Vision-Based Robot Learning Towards RoboCup: Osaka University "Trackies"

S. Suzuki[1], Y. Takahashi[2], E. Uchibe[2], M. Nakamura[2], C. Mishima[1], H. Ishizuka[2], T. Kato[2], and M. Asada[1]

[1] Dept. of Adaptive Machine Systems, Osaka University, Yamadaoka 2-1, Suita, Osaka 5650871, Japan.
[2] Dept. of Computer-Controlled Machinery, Osaka University, Yamadaoka 2-1, Suita, Osaka 5650871, Japan.

Abstract. The authors have applied reinforcement learning methods to real robot tasks in several aspects. We selected a skill of soccer as a task for a vision-based mobile robot. In this paper, we explain two of our method; (1)learning a shooting behavior, and (2)learning a shooting with avoiding an opponent. These behaviors were obtained by a robot in simulation and tested in a real environment in RoboCup-97. We discuss current limitations and future work along with the results of RoboCup-97.

1 Introduction

Building robots that learn to perform a task in a real world has been acknowledged as one of the major challenges facing AI and Robotics. Reinforcement learning has recently been receiving increased attention as a method for robot learning with little or no *a priori* knowledge and higher capability of reactive and adaptive behaviors [3]. In the reinforcement learning scheme, a robot and an environment are modeled by two synchronized finite state automatons interacting in discrete time cyclical processes. The robot senses the current state of the environment and selects an action. Based on the state and the action, the environment makes a transition to a new state and generates a reward that is passed back to the robot. Through these interactions, the robot learns a purposive behavior to achieve a given goal.

As a testbed to apply the reinforcement learning method for real robot tasks, we have selected soccer playing robots [1]. We have been doing various kinds of research topics as follows;

1. learning a shooting behavior in a simple environment [11]
2. learning a coordinated behavior of shooting and avoiding an opponent [12][15]
3. self construction of a state space [13]
4. learning of a real robot in a real environment [14]
5. modeling other agents [16]

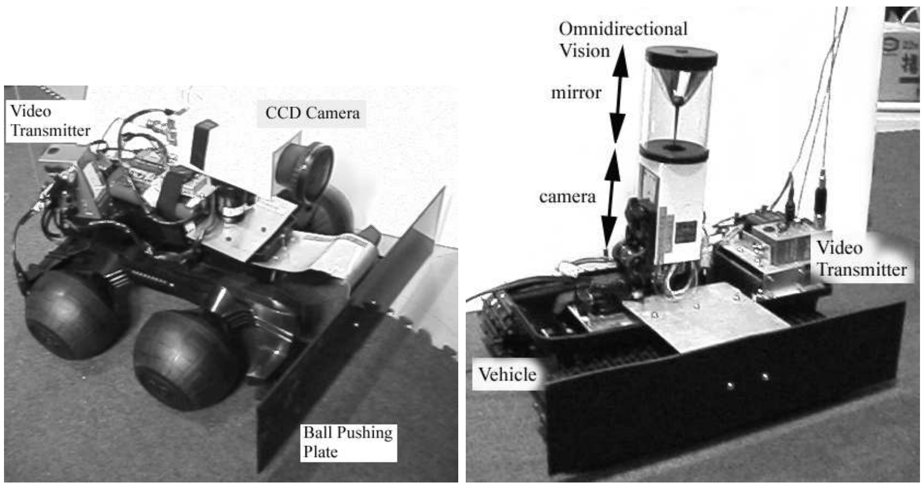
Two methods ([11] and [15]) are tested in RoboCup-97 in which robots take actions based on the learned policy that has not include cooperation between teammate robots yet this year.

In this paper, we summarize our research issues involved in realizing a robot team for RoboCup-97. This article is structured as follows: In section 2, we explain the configuration of our robot system. In section 3, we give a brief overview of Q-learning. In section 4, we explain acquisition of shooting behavior. In section 5, we explain acquisition of a coordinated behavior combined shooting and avoiding an opponent. In section 6, we describe the result. Finally, we give a conclusion.

2 The Configuration of the Robot System

We have decided to use a radio-controlled model car as a robot body and to control it based on the remote brain approach [9]. This makes us implement and monitor the system activities easy.

