Proc. of MLC-COLT (Machine Learning Confernce and Computer Learning Theory) Workshop on Robot Learning, Rutgers, New Brunswick, July 10, 1994

Purposive Behavior Acquisition On A Real Robot By A Vision-Based Reinforcement Learning

Minoru Asada, Shoichi Noda, Sukoya Tawaratsumida, and Koh Hosoda
Dept. of Mech. Eng. for Computer-Controlled Machinery
Osaka University, 2-1, Yamadaoka, Suita, Osaka 565, Japan
asada@robotics.ccm.eng.osaka-u.ac.jp

Abstract

In [1], we have presented the soccer robot which had learned to shoot a ball into the goal using the Q-learning. In this paper, we discuss several issues in applying the Q-learning method to a real robot with vision sensor. First, to speed up the learning rate, we implement a mechanism of *Learning form Easy Missions* (or LEM) which is a similar technique to "shaping" in animal learning. LEM reduces the learning rate from the exponential order in the size of the state space to about the linear order of the size of the state space. Also, we save the learning time by *policy transfer* in which the goal-directed behavior is first acquired by Q-learning in computer simulation, and then the learned policy is copied into a real robot brain. Next, a "state-action deviation" problem is found as a form of perceptual aliasing when we try to construct such a state (action) space that reflects the outputs from physical sensors (actuators). To cope with this, we construct an action set in such a way that one action consists of a series of the same action primitives which is successively executed until the current state changes. We give the results of computer simulation and real robot experiments.

1 Introduction

The ultimate goal of AI and Robotics is to realize autonomous agents that organize their own internal structure in order to behave adequately with respect to their goals and the world. That is, they learn. As a method for robot learning, reinforcement learning has recently been receiving increased attention with little or no *a priori* knowledge and higher capability of reactive and adaptive behaviors [2]. In the reinforcement learning scheme, a robot and environment are modeled by two synchronized finite state automatons interacting in a discrete time cyclical processes. The robot senses the current state of the environment and selects an action. Based on the state and 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.

Although the role of the reinforcement learning is very important to realize autonomous systems, the prominence of that role is largely determined by the extent to which it can be scaled to larger and complex robot learning tasks. Many theoretical works have argued the convergence time of the learning, and how to speed up it by using some techniques and to extend these techniques from a single goal task to multiple ones [3]. However, almost of them have only shown computer simulations in which they assume ideal sensors and actuators, and therefore they can construct consistent state and action spaces. A typical example is a 2-D grid environment in which a robot can take an action of forward, backward, left move, or right move, and its state is encoded by the coordinate of the grid, that is, the absolute (therefore global) positioning system is assumed. Although the uncertainties of sensor and actuator outputs are taken into account by stochastic transitions in the state space, it does not seem realistic because the localization of the robot is sensitive to sensor noise which can be easily accumulated into non-negligible amount.

From a viewpoint of real robot applications, we should construct a state space so that it can reflect the outputs of the physical sensors which are currently available and can be mounted on the robot. These sensors generally output the relative and often local information of the environment. The following two works deal with such sensors.

Mahadevan and Connenl [4] proposed a method of rapid task learning on real robot. They separated a pushing task into three subtasks of "finding a box," "pushing a box," and "getting unwedged," and ap-

plied the Q-learning to each of them. Two algorithms for clustering the state space are implemented in each subtask. Since they used the proximity sensors such as bumper and sonar sensors, the task is limited to local behavior and not suitable for the goal-directed more global task such as carrying a box to the specified location. To cope with more global tasks, the vision sensor seems more suitable. However, the use of vision in the reinforcement learning is somehow very rare.

To the best of our knowledge, only Whitehead and Ballard [5] argued this problem. Their task is a simple manipulation of blocks on the conveyer belt. Although each block is colored to be easily discriminated, they still have a large size of state space. To cope with this problem, they assumed that observer could control its gaze to attended object so as to reduce the size of the state space. However, this causes so-called "perceptual aliasing" problem. That is, both the observer motion and actual changes happened in the environment cause the changes inside the image captured by the observer. Therefore, it seems difficult to discriminate the both from only the image. Then, they proposed a method to cope with this problem by adopting the internal states and separating action commands into "Action frame" and "Attention frame" commands. However, they have not shown the real experiments.

In [1], we have presented a soccer robot which had learned to shoot a ball into the goal using the Q-learning. The robot does not need to know any parameters of the 3-D environment or its kinematics/dynamics. Information about the changes of the environment is only the image captured from a single TV camera mounted on the robot. Image positions of the ball and the goal are used as state variables. In this paper, we discuss several issues dealt in the method from a viewpoint of robot learning: a) learning from easy missions mechanism for rapid task learning instead of task decomposition, and b) coping with a "state-action deviation" problem which occurs in constructing state and action spaces in accordance with outputs from physical sensors and actuators.

The remainder of this article is structured as follows: In the next section, we show the problems we face with in applying Q-learning scheme to real robot applications, then we give a brief overview of the Q-learning. Next, we explain the task and basic assumptions, and the learning scheme in our method. Finally, we show the experiments with computer simulations and a real robot system, and give a conclusion.

