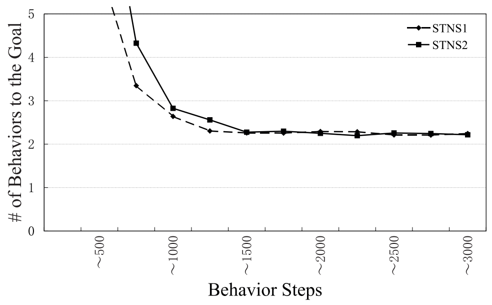


Figure 9: Learning Curves of STNS1 and STNS2

On the other hand, the elapsed time for the simulation of STNS2 was much shorter than STNS1. Fig.10 shows the changes of the average elapsed times of 1000 behaviors in Experiment 1 on an SGI Indy R5000SC workstation comparing STNS1 and STNS2. The elapsed time of STNS2 was about from 2.5 times to 5 times as short as STNS1, and furthermore, converged much earlier than STNS1. The reason for this is that the planning time of STNS2 is much shorter than STNS1. Although it is necessary to conduct still more experiments for general comparative conclusion, STNS2 seems to be more efficient than STNS1.

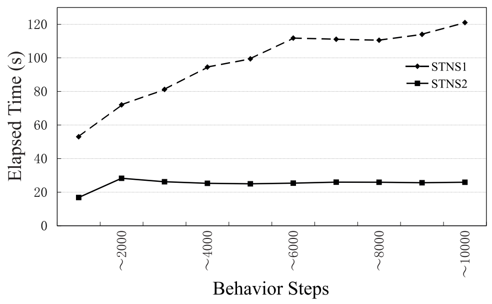
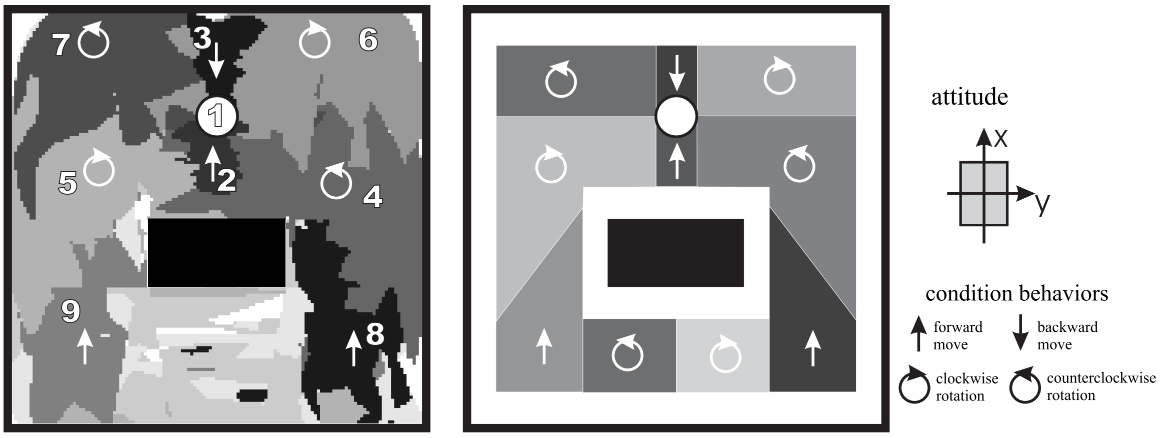


Figure 10: Elapsed Time of STNS1 and STNS2



a. Segmentation after enough learning b. Optimum segmentation

Figure 8: A segmentation after enough learning and the optimum segmentation

6 RELATED WORKS

There are many systems that can learn the state representation for reinforcement learning. Some of them are based on the similarly of rewards. Chapman and Kaelbling [2] proposed the G algorithm that splits the state space statistically on the basis of rewards. Mahadevan and Connell [5] proposed an algorithm that extracts clusters of states in the state space statistically on the basis of rewards. Both methods deal with states each of which consists of a bit sequence. Therefore, their systems are different from STNS that deals with a problem with a continuous state space. Asada et al. [1] proposed a method that divides a continuous state space by hyper-ellipsoids. The method executes cognitive learning in a off-line way: the system first learns the whole state representation on the basis of experiences of random behaviors and then shifts to task execution. Ishiguro et al. [4] proposed a method that first divides a continuous state space by hyper-planes on the basis of experiences of random (or given) behaviors, and then learns a behavior policy by Q-learning. It can execute these two types of learning alternately. But it constructs fixed boundaries between states. It is different from STNS at this point because STNS constructs a flexible situation representation. Takahashi et al. [6] proposed a method that segments a continuous state space by the Nearest Neighbor methods while executing the task. The segmentation of the method is based on rewards in the areas near to rewards and based on the relationship between the sensory input and its gradient in the areas far from rewards. By the latter policy, the method can segment the state space stably even in the areas far from rewards. However, the validity of this policy needs to be confirmed.

7 CONCLUSION

We have proposed STNS2, a method for performing both cognitive learning and behavior learning simultaneously with task execution, and shown that the method is effective to acquire good state representation and good behavior policy in continuous state space. The representation also have flexibility to changes of the environment. Furthermore, we have shown STNS2 is superior to the previous version, STNS1, in efficiency of planning.

There needs more work for the symbol processing system grounded on real environments. Reinforcement learning system can deal with symbols only for representing states or behaviors. Therefore, the dual learning system like STNS can execute only very simple symbol processing. In order to expand this sort of symbol system, the following four functions are worth considering: structuring symbols, symbolizing objects, reusing symbols, and sharing symbols.

In order to confirm the validity of STNS2, it is necessary to conduct experiments on real robots. We are now preparing to put it on a real robot.

REFERENCES

[1] M. Asada, S. Noda, and K. Hosoda. Action-based sensor space categorization for robot learning. In *Proc. of IROS 96*, Vol. 3, pp. 1502–1509, 1996.

[2] D. Chapman and L. P. Kaelbling. Input generalization in delayed reinforcement learning: An algorithm and performance comparisons. In *Proc. of IJCAI-91*, pp. 726–731, 1991.

[3] K. Hasida and H. Matsubara. Partiality of information and the structure of the frame problem. In *Proc. of PRICAI '90*, pp. 711–716, 1990.

[4] H. Ishiguro, R. Sato, and T. Ishida. Robot oriented state space construction. In *Proc. of IROS 96*, Vol. 3, pp. 1496–1501, 1996.

[5] S. Mahadevan and J. Connell. Automatic programming of behavior-based robots using reinforcement learning. In *Proc. of AAAI-'91*, pp. 768–773, 1991.

[6] Y. Takahashi, M. Asada, and K. Hosoda. Reasonable performance in less learning time by real robot based on incremental state space segmentation. In *Proc. of IROS 96*, Vol. 3, pp. 1518–1524, 1996.

[7] A. Ueno, H. Takeda, and T. Nishida. Cooperation of Cognitive Learning and Behavior Learning. To Appear in *Proc. of IROS 99*, Oct. 1999