In RoboCup-97, we participated with five robots consisting four attackers and one goalie (see Figure 1). In this section, we explain the hardware and the control architecture for our robots.



(a) The attacker robot (b) The goalie robot

Fig. 1. Our Robots

2.1 Hardware of the Robots

We use radio-controlled model cars with a PWS (Power Wheeled Steering) locomotion system. Four of them are called “Black Beast” produced by Nikko as an attacker robot (see Figure 1(a)), and one called “Blizzard” produced by Kyosho as a goalie (see Figure 1(b)). A plate is attached to push the ball on the field. The attacker has the plate in front of the robot and the goalie has on its side. The robots are controlled by signal generated on the remote computer through the radio link.

Each robot has a single color CCD camera for sensing the environment and a video transmitter. The attacker robot has a SONY CCD camera with a wide lens while the goalie has an omnidirectional vision system [10] so that it can see the goal and the ball coming in any direction at the same time. The image taken by the camera is transmitted to the remote computer and processed on it.

For power supply, three Tamiya 1400NP batteries are mounted on the robot. Two drive two motors for locomotion, and the remaining one supplies 12V through a DC-DC converter to drive the camera and the transmitter. The life of the battery is about 20 minutes for locomotion and 60 minutes for the camera and the transmitter.

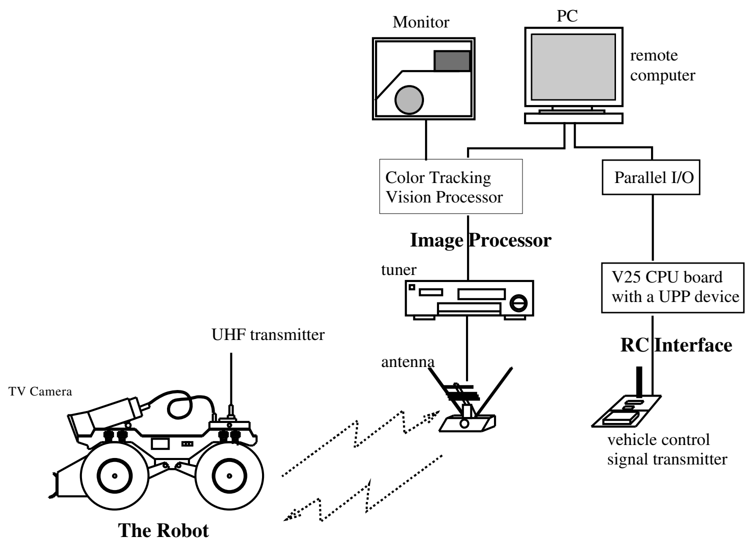


Fig. 2. Configuration of robot controller

2.2 The Control Architecture

The controller of each robot consists of three parts; a remote computer, an image processor, and a radio-control interface (RC interface). Figure 2 shows a configuration of the controller in which PC is used as the remote computer.

The action of the robot is controlled by the following steps:

1. the robot transmits the image from its camera,
2. the image processor receives the image through UHF and processes it,
3. the remote computer decides the robot's action based on the result of image processing,
4. the RC interface generates a signal corresponding to the decided action, and
5. the robot receives the signal and drives its motors.

We use a color tracking vision board produced by Fujitsu for the image processing, and a UPP device to generate the control signal. Objects in the environment (a ball, a goal, and an opponent) are detected as colored regions in the image according to RoboCup regulations.

3 Q-learning for Robot Learning

In the reinforcement learning scheme, the robot senses the current state of the environment and selects an action. Based on the state and the action, the environment makes a transition to a new state and generates a reward that is passed back to the robot. Through these interactions, the robot learns a purposive behavior to perform a given task (see Figure 3). As a method for reinforcement learning, we adapted Q-learning that is one of most widely used reinforcement learning method. In this section, we give a brief overview of Q-learning and problems when we apply it to real robot tasks.