2 Problems in Applying Q-learning to Real Robot Applications

(a) learning rate

In order to reduce the learning rate, the whole task was separated by the programmer in [4]. However, we have not separated the shooting task into subtasks of "finding a ball," "dribblling a ball," and "shooting a ball." This is just a monolithic approach. To reduce the learning time, we implement a mechanism of *Learning from Easy Missions* (or LEM) which is similar to a widely known "shaping" technique in animal learning [6]. In the LEM scheme, the robot begins to learn the behavior from easy missions such as shooting a ball given the ball and the robot are located near the goal. This approach reduces the learning time from the exponential order in the size of the state space [7] to about the linear order in the size of the state space.

The difference between task decomposition and LEM (or easy mission specification) can be explained as follows; the task decomposition should be complete in the sense of constructing the whole task while LEM does not always need the completeness of the easy mission specification although the reduction of the learning rate into linear order depends on the completeness. That is, partial knowledge about easiness of the mission can be used in LEM scheme, but such knowledge seems difficult to be used in the task decomposition scheme.

A strategy of different aspect to save the learning rate is "a policy transfer" in which the goal-directed behavior is first acquired by Q-learning in computer simulation, and then the learned policy is copied into a real robot brain. The policy transfer is useful not only to save the learning rate but also to find the differences between computer simulations and real experiments.

(b) a state-action deviation problem

In order to realize a shooting behavior, a goal-directed global behavior, we adopt the vision sensor. One of the perceptual aliasing problems [5] caused due to the lack of the reference point in the environment. In our case, the goal post is fixed on the ground plane, and one action causes both body and camera motions together. Therefore we do not need to discriminate between "Action frame" and "Attention frame" commands. Instead, another kind of perceptual aliasing problem, a "state-action deviation" problem occurs due to the peculiarities of the visual information and its processing. If it is located far from the goal, the robot needs a number of forward motions for state transition from "the goal is far" to "the goal is near." While, one action near the goal might be sufficient to shoot a ball. To cope with this problem, we construct an action space in such a way that an action consists of a sequence of the same action primitives, and one action primitive is successively executed until the current state changes.

Another aspect of this problem is that there is a delay due to image acquisition and processing. Usually, it takes one video frame rate (1/30 sec.) to acquire one image and at least one more frame rate for image processing. This means that what the robot perceives

(state discrimination) is what the environment was (two video frame rates ago). This makes the convergence of the learning difficult due to the wider variations of state transitions than in the case of less delay. If the robot is located far from the goal, this does not cause serious situations because we construct the state space very coarsely which absorbs a small amount of delay. However, the robot might take actions not suitable for shooting near the goal. To avoid these situations, we first obtain the policy without almost no delay in computer simulation, and then we apply it to the real robot experiments.

3 Q-learning

Before getting into the details of our system, we briefly review the basics of the Q-learning. For more through treatment, see [8]. We follow the explanation of the Q-learning by Kaelbling [9].

We assume that the robot can discriminate the set $\boldsymbol{S}$ of distinct world states, and can take the set $\boldsymbol{A}$ of actions on the world. The world 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 that the world will transit to the next state s' from the current state-action pair (s, a). For each state-action pair (s, a), the *reward* $r(s, a)$ is defined.

The general reinforcement learning problem is typically stated as finding a policy that maximizes discounted sum of the reward received over time. A policy f is mapping from $\boldsymbol{S}$ to $\boldsymbol{A}$. This sum called the *return* and is defined as:

$$\sum_{n=0}^{\infty} \gamma^n r_{t+n}, \tag{1}$$

where r_t is the reward received at step t given that the agent started in state s and executed policy f. γ is the discounting factor, it controls to what degree rewards in the distant future affect the total value of a policy and is just slightly less than 1.

Given definitions of the transition probabilities and the reward distribution, we can solve the optimal policy, using methods from dynamic programming [10]. A more interesting case occurs when we wish to simultaneously learn the dynamics of the world and construct the policy. Watkin's Q-learning algorithm gives us an elegant method for doing this.

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

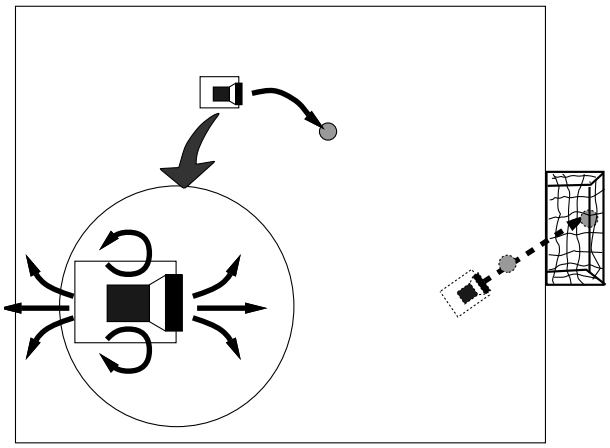
$$Q^*(s, a) = r(s, a) + \gamma \sum_{s' \in S} T(s, a, s') \max_{a' \in A} Q^*(s', a'). \tag{2}$$

