Vision-Based Reinforcement Learning for Purposive Behavior Acquisition

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

This paper presents a method of vision-based reinforcement learning by which a robot learns to shoot a ball into a goal, and discusses several issues in applying the reinforcement learning method to a real robot with vision sensor. First, a "state-action deviation" problem is found as a form of perceptual aliasing in constructing the state and action spaces that reflect the outputs from physical sensors and actuators, respectively. To cope with this, an action set is constructed in such a way that one action consists of a series of the same action primitive which is successively executed until the current state changes. Next, to speed up the learning time, a mechanism of *Learning form Easy Missions* (or LEM) which is a similar technique to "shaping" in animal learning is implemented. LEM reduces the learning time from the exponential order in the size of the state space to about the linear order in the size of the state space. The results of computer simulations and real robot experiments are given.

1 Introduction

Realization of autonomous agents that organize their own internal structure in order to behave adequately with respect to their goals and the world is the ultimate goal of AI and Robotics. That is, the autonomous agents have to learn. 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 [1]. In the reinforcement learning scheme, the robot and the 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.

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 researchers in the field of machine learning have concerned with the convergence time of the learning, and developed the methods to speed it up by some techniques and to extend these techniques from a single goal task to multiple ones [2]. However, almost all of them have only shown computer simulations in which they assume ideal sensors and actuators, and therefore they can easily construct the state and action spaces consistent with each other. From a viewpoint of real robot applications, we should construct the state space so that it can reflect the outputs of the physical sensors which are currently available and can be mounted on the robot.

Some applications are recently reported to control robot arms [3] or mobile robots [4] in which the initial controller and the correct reward function are given in advance. Therefore, the robot learns the control policy given much knowledge about the environment and itself. We intend to apply the reinforcement learning scheme to the task of purposive behavior acquisition in real world with less knowledge about the environment and the robot.

Mahadevan and Connel [5] proposed a method of rapid task learning on a real robot. They separated a pushing task into three subtasks of "finding a box," "pushing a box," and "getting unwedged," and applied the Q-learning, a widely used reinforcement learning method, to each of them. Since the proximity sensors such as bumper and sonar sensors are used, the acquired behaviors are limited to local ones and therefore these behaviors are not suitable for the more global and goal-directed tasks such as carrying a box to the specified location. For such tasks, visual sensors could be more useful because they might be able to capture the image of the goal in the distant place. However, the use of visual information in the reinforcement learning is very few [1] due to its processing cost.

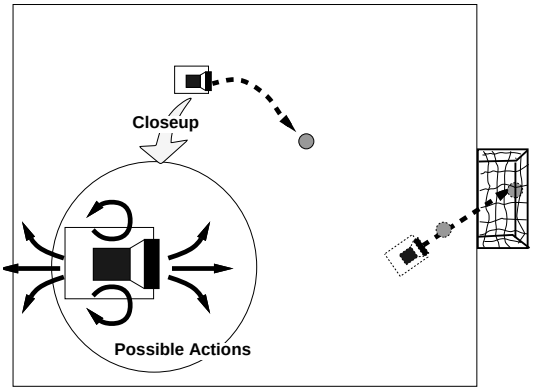
In this paper, we present a method of vision-based reinforcement learning by which a robot learns to shoot a ball into a goal. The robot does not need to know any parameters of the 3-D environment or its kinematics/dynamics. The image captured from a single TV camera mounted on the robot is only the source of the information telling the changes of the environment. Image positions and sizes of the ball and the goal are used as a state vector. We discuss several

[1] To the best of our knowledge, only Whitehead and Ballard [6] used the active vision system and argued the importance of so-called "perceptual aliasing" problem. However, they have not shown the real experiments.

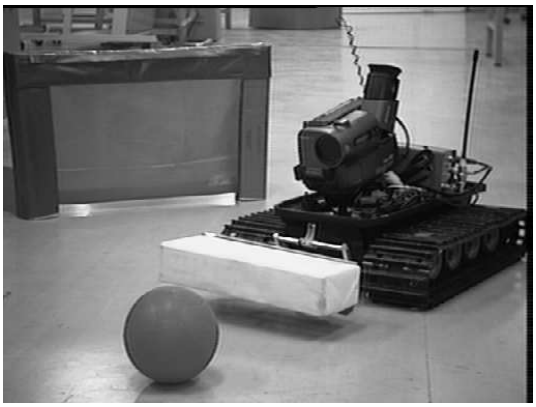
issues from a viewpoint of robot learning: a) coping with a "state-action deviation" problem which occurs in constructing the state and action spaces in accordance with outputs from the physical sensors and actuators, and b) learning from easy missions mechanism for rapid task learning instead of task decomposition.