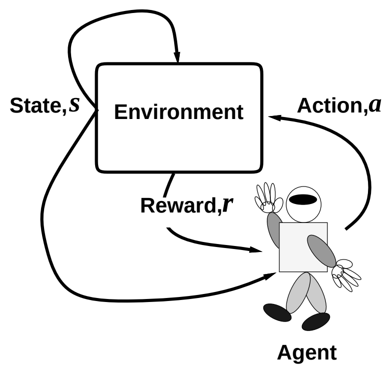


Fig. 3. Interaction between the robot and the environment

3.1 Basics of Q-learning

We assume that the robot can discriminate the set $\boldsymbol{S}$ of distinct environment states, and can take the set $\boldsymbol{A}$ of actions on the environment. The environment is modeled as a Markov process, making stochastic transitions based on its current state and the action taken by the robot. Let $T(s, a, s')$ be the probability of transition to the state s' from the current state-action pair (s, a). For each state-action pair (s, a), the *reward* $r(s, a)$ is defined.

Given the definitions of the transition probabilities and the reward distribution, we can solve for the optimal policy(a policy f is a mapping from $\boldsymbol{S}$ to $\boldsymbol{A}$), using methods from dynamic programming [2]. A more interesting case occurs when we wish to simultaneously learn the dynamics of the environment and construct the policy. Watkin's Q-learning algorithm gives us an elegant method for doing this [6].

Let $Q^*(s, a)$ be the expected *action-value function* for taking action a in a situation s and continuing thereafter with the optimal policy. It can be recursively defined as:

$$Q^*(s, a) = r(s, a) + \gamma \sum_{s' \in S} T(s, a, s') \max_{a' \in A} Q^*(s', a'). \tag{1}$$

Because we do not know T and r initially, we construct incremental estimates of the Q-values on-line. Starting with $Q(s, a)$ equal to an arbitrary value (usually 0), every time an action is taken, the Q-value is updated as follows:

$$Q(s, a) \Leftarrow (1 - \alpha) Q(s, a) + \alpha(r(s, a) + \gamma \max_{a' \in A} Q(s', a')). \tag{2}$$

where r is the actual reward value received for taking action a in a situation s, s' is the next state, and α is a learning rate (between 0 and 1).

3.2 Problems in Applying Q-learning to Real Robot Tasks

To apply Q-learning, we must cope with several problems which occur in real environments. Two major problems are construction of state and action sets, and reduction of learning time [11].

Construction of State and Action Sets In the environment where the robot exist, everything changes asynchronously. Thus traditional notions of state in the existing applications of the reinforcement learning algorithms dose not fit nicely [5]. The following principles should be considered for the construction of state and action spaces.

– Natural segmentation of the state and action spaces: The state (action) space should reflect the corresponding physical space in which a state (an action) can be perceived (taken).

– Real-time vision system: Physical phenomena happen continuously in the real environment. Therefore, the sensor system should monitor the changes of the environment in real time. This means that the visual information should be processed in video frame rate (33ms).

The state and action spaces are not discrete but continuous in the real environment, therefore it is difficult to construct the state and action spaces in which one action always corresponds to one state transition. We call this **"state-action deviation problem"** as a kind of the so-called "perceptual aliasing problems" [7] (i.e., a problem caused by multiple projections of different actual situations into one observed state). The perceptual aliasing problem makes it very difficult for a robot to take an optimal action. The state and action spaces should be defined considering this state-action deviation problem.

Reduction of Learning Time This is the famous *delayed reinforcement* problem due to no explicit teacher signal that indicates the correct output at each time step. To avoid this difficulty, we construct the learning schedule such that the robot can learn in easy situations at the early stages and later on learn in more difficult situations. We call this *Learning from Easy Missions* (or LEM).

4 Learning a Shooting Behavior

For the first stage, we set up a simple task for a robot [11], to shoot a ball into a goal as shown in Figure 4. We assume that the environment consists of a ball and a goal. The ball is painted in red and the goal in blue so that the robot can detect them easily. In this section, we describe a method for learning the shooting behavior with consideration of the problem mentioned in section 3. Here we focus on the method implemented on the attacker robot in RoboCup-97 (see [11] for more detail).