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

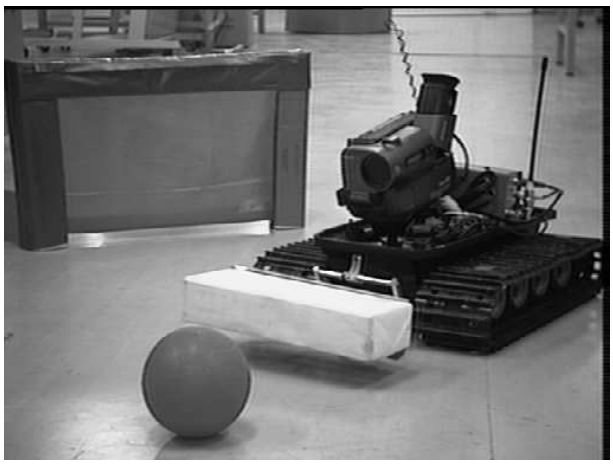
$$Q(s, a) \Leftarrow (1 - \alpha) Q(s, a) + \alpha (r(s, a) + \gamma \max_{a' \in A} Q(s', a')). \tag{3}$$

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 leaning rate (between 0 and 1).

4 Task and Assumptions



(a) The task is to shoot a ball into the goal.



(b) A picture of the radio-controlled vehicle.

Figure 1: Task and our real robot.

The task for a mobile robot is to shoot a ball into the goal as shown in **Fig.1**(a). The problem we are

attacking here is to develop a method which automatically acquires strategies how to do this. We assume that the environment consists of a ball and a goal, and a mobile robot has a single TV camera, and that the robot does not know location and size of the goal, size and weight of the ball, any camera parameters such as focal length and tilt angle, or kinematics/dynamics of itself.

Fig.1(b) shows a picture of the real robot with a TV camera (Sony handy-cam TR-3) and video transmitter. **Fig.2** shows a sequence of the images in which the robot succeeded in shooting a ball into the goal by the method.

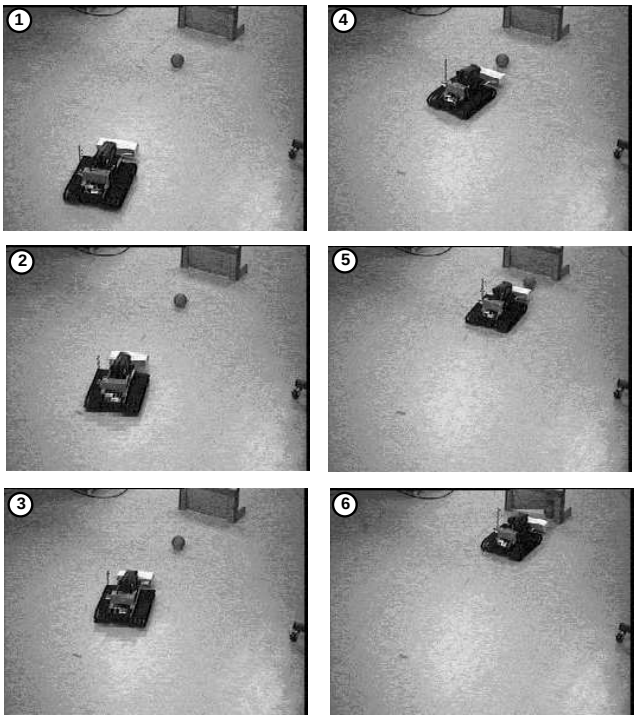


Figure 2: The robot succeeded in shooting a ball into the goal.

5 Construction of State and Action Sets

In order to apply the Q-learning scheme to the task, we define a number of sets and parameters. Many existing applications of the reinforcement learning schemes have constructed the state and action spaces in such a way that each action causes the state transition (ex. one action is forward, backward, left, or right, and states are encoded by the locations of the agent) in order to make the quantization problem (the structural credit assignment problem) easy. This makes a gap between the computer simulations and real robot systems. Each space should reflect the corresponding physical space in which a state (an action) can be perceived (taken). However, such construction of state and action spaces sometimes causes one kind of perceptual aliasing problem; "state-action deviation" problem. In the followings, we describe how to construct the state and action spaces, and then how to cope with the state-action deviation problem.

5.1 Construction of Each Space

(a) a state set S

Only the information the robot can obtain about the environment is the image supposed to be capturing the ball and/or the goal. The ball image is quantized into 9 sub-states, combinations of three positions (left, center, and right) and three sizes (large (near), middle, and small (far)). The goal image has 27 sub-states, combinations of three parameters each of which is quantized three parts. Each sub-state corresponds to one posture of the robot toward the goal, that is, position and orientation of the robot in the field. In addition to these 243 (27×9) states, we add other states such as these cases in which only the ball or the goal is captured in the image. Totally, we have 319 states in the set $\boldsymbol{S}$.