The remainder of this article is structured as follows: In the next section, we explain the task and assumptions, and give a brief overview of the Q-learning. Next, we show how to construct the state and action spaces for the task at hand, and how to reduce the learning time by the learning from easy missions (or LEM) mechanism. Finally, we show the experimental results by the computer simulations and the real robot system, and give a discussion and concluding remarks.

2 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 a goal as shown in Figure 1(a). The problem we attack 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 the mobile robot has a single TV camera, and that the robot does not know the location and the size of the goal, the size and the weight of the ball, any camera parameters such as focal length and tilt angle, or kinematics/dynamics of itself. Figure 1(b) shows a picture of the real robot with a TV camera (Sony handy-cam TR-3) used in the 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 [7]. We follow the explanation of the Q-learning by Kaelbling [8].

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 [9]. 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 Construction of State and Action Sets

4.1 Construction of Each Space

(a) a state set S

The image supposed to capture the ball and/or the goal is only the source of the information the robot can obtain about the environment. The ball image is classified into 9 sub-states, combinations of three sorts of positions (left, center, or right) and three sorts of sizes (large (near), middle, or small (far)). The goal image has 27 sub-states, combinations of three properties each of which is classified into three categories (see Figure 2). 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 $\boldsymbol{S}$.

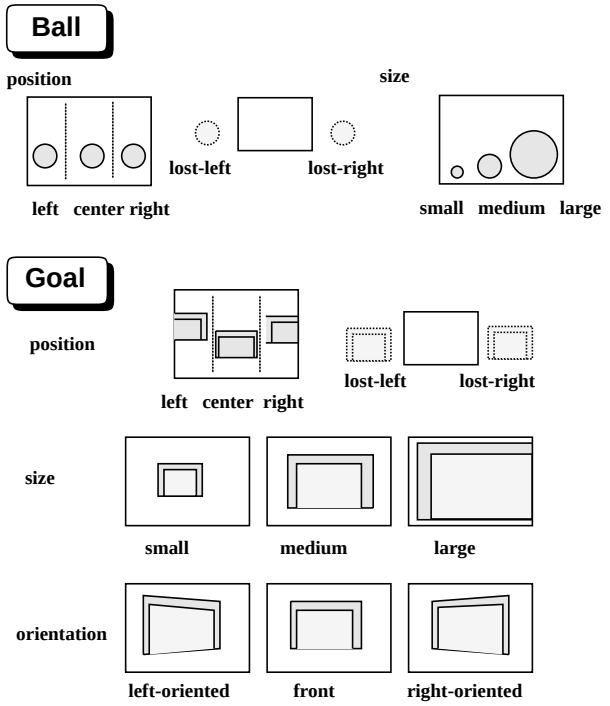


Figure 2: The ball substates and the goal substates

After some simulations, we realized that as long as the robot captures the ball and the goal positions in the image it succeeds in shooting a ball. However, once it lost the ball, the robot randomly moves because it does not know to which direction it should move to find the ball. This occurs because the ball-lost state is just one, therefore the robot 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. Similarly, 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 construct the action set in terms of two motor commands ω_l and ω_r, each of which has 3 subactions: 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 kicked 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 ($\gamma = 0.8$).

4.2 Solving A State-Action Deviation Problem

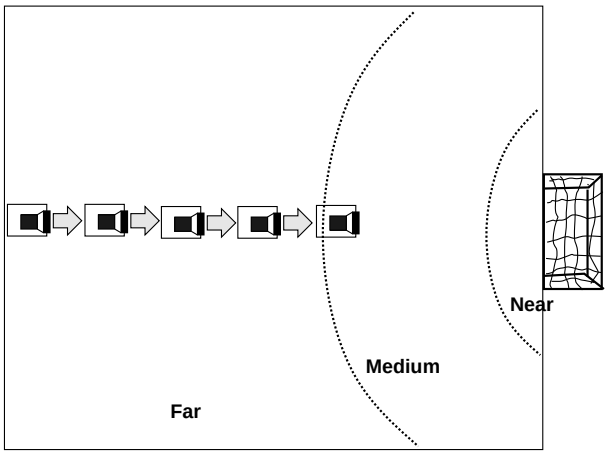


Figure 3: A state-action deviation problem

In the previous section, we constructed the state space in such a way that the position and the size of the ball or goal are naturally and coarsely classified into each state. Due to the peculiarity of the visual information, that is, a small change near the observer results in a large change in image and vice versa, one action does not always corresponds to one state transition. We call this **"state-action deviation problem."** Figure 3 indicates this problem, where the area of which state is "the goal is far" has a large area, and

therefore the robot frequently returns 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 converge correctly.