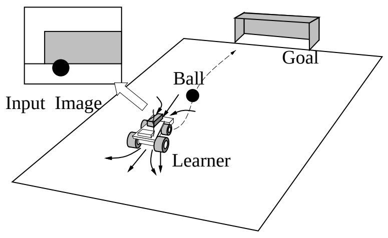


Fig. 4. The task is to shoot a ball into a goal

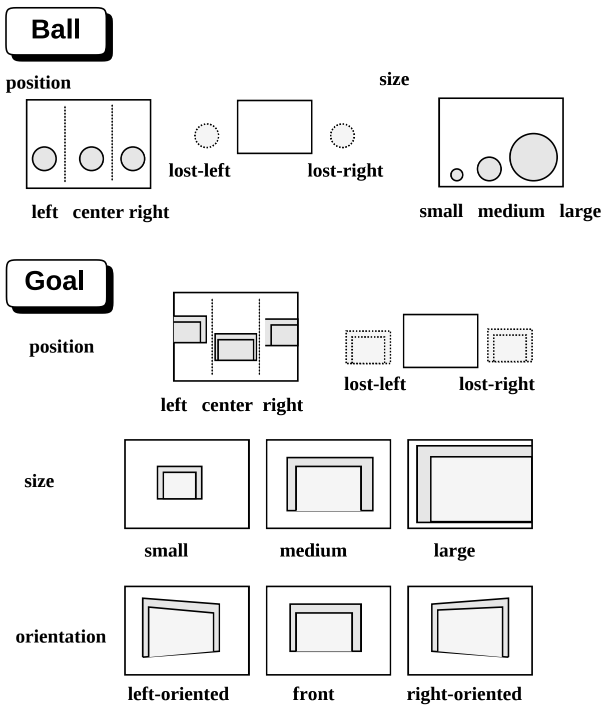


Fig. 5. The ball substates and the goal substates

4.1 Construction of Each Space

(a) a state set *S*: The ball image is classified into 9 substates, combinations of three classifications of positions (left, center, or right) and three types of sizes (large (near), middle, or small (far)). In addition to the size and the positions, the goal image has 27 substates considering the orientation which is also classified into three categories (see Figure 5). Each substate corresponds to one posture of the robot towards the goal, that is, the position and the orientation of the robot in the field.

In addition, we define states for the cases in which the ball or the goal is not captured in the image: three states (ball-unseen, ball-lost-into-right, and ball-lost-into-left) for the ball, and three more states (goal-unseen, goal-lost-into-right and goal-lost-into-left) for the goal. In all, we define 12 (9 + 3) states for the ball and 30 (27 + 3) states for the goal, and therefore the set of states ***S*** is defined with 360 (12 $\times$ 30) states.

(b) an action set *A*: The robot can select an action to be taken in the current state of the environment. The robot moves around using a PWS (Power Wheeled

Steering) system with two independent motors. Since we can send the motor control command ω_l and ω_r to each of the two motors separately, each of which has forward, stop, and back, we have nine action primitives all together.

We define the action set $\boldsymbol{A}$ as follows to avoid the state-action deviation problem. The robot continues to take one action primitive at a time until the current state changes. This sequence of the action primitives is called an action.

(c) a reward and a discounting factor γ: We assign the reward value to be 1 when the ball is kicked into the goal and 0 otherwise. This makes the learning very time-consuming. Although adopting a reward function in terms of distance to the goal state makes the learning time much shorter in this case, it seems difficult to avoid the local maxima of the action-value function Q.

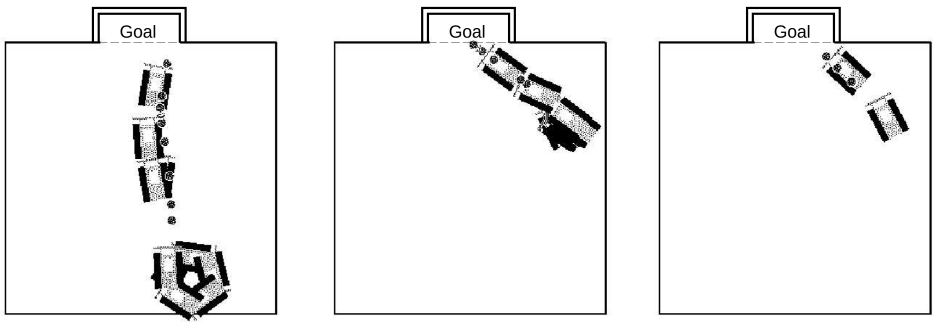
A discounting factor γ is used to control to what degree rewards in the distant future affect the total value of a policy. In our case, we set the value at slightly less than 1 ($\gamma = 0.8$).