After some simulations, we realized that as long as the robot is capturing the ball and the goal positions in the image, it succeeds in shooting a ball. However, once it lost the ball, it randomly moves because it does not know to which direction it should move to find the ball. This causes because the ball-lost state is just one, therefore it cannot discriminate to which direction the ball is lost. Then, we separate the ball-lost state into two states; the ball-lost-into-right and the ball-lost-into-left states. Also, we set up goal-lost-into-right and goal-lost-into-left states, too. This improved the robot behavior much better.

(b) an action set A

The robot can select an action to be taken against the environment. In real system, the robot moves around the field by a PWS (Power Wheeled Steering) system with two independent motors. Since we can send the motor control command to each of two motors independently, we quantized the action set in terms of two motor commands ω_l and ω_r, each of which has 3 sub-actions (forward, stop, and back motions, respectively). Totally, we have 9 actions in the action set $\boldsymbol{A}$.

(c) a reward and a discounting factor γ

We assign a reward value 1 when the ball was entered into the goal or 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, 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 a slightly less than 1 ($\gamma^g = 0.8$).

5.2 Solving A State-Action Deviation Problem

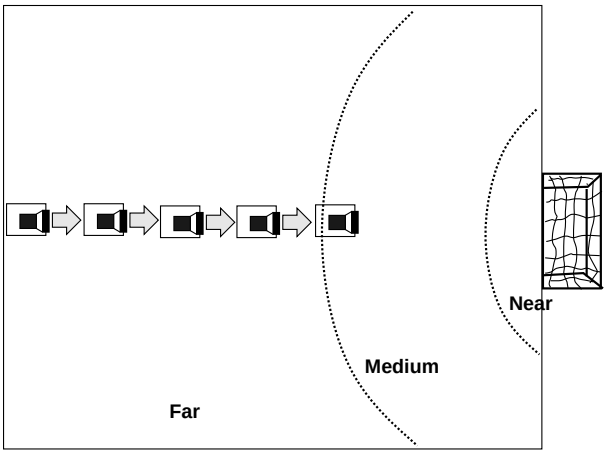


Figure 3: A state-action deviation problem

In 5.1, we constructed the state space in such a way that the position and the size of the ball or goal are naturally and coarsely quantized into each state. The peculiarity of the visual information, that is, a small change near the observer might cause a large change in image and vice versa, causes a state-action deviation problem because each action produces almost the same amount of motion in the environment. Fig.3 indicates this problem, where the area of which state is "the goal is far" has a large area, and therefore the robot frequently transits to the same state if the action is forward. This is highly undesirable because the variance of the state transitions is vary large, and therefore the learning does not converged correctly. In the case of Fig.3, the major transition from the state "the goal is far" is returning to the same state, and we cannot obtain the optimal policy.

Then, we reconstruct the action space as follows. Each action defined in 5.1 is regarded as an action primitive. The robot continues to take one action primitive until the current state changes. This sequence of the action primitive is called an action. In the above case, the robot takes a forward motion many times until the state "the goal is far" changes into the state "the goal is middle." The number of action primitives needed for state changes has no meanings. Once the state has changed, we apply the update equation (3 of the action value function.

6 Learning from Easy Missions

Unlike the approach in [4], we do not decompose the whole task into subtasks of finding, dribblling, and shooting a ball. Instead, we first used a monolithic approach. That is, we set the ball and the robot at arbitrary positions. In almost cases, the robot crossed over the field line without shooting a ball into the goal. This means that the learning has not converged after many trials (a week running on SGI Elan with R4000). This situation resembles a case that a small child tries to shoot a ball into the goal, but he cannot imagine in which direction and how far the goal is because a reward is received just after the ball is entered into the goal. Further, he does not know which action to select. This is the famous *delayed reinforcement* problem due to no explicit teacher signal that indicates the correct output at each time step. Then, we construct the learning schedule such that the robot can learn in easy situations at the early stage and learn in more difficult situations at the later stage. We call this *Learning from Easy Missions* (or LEM). This technique is similar to a widely known "shaping" technique in animal learning in letting an agent know how to achieve the goal.

Figure 4: The simplest state space.

Instead of critical analysis of the time complexity for LEM, we give an intuitive explanation for it by using a very simple example. Following the complexity analysis by Whitehead [7], we assume the "homogeneous" state space uniformly k-bounded with polynomial width of the depth k and zero-initialized Q-learning. Further, we assume that state transition is deterministic and the robot can take m-kinds actions with equal opportunities. In order to figure out how many steps are needed to converge the Q-learning, we use $O(k)$ state space and simplify the convergence such that the action value function converged if it is updated from the initial value (0) [1].

Fig.4 show an example of such state spaces. Since we assigned a reward 1 when the robot achieves the goal and 0 otherwise, the unbiased Q-learning takes long time. From the above formulation, it needs m trials to transit from the initial state S_k to the state S_{k-1} in the worst case, therefore it takes m^k trials to achieve the goal for the first time in the worst case, and the value of the action value function for only the state

[1] Strictly speaking, this might be incorrect, however, it seems easy to figure out the order of the search time.