Then, we reconstruct the action space as follows. Each action defined in **4.1** 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 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 medium." The number of action primitives needed for state changes has no meanings. Once the state has changed, we update the action value function by eqn.(3).

5 Learning from Easy Missions

Unlike the approach in [5], we do not decompose the whole task into subtasks of finding, dribbling, and shooting a ball. Instead, we first used a monolithic approach. That is, we place the ball and the robot at arbitrary positions. In almost all the cases, the robot crossed over the field line without shooting the ball into the goal. This means that the learning did not converge 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 a goal, but he or she cannot imagine in which direction and how far the goal is because a reward is received only after the ball was kicked into the goal. Further, he or she does not know which action to be selected. 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 the order of the situations to achieve the goal.

Figure 4: The simplest state space.

5.1 Complexity analysis

We roughly estimate the time complexity for LEM following the complexity analysis by Whitehead [10]. We assume the "homogeneous" state space uniformly k-bounded with polynomial width of the depth k and zero-initialized Q-learning with a problem solving task. The problem solving task is defined as any learning task where the system receives a reward only upon entering a goal state. Further, we assume that state transition is deterministic and the robot can take one of 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 value of the action value function in each state converges if it is updated from the initial value (0).

Figure 4 shows an example of such state space. Since we assume the problem solving task, the unbiased Q-learning takes long time. From the above formulation, it needs m trials [2] 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 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 needs $(m^k + m^{k-1} + \cdots + m)$ trials to converge the values 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 [10].

While, in the Learning from Easy Missions algorithm, we place 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 place 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 the time 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 with a problem solving task, we roughly collect the states from which the agent can achieve the goal with high probability. We call the set of these states $\mathbf{S_1}$, a set of slightly more difficult states $\mathbf{S_2}$, and then $\mathbf{S_3}, \cdots$, respectively. Shifting to the next more difficult state set is occurs when

$$\Delta Q_t(\mathbf{S_k}, a) < \epsilon, \tag{4}$$

where

$$\Delta Q_t(\mathbf{S_k}, a) = \sum_{s \in \mathbf{S_k}} \left| \max_{a \in \mathbf{A}} Q_t(s, a) - \max_{a \in \mathbf{A}} Q_{t-\Delta t}(s, a) \right|$$

$$(k = 1, 2, 3, \cdots).$$

Δt indicates a time interval for a number of steps of state changes.

The closer to zero ϵ in eqn. (4) is, the earlier Q converges to Q^* in a deterministic world because it takes longer time to converge Q-values in the state set $\mathbf{S_k}$ if we shift the initial states to the state set $\mathbf{S_{k-1}}$ earlier due to $\epsilon \gg 0$. It is, however, not always true in a non-deterministic world that $\Delta Q_t(\mathbf{S_k}, a) = 0$ when the learning is at its final stage. Therefore, we have to set up an adequate value for ϵ.

[2] Here, we define one trial as one step state transition by one action.

We suppose that the result of the stochastic state transition of the current state set $\mathbf{S_{k-1}}$ cannot be every state but limited to the state in the neighbor state sets $\mathbf{S_{k-2}}$ and $\mathbf{S_k}$. If the learning sufficiently converged, we have the following from eqn. (3).

$$\Delta Q(s,a) = \left| \alpha(r(s,a) + \gamma \max_{a' \in \mathbf{A}} Q(s',a') - Q(s,a)) \right|.$$

Since $r(s,a) = 0$ except the goal state,

$$\Delta Q(s,a) = \begin{cases} \alpha\frac{1-\gamma}{\gamma} Q(s,a), & \text{for } s' \in \mathbf{S_{k-2}} \\ 0, & \text{for } s' \in \mathbf{S_{k-1}} \\ \alpha(1-\gamma)Q(s,a). & \text{for } s' \in \mathbf{S_k} \end{cases}$$