4.2 Simulation

We performed the computer simulation. Figure 6 shows some kinds of behaviors obtained by our method. In (a), the robot started at a position from where it could not view a ball and a goal, then found the ball by turning, dribbled it towards the goal, and finally shot the ball into the goal. This is just a result of learning. We did not decompose the whole task into these three tasks. The difference in the character of robot player due to the discounting factor γ is shown in (b) and (c) in which the robot started from the same position. In the former, the robot takes many steps in order to ensure the success of shooting because of a small discount, while in the latter the robot tries to shoot a ball immediately because of a large discount. In the following experiments, we used the average value of γ 0.8 as an appropriate discount.

We applied the LEM algorithm to the task in which $\mathbf{S_i}$ ($\mathbf{i}$=1,2, and 3) correspond to the state sets of "the goal is large", "medium", and "small", respectively, regardless of the orientation and the position of the goal, and the size and position of the ball. Figure 7 shows the changes of the summations of Q-values with and without LEM, and ΔQ. The axis of time step is scaled by M (10^6), which corresponds to about 9 hours in the real environment since one time step is 33ms. The solid and broken lines indicate the summations of the maximum value of Q in terms of action in states $\in \mathbf{S_1} + \mathbf{S_2} + \mathbf{S_3}$ with and without LEM, respectively. The Q-learning without LEM was implemented by setting initial positions of the robot at completely arbitrary ones. Evidently, the Q-learning with LEM is much better than that without LEM.

The broken line with "$\times$" indicates the change of $\Delta Q(\mathbf{S_1} + \mathbf{S_2} + \mathbf{S_3}, a)$. Two arrows indicate the time steps (around 1.5M and 4.7M) when a set of the initial states changed from $\mathbf{S_1}$ to $\mathbf{S_2}$ and from $\mathbf{S_2}$ to $\mathbf{S_3}$, respectively. Just after these steps, ΔQ drastically increased, which means the Q-values in the inexperienced states are updated. The coarsely and finely dotted lines expanding from the time



(a) finding, dribbling, and shooting (b) shooting ($\gamma = 0.999$) (c) shooting ($\gamma = 0.6$)

Fig. 6. Some kinds of behaviors obtained by the method

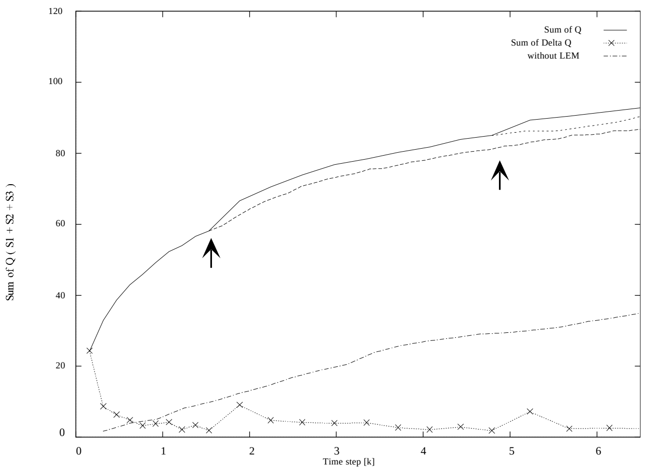


Fig. 7. Change of the sum of Q-values with LEM in terms of goal size

steps indicated by the two arrows show the curves when the initial positions were not changed from $\mathbf{S_1}$ to $\mathbf{S_2}$, nor from $\mathbf{S_2}$ to $\mathbf{S_3}$, respectively. This simulates the LEM with partial knowledge. If we know only the easy situations ($\mathbf{S_1}$), and nothing more, the learning curve follows the finely dotted line in Figure 7. The summation of Q-values is slightly less than that of the LEM with more knowledge, but much better than that without LEM.

5 Shooting a Ball with Avoiding an Opponent

In the second stage, we set up an opponent just before the goal and make the robot learn to shoot a ball into a goal avoiding the opponent (see Figure8). This task can be considered as a combination of two subtasks; a shooting behavior and an avoiding behavior of an opponent. The basic idea is first to obtain the desired behavior for each subtask, and then to coordinate two learned behaviors. In this section we focus on the coordination method implemented on the attacker robot in RoboCup-97, see [12] and [15] for more detail.