S_1 is updated. Next, it needs m^{k-1} trials to update the value of the action value function for the state S_2, and totally it need $(m^k + m^{k-1} + \cdots + m)$ trials to converge the value of the action value function for all the states. Therefore, the unbiased Q-leaning can be expected time moderately exponential in the size of k [7].

While, in the Learning from Easy Missions algorithm, we set the agent at the state S_1 first and make it try to achieve the goal. In the worst case, it takes m trials. Then, we set the agent at the state S_2 and repeat it. In the worst case, it needs $m \times k$ trials to converge the action value function. Therefore, the LEM algorithm requires about linear in the size of k.

In actual situations like our task, the state transition is stochastic, the state space is not homogeneous, and therefore it seems difficult to correctly decide which state is an easy one to achieve the goal and when to shift the initial situations into more difficult ones. Since the convergence to the optimal policy is guaranteed in the Q-learning scheme, we roughly collect the easy states S_1 in which the agent can achieve the goal with high probability and shift to a slightly more difficult situations when

$$\Delta Q_t(S_1, a) < \epsilon \quad \text{and} \quad 0 < \epsilon \ll 1, \tag{4}$$

where

$$\Delta Q_t(S_1, a) = |\sum_{a \in \boldsymbol{A}} Q_t(S_1, a) - \sum_{a \in \boldsymbol{A}} Q_{t-1}(S_1, a)|. \tag{5}$$

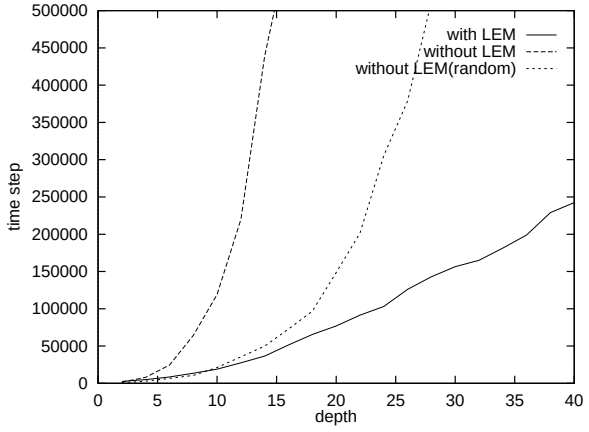


Figure 5: Search time complexity as a function k.

Fig.5 shows a plot of the search time versus maximum distance k for a simple *get food* problem in the 2-D grid environment, where one step Q-learning algorithm is applied. As we expected, the search time with LEM (solid line) is almost linear in the size of k while that of the normal Q-learning without LEM (dotted and broken lines) indicates the exponential order in the size of k. The initial position is fixed (broken line) or randomly positioned (dotted line).

The LEM algorithm differs in some aspects from the existing approaches to speed up the search time. In the task decomposition approach [4], the Q-learning is closed inside each subtask. In LEM, however, the robot wanders around the field crossing over the easy states to achieve the goal even if we initially set it at such states. We just advise the position of the easy state. In other words, we do not need to care so much about the segmentation of the state space in order to decompose the whole task.

In the Learning with an External Critic (or LEC) [7], the robot receives an advise in each state from the external critic. In order to let LEC work correctly, the complete knowledge about the path to the absorbing goal is needed. While, the partial knowledge is available in LEM. The completeness of the knowledge does not make any effect on the correct convergence of Q-learning, but on the search time in LEM.

7 Experiments

The experiment consists of two parts: first, learning the optimal policy f through the computer simulation, then apply the learned policy to a real situation. The merit of the computer simulation is not only to check the validity of the algorithm but also to save the running cost of the real robot during the learning process. Further, this *policy transfer* helps us improve the system by finding bugs in the simulation program and difference between the simulation and the real robot system. The computer simulation cannot completely simulate the real world [11].

7.1 Simulation

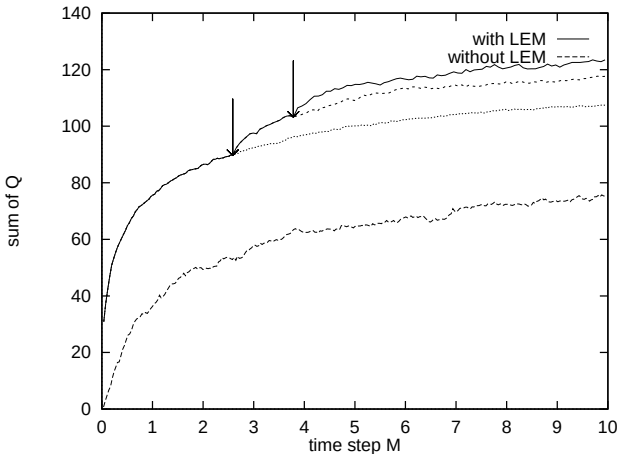


Figure 6: Change of the sum of Q-values.