If $Q(s,a)$ has converged,

$$\Delta Q(s,a) \leq \alpha\frac{1-\gamma}{\gamma} Q(s,a).$$

Taking a summation in the state set $\mathbf{S_k}$ leads:

$$\Delta Q(\mathbf{S_k},a) \leq \alpha\frac{1-\gamma}{\gamma} \sum_{s \in \mathbf{S_k}} \max_{a \in \mathbf{A}} Q(s,a).$$

Note that the sampling time Δt should be long to exactly check the convergence without being affected by small fluctuations of the summation of Q-values, but also short to reduce the total learning time. Since the above inequality supposes that the Q-values in all $s \in \mathbf{S_k}$ are updated, an adequate Δt should be longer than $|\mathbf{S_k}|$ steps which correspond to the necessary steps to make trials initiating from every $s \in \mathbf{S_k}$. From the above discussions, we adopt:

$$\epsilon = \alpha\frac{1-\gamma}{\gamma} \sum_{s \in \mathbf{S_k}} \max_{a \in \mathbf{A}} Q(s,a). \quad (5)$$

5.2 Simulation results

As a simple example, we consider the grid world problem. In this example, the robot is free to roam about a bounded 2-dimensional grid, $n \times n$. It can move one of four principle directions, left, right, up, or down. The robot is reset after it reached the goal or crossed over the grid world boundary. In order to simulate the stochastic state transitions, the robot can move to the desired grid with a 3/4 probability but stays at the current grid with a 1/4 probability. The goal is set at the top right corner and the state is encoded as a coordinate of the grid. We assign a reward value 1 when the robot achieves the goal or 0 otherwise. The depth of the state space, the maximum distance to the goal of the optimal paths, is $2(n-1)$.

We implemented LEM as follows. The state set $\mathbf{S_1}$ consists of two states one grid neighbor to the goal, $\mathbf{S_2}$ three states one grid neighbor to $\mathbf{S_1}$, and so on. We shift the initial grids of $\mathbf{S_k}$ to that of $\mathbf{S_{k+1}}$ according to eqns. (4) and (5) with $\Delta Q(\mathbf{S_j},a) = |\mathbf{S_j}|$.

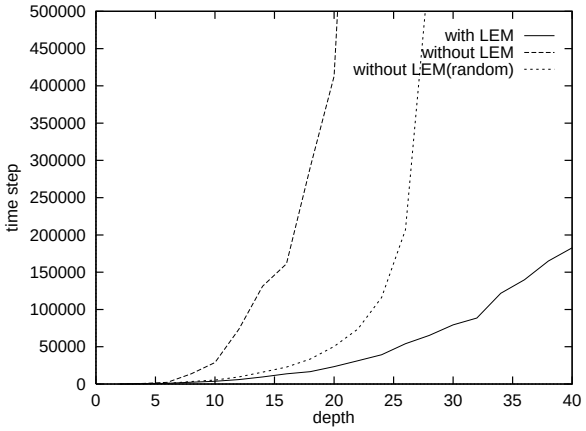


Figure 5: Search time complexity in terms of the size of state space.

Figure 5 shows a plot of the search time (number of steps) versus maximum distance k, where one step Q-learning algorithm is applied with the learning rate $\alpha = 0.25$ and the discounting factor $\gamma = 0.9$. As we expected, the search time 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 at the bottom left corner (broken line) or randomly placed (dotted line). While, the search time with LEM (solid line) is almost linear in the size of k although the curve is a little bit upward because some inadequate trials wasted much time due to useless search such as backward motions.

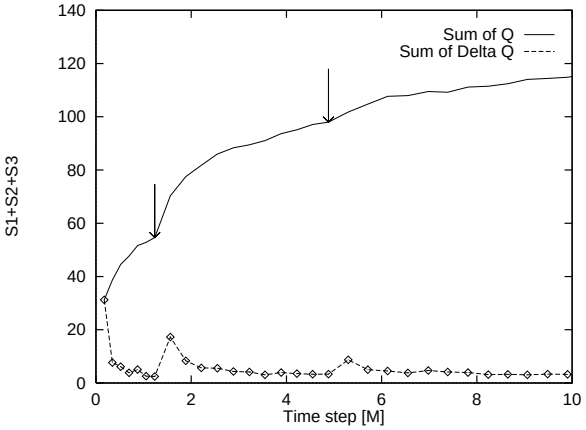


Figure 6: Change of the sum of Q and ΔQ-values in $\mathbf{S_1} + \mathbf{S_2} + \mathbf{S_3}$

6 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.