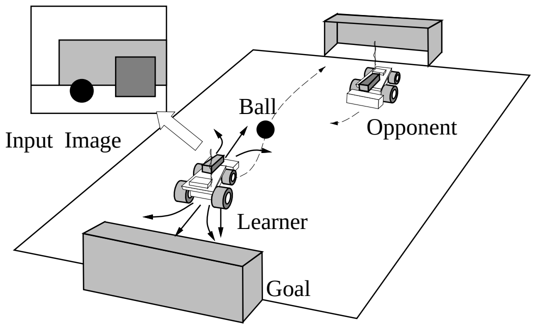


Fig. 8. The task is to shoot a ball into the goal avoiding an opponent.

5.1 Learning a Task from Previously Learned Subtasks

The time needed to acquire an optimal policy mainly depends on the size of state space. If we apply the monolithic Q learning into a complex task, the expected learning time is exponential in the size of state space [8]. One technique to reduce learning time is to divide the task into some subtasks and to coordinate behaviors which is independently acquired. The simple coordination method is summation or switching of the previously learned action value functions.

However, these method cannot cope with local maxima and/or hidden states caused by direct product of individual state spaces corresponding to the subtasks. Consequently, an action suitable for these situations has never been learned. To cope with these new situations, the robot needs to learn a new behavior by using the previously learned behaviors [12]. The method is as follows:

1. Construct a new state space $\boldsymbol{S}$:
   (a) construct the directly combined state space from subtasks' state $\boldsymbol{s}_1$ and $\boldsymbol{s}_2$
   (b) find such states that are inconsistent with $\boldsymbol{s}_1$ or $\boldsymbol{s}_2$
   (c) resolve the inconsistent states by adding new substates $s_{sub} \in \boldsymbol{S}$.
2. Learn a new behavior in the new state space $\boldsymbol{S}$:

(a) calculate the value of the action value function Q_{ss} by simple summation of the action value functions of each subtasks.

$$Q_{ss} = \max_{a \in \boldsymbol{A}}(Q_1((s_1, *), a) + Q_2((*, s_2), a)) \tag{3}$$

where $Q_1((s_1, *), a)$ and $Q_2((*, s_2), a)$ donate the extended action value functions. $*$ means any states, therefore each of these functions considers only the original states and ignores the states of other behaviors.

(b) initialize the value of the action value function Q for the normal states s and the new substates s_{sub} with Q_{ss}. That is,

$$\begin{aligned} Q(s, a) &= Q_{ss}(s, a) \\ Q(s_{sub}, a) &= \text{original value of } Q_{ss}(s, a) \end{aligned} \tag{4}$$

(c) control the strategy for the action selection in such a way that a conservative strategy is used around the normal states s and a high random strategy around the new substates s_{sub} in order to reduce the learning time.

For the first subtask (shooting behavior), we have already obtained the policy by using the state space shown in Figure 5. For the second subtask (avoiding behavior), we defined the substates for the opponent in the same manner to the substate of the ball in Figure 5. That is, a combination of the position (left, center, and right) and the size (small, medium, and large) is used.

A typical example of inconsistent states is the case where the ball and the opponent are located at the same area and the ball is occluded by the opponent from the viewpoint of the robot. In this case, the robot cannot observe the ball, and therefore the corresponding state for shooting behavior might be the state of "ball-lost," but it is not correct. Of course, if both the ball and the opponent can be observed, this situation can be considered consistent. This problem is resolved by adding new substates $s_{sub} \in \boldsymbol{S}$. In the above example, a new situation "occluded" is found by estimating the current state from the previous state, and the corresponding new substates are generated (see [12] for more detail).

5.2 Simulation

Based on the LEM algorithm, we limit the opponent's behavior when the robot learns. If the opponent has learned the professional techniques to keep the goal, the robot might not be able to learn how to shoot the ball into the goal anymore because of almost no goals. From this viewpoint, the opponent's behavior is scheduled so that the shooting robot has its confidence to shoot a ball into the goal.

In the simulation the robot has succeeded to acquire a behavior for a shooting the ball into the goal (see Figure 9). In the figure, the black is the learner and the white is the opponent. In (a), the robot watches the ball and the opponent. In (b),(c), and (d), the robot avoids the opponent and moves toward the ball. In (e) and (f), the robot shoots the ball into the goal.