We performed the computer simulation with the following specifications (the unit is an arbitrary-scaled length). The field is a square of which side length is

200. The goal post is located at the center of the top line of the square (see Fig.1) and its height and width are 10 and 50, respectively. The robot is 16 wide and 20 long and kicks a ball of which diameter is 6. The camera is horizontally mounted on the robot (no tilt), and its visual angle is 30 degrees. These parameters are selected so that they can roughly simulate the real world. Therefore, they are not so accurate.

Following the LEM algorithm, we began the learning of the shooting behavior by setting the ball and the robot near the goal. Once the robot succeeded in a shooting, the robot begin to learn (the sum of Q is increasing), but after that the robot wonders again in the field. After many iterations of these successes and failures, the robot learned to shoot a ball into the goal when the ball is near the goal. After that, we set the ball and the goal slightly further from the goal, and repeat the robot learning again.

Fig.6 shows the change of the sum of Q-values with (solid line) or without (broken line) LEM. The Q-learning with LEM is much better than that without LEM. Two arrows indicate the time step at which we changed the initial position from S_1 (S_2) to S_2 (S_3). Fine and coarse dotted lines show the curve when the initial position was not changed. This simulates the LEM with partial knowledge. If we know only the easy situation of S_1, and nothing more, the learning curve follows the fine dotted line in 6. The sum of Q values is slightly less than that of the LEM with more knowledge, but much better than without LEM.

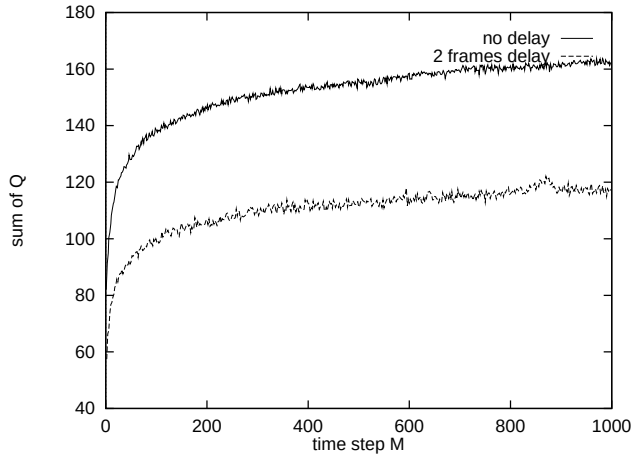


Figure 7: Change of the sum of Q-values in terms of delay time.

In the above simulations shown in **Fig.6**, the delay of image acquisition and processing is set almost zero. However, a real system needs at least one video frame (1/30 sec.) for image acquisition and one more video frame for image processing (state discrimination). **Fig.7** shows the changes of the sum of Q-values in terms of the delay time. The solid and broken curves indicate the learning curves with no delay and two video frames delay, respectively. Evidently, the learning curve with delay is worse than that of no delay, and also its performance is not so good as that of no delay. Also, we compared the performance of the learned policies with and without delay assuming that the real robot needs two video frames time (1/15 sec.). The shooting rates are 70% with the policy of no delay and 60% with the policy of delay. Therefore, we copy the leaned policy of no delay to a real robot.

7.2 Real System

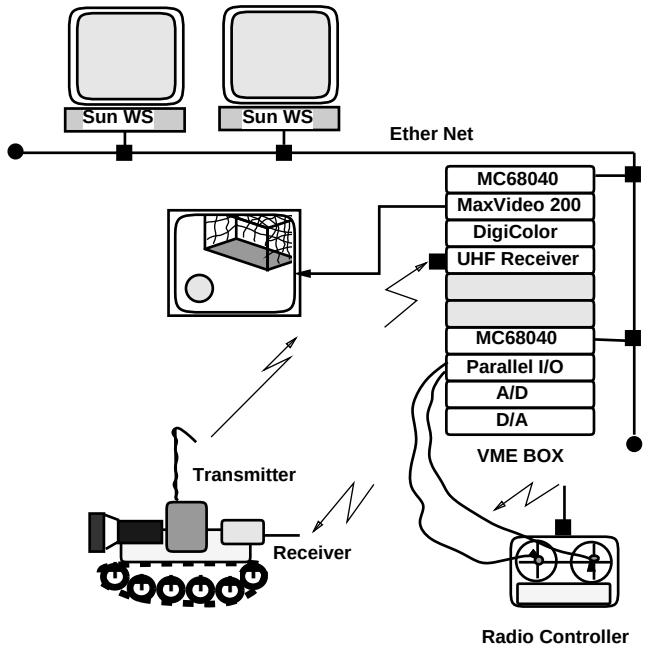


Figure 8: A configuration of the real system

Fig.8 shows a configuration of the real mobile robot system. The image taken by a TV camera mounted on the robot is transmitted to a UHF receiver and processed by Datacube MaxVideo 200, a real-time pipeline video image processor. In order to simplify and speed up the image processing time, we painted the ball in red and the goal in blue. We constructed the radio control system of the vehicle, following the remote-brain project by Profs. Inaba and Inoue at University of Tokyo [12]. The image processing and the vehicle control system are operated by VxWorks OS on MC68040 CPU which are connected with host Sun workstations via Ether net. We have shown a picture of the real robot with a TV camera (Sony handy-cam TR-3) and video transmitter in Fig.1(b).