6.1 Computer simulation

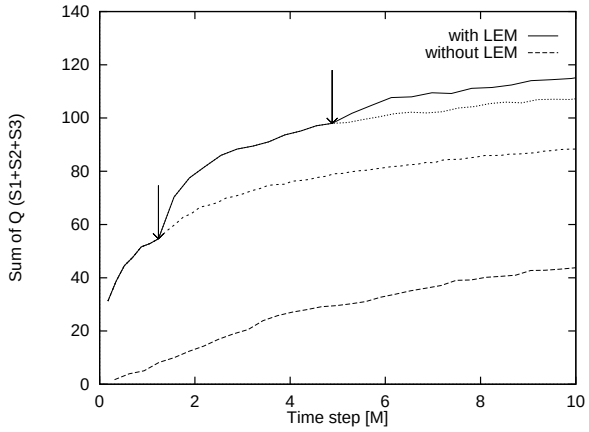


Figure 7: Changes of the sum of Q-values with and without LEM

We performed the computer simulation with the following specifications. The field is a square of 3.0[m] × 3.0[m]. The goal post is located at the center of the top line of the square (see Figure 1) and its height and width are 0.23[m] and 0.9[m], respectively. The robot is 0.31[m] wide and 0.45[m] long and kicks a ball of diameter 0.09[m]. The maximum translation velocity is 1.1[m/s], and the maximum angular velocity is 4.8[rad/s]. The camera is horizontally mounted on the robot (no tilt), and its visual angle is 36 degrees. The velocities of the ball before and after being kicked by the robot is calculated by assuming that the mass of the ball is negligible compared to that of the robot. The ball speed is temporally decreased by a factor 0.8 in order to reflect the so-called "viscous friction." The values of these parameters are determined so that they can roughly simulate the real world.

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. ϵ is determined by eqns. (4) and (5). Since $|\mathbf{S_1}| = |\mathbf{S_2}| = |\mathbf{S_3}| = 81$, we fixed $\Delta Q(\mathbf{S_j}, a) = 3000$ that is much larger than $|\mathbf{S_j}|$.

Figure 6 shows the changes of Q and ΔQ where the solid and broken lines indicate the maximum value of Q and the summation of ΔQ in states $\in \mathbf{S_1}+\mathbf{S_2}+\mathbf{S_3}$, respectively. The axis of time step is scaled by M (10^6) that corresponds to about 9 hours in real world since one time step is about 30[ms]. Two arrows indicate the time steps (around 1.4M and 5M) when a set of the 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.

Figure 7 shows the changes of the sum of Q-values with (solid line) and without (broken line) LEM. The Q-learning with LEM is much better than that without LEM. The fine and coarsely dotted lines show the curves when the initial positions were not changed from $\mathbf{S_1}$ to $\mathbf{S_2}$, and from $\mathbf{S_2}$ to $\mathbf{S_3}$, respectively. This simulates the LEM with the partial knowledge. If we know only the easy situation of $\mathbf{S_1}$, and nothing anymore, the learning curve follows the coarse dotted line in Figure 7. The sum of Q values is slightly less than that of the LEM with more knowledge, but much better than without LEM.

6.2 Experimental results on a real robot

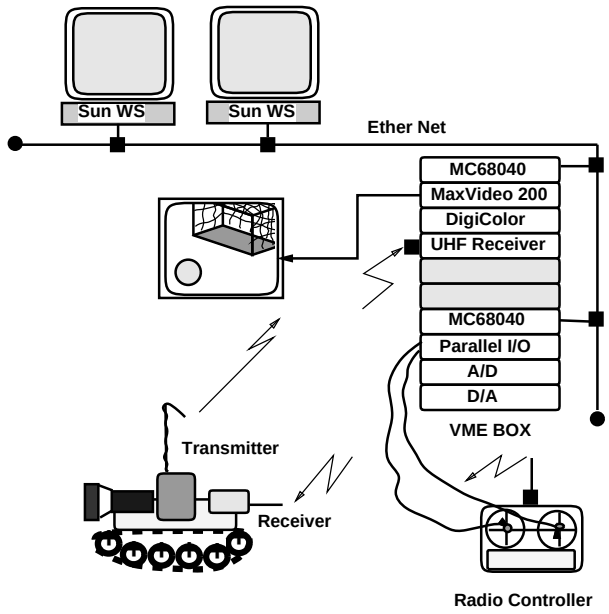