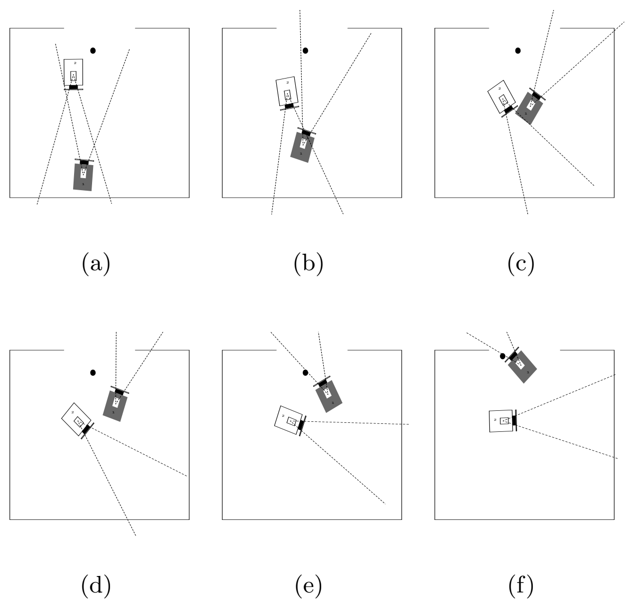


Fig. 9. The robot succeeded in shooting a ball into the goal

6 Experimental Result in RoboCup-97

We participated the middle size robot league of RoboCup-97 with five robots: four attackers and one goalie. For the goalie, we defined a set of rules and implemented on it as a goal keeping behavior. For the attackers, we implemented the behavior obtained by the simulation described in section 4.2 and 5.2.

Our team had five matches in total; two preliminary, two exhibition matches and the final. The result is shown in Table 1. Figure 10 and Figure 11(a) show a scene of a match, in which an attacker shoots the ball and the goalie keeps the goal respectively. Figure 11(b) is the view of the goalie in the situation of Figure 11(a). Our robot could get two goals in total, because four of two goals were own goals by the opponent team (USC).

7 Conclusions

In this paper, we have explained two of our reinforcement learning methods applied for real robot tasks tested in RoboCup-97. Our robots had learned a shooting behavior and a shooting behavior with avoiding an opponent, and played five matches there. They got two goals during more than 50 minutes of total playing time (time of one match was 10 minutes).

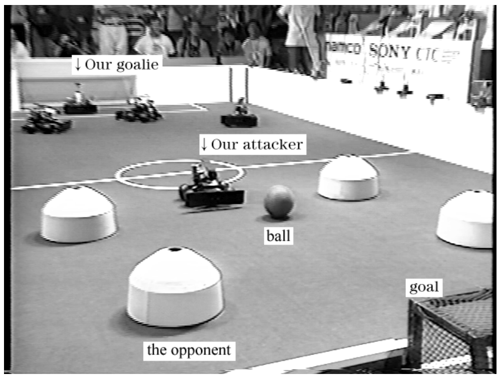
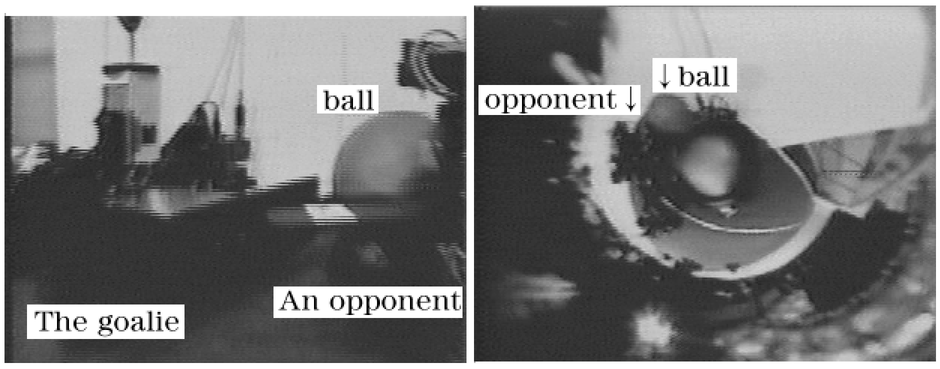


Fig. 10. One attacker shoots the ball



(a) The goalie and an opponent (b) The view of the goalie

Fig. 11. A behavior of the goalie

We are difficult to say that the robot performed the task well. However, getting two goals means that the robot could performed the task when it met a certain situation. This fact shows a potential ability of reinforcement learning methods to make the robot adapt to the real environment.