Fig.9 shows a flow of image processing where input NTSC color video signal is first converted into HSV color components in order to make extraction of a red

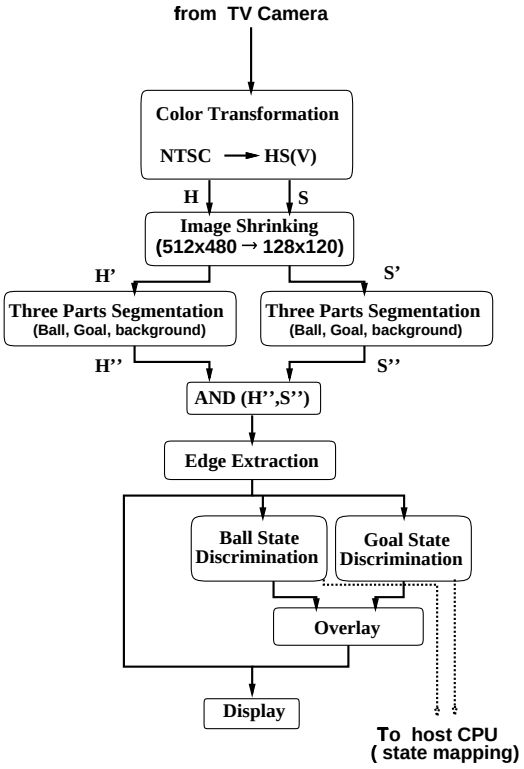
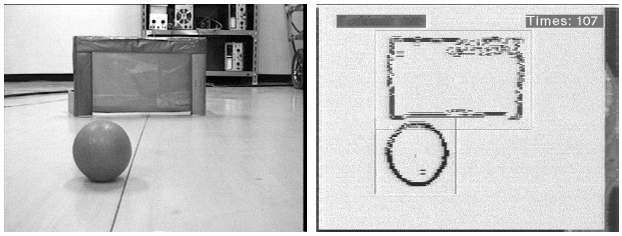


Figure 9: A flow of image processing

ball and a blue goal easy. Then, the image size is reduced to speed up the image processing time, and boundaries of two region are extracted for state discrimination. The result of image processing are sent to the host CPU to decide a optimal action against the current state.

The shooting rate in the real robot system was less than 40% which was more than 20% worse than the simulation. The main reason is that the ball often moves towards unpredictable directions due to its eccentricity of the centroid. The second one is noise of the image processing explained in the following.

Fig.10 (a) and (b) show a result of the image processing where the ball and the goal are detected and their positions are calculated in real time (1/30 seconds). Table 1 shows the image processing and state mapping result in each time stamp (1/30 sec.) for the sequence of the images captured by the robot in **Fig.2**. Each column indicates time step (1/30 sec.), the state transition step, mapped state, action command, and error, respectively. The state transition number shows the state the robot could discriminate. The mapped state consists of five substates: two for ball position (**L**eft, **C**enter, or **R**ight) and size (large (**N**ear), **M**iddle, or small (**F**ar)), and three for the goal position, size, and orientation (**L**eft-**o**riented, **F**ront-



(a) input image (b) detected image

Figure 10: Result of image processing

oriented, or **R**ight-**o**rinented). "D" means a lost state (disappear). Incorrectly mapped substates are with "*"s, and the number of these substates are shown in error box. Action commands consist of a combination of two independent motor commands (**F**orward, **S**top, or **B**ackward).

Amazingly, the ratio of the completely correct mappings is about 60%. Almost incorrect mapping occurs when the size of the ball is misjudged as smaller one due to mistakes in edge detection or small up-down motions of the robot. As long as the ball and the goal are captured at the center of the image, this does not cause serious situations because the optimal action is just forward. However, it fails to shoot a ball when the ball is captured at the right or left of the image because it has to follow a curved path and misjudges the distance to the ball. Due to the noise of the transmitter, completely incorrect mapping occurs at the ratio of 15%. Unless this situation continues two or more time steps, the robot can obtain the almost correct state mapping and therefore almost correct action can be executed. In our experiments the action execution seldom fails because each action consists of a number of action primitives, and consecutive failure of the action primitives is very rare. However, also we have some delay in changing from the forward motion to the backward one.

8 Conclusion and Future Works

We have shown a vision-based reinforcement learning on real robot system, which adopted the Learning from Easy Missions algorithm similar to a "shaping" technique in animal learning in order to speed up the learning rate instead of task decomposition. The state-action deviation problem due to the peculiarity of the visual information is pointed out as one of the perceptual aliasing problem in applying the Q-learning to real robot application, and we constructed an action space to cope with this problem.