There are some reasons why the performance was not good enough. We had a trouble with color recognition because of noise on image transmission and uneven lighting condition on the field. Especially there were a plenty of noise sources around the field and the image became black and white so often. Though these problems are beyond the scope of our research issue, treatment of these

date	match	opponent team	score		result
25 August	preliminary	RMIT Raiders	0-1	us	win
26 August	preliminary	USC Dreamteam	2-2	us	draw
27 August	exhibition	UTTORI United	0-1	us	win
28 August	final	USC Dreamteam	0-0	us	draw
28 August	exhibition	The Spirit of Bolivia	1-0	us	lose

Table 1. The Result of matches

problems will improve the performance of the task.

A problem of our methods was construction of the state space. We ignored the case when the robot watches several robots in its view at a time, though nearly 10 robots existed on the field in every matches. In our future work, we need to focus state construction in a multi robot environment. Some topics have been already started, such as self construction of states by the robot [13],[14] and estimation and prediction of an opponent's behavior [16].

References

1. Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., Osawa, E., Matsubara, H.: RoboCup: A Challenge Problem of AI. AI Magazine **18** (1997) 73–85
2. Bellman, R.: Dynamic Programming. Princeton University Press (1957)
3. Connel, J. H., Mahadevan, S.: Robot Learning. Kluwer Academic Publishers (1993)
4. Kaelbling, L. P.: Learning to Achieve Goals. Proc. of IJCAI-93 (1993) 1094–1098
5. Mataric, M.: Reward Functions for Accelerated Learning. In Proc. of Conf. on Machine Learning-1994 (1994) 181–189
6. Watkins, C. J. C. H., Dayan, P.: Technical note: Q-learning, Machine Learning **8** (1992) 279–292
7. Whitehead, S. D., Ballard, D. H.: Active Perception and Reinforcement Learning. In Proc. of Workshop on Machine Learning-1990 (1990) 179–188
8. Whitehead, S. D.: Complexity and Coordination in Q-Learning, In Proc. of the 8th International Workshop on Machine Learning (1991) 363–367
9. Inaba, M.: Remote-Brained Robotics: Interfacing AI with Real World Behaviors. In Preprints of ISRR'93 (1993)
10. Yagi, Y., Kawato, S.: Panoramic Scene Analysis with Conic Projection. Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems (1990)
11. Asada, M., Noda, S., Tawaratsumida, S., Hosoda, K.: Purposive Behavior Acquisition for a Real Robot by Vision-Based Reinforcement Learning. Machine Learning **23** (1996) 279–303
12. Asada, M., Uchibe, E., Noda, S., Tawaratsumida, S., Hosoda, K.: Coordination Of Multiple Behaviors Acquired By A Vision-Based Reinforcement Learning. Proc. of the 1994 IEEE/RSJ International Conference on Intelligent Robots and Systems (1994) 917–924

13. Asada, M., Noda, S., Tawaratsumida, S., Hosoda, K.: Vision-Based Reinforcement Learning for Purposive Behavior Acquisition. Proc. of the IEEE Int. Conf. on Robotics and Automation (1995) 146–153
14. Takahashi, Y., Asada, M., Noda, S., Hosoda, K.: Sensor Space Segmentation for Mobile Robot Learning. Proceedings of ICMAS'96 Workshop on Learning, Interaction and Organizations in Multiagent Environment (1996)
15. Uchibe, E., Asada, M., Hosoda, K.: Behavior Coordination for a Mobile Robot Using Modular Reinforcement Learning. Proc. of the 1996 IEEE/RSJ International Conference on Intelligent Robots and Systems (1996) 1329 – 1336
16. Uchibe, E., Asada, M., Hosoda, K.: Vision Based State Space Construction for Learning Mobile Robots in Multi Agent Environments. Proc. of Sixth European Workshop on Learning Robots(EWLR-6) (1997) 33–41