The delay due to image acquisition and processing

Table 1: State-Action data

time step	state step	state		action		errer
		ball	goal	L	R	
1	1	(C,F)	(C,F,Fo)	F	F	
2	2	(R*,F)	(C,F,Fo)	F	F	1
3	3	(D*,D*)	(C,F,Ro*)	B	B	3
4	4	(C,F)	(C,F,Lo*)	B	S	1
5	5	(C,F)	(C,F,Fo)	F	F	
6		(C,F)	(C,F,Fo)	F	F	
7		(C,F)	(C,F,Fo)	F	F	
8		(C,F)	(C,F,Fo)	F	F	
9	6	(C,F)	(C,F,Ro*)	B	S	1
10	7	(C,F)	(C,F,Fo)	F	F	
11	8	(C,F)	(R,M,Fo)	F	F	
12	9	(R,F)	(R,M,Fo)	F	F	
13	10	(R,M*)	(R,F*,Lo*)	F	B	3
14	11	(L*,F)	(R,M,Ro*)	F	S	2
15	12	(L*,F)	(R,M,Fo)	F	S	1
16	13	(R,M)	(R,M,Fo)	S	B	
17	14	(C,M)	(C,M,Fo)	F	F	
18	15	(L,M)	(L,M,Fo)	S	F	
19	16	(L,N)	(L,M,Fo)	B	S	
20		(L,N)	(L,M,Fo)	B	S	
21	17	(L,M*)	(L,M,Fo)	S	F	1
22	18	(L,N)	(L,M,Fo)	B	S	
23		(L,N)	(L,M,Fo)	B	S	
24	19	(C,N)	(C,M,Fo)	F	B	
25	20	(C,M)	(C,M,Fo)	F	F	
26		(C,M)	(C,M,Fo)	F	F	
27	21	(C,M)	(C,N,Fo)	F	S	
28	22	(C,M)	(C,M*,Lo*)	F	S	2
29	23	(C,M)	(C,M*,Ro*)	S	B	2
30	24	(C,F)	(D,D,D)	F	S	

causes serious situations near the goal because the state segmentation around here seems too coarse to find the optimal action and delay of state discrimination sometimes fatal for shooting behavior. The method of dynamic construction of the state space considering the delay would be necessary. This is now under the investigation.

Although the real experiments are encouraging, still we have a gap between the computer simulation and the real system. We have not made the real robot learn but only execute the optimal policy obtained by the computer simulation. We are planning to make the real robot begin to learn from the policy.

As one extension of the work here, we have done some simulations of obtaining a shooting behavior avoiding a goal keeper [13]. Three kinds of coordinations of different behaviors (shooting and avoiding) are simulated and compared with each other. Now, we try to transfer the learned policy to real robots system.

References

[1] M. Asada, S. Noda, S. Tawaratsumida, and K. Hosoda. "Vision-based behavior acquisition for a shooting robot by using a reinforcement learning". In *Proc. of IAPR / IEEE Workshop on Visual Behaviors-1994*, pages 112–118, 1994.

[2] J. H. Connel and S. Mahadevan, editors. *Robot Learning*. Kluwer Academic Publishers, 1993.

[3] R. S. Sutton. "Special issue on reinforcement learning". In R. S. Sutton(Guest), editor, *Machine Learning*, volume 8, pages –. Kluwer Academic Publishers, 1992.

[4] J. H. Connel and S. Mahadevan. "Rapid task learning for real robot". In J. H. Connel and S. Mahadevan, editors, *Robot Learning*, chapter 5. Kluwer Academic Publishers, 1993.

[5] S. D. Whitehead and D. H. Ballard. "Active perception and reinforcement learning". In *Proc. of Workshop on Machine Learning-1990*, pages 179–188, 1990.

[6] J. M. Pearce. *Introduction to Animal Learning*. Lawrence Erlbaum Associate Ltd., 1987.

[7] S. D. Whitehead. "A complexity analysis of cooperative mechanisms in reinforcement learning". In *Proc. AAAI-91*, pages 607–613, 1991.

[8] C. J. C. H. Watkins. *Learning from delayed rewards"*. PhD thesis, King's College, University of Cambridge, May 1989.

[9] L. P. Kaelbling. "Learning to achieve goals". In *Proc. of IJCAI-93*, pages 1094–1098, 1993.

[10] R. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, 1957.

[11] R. A. Brooks and M. J. Mataric. "Real robot, real learning problems". In J. H. Connel and S. Mahadevan, editors, *Robot Learning*, chapter 8. Kluwer Academic Publishers, 1993.

[12] M. Inaba. "Remote-brained robotics: Interfacing ai with real world behaviors". In *Preprints of ISRR'93*, Pitsuburg, 1993.

[13] M. Asada, E. Uchibe, S. Noda, S. Tawaratsumida, and K. Hosoda. "A vision-based reinforcement learning for coordination of soccer playing behaviors". In *Proc. of AAAI-94 Workshop on AI and A-life and Entertainment*, pages 16–21, 1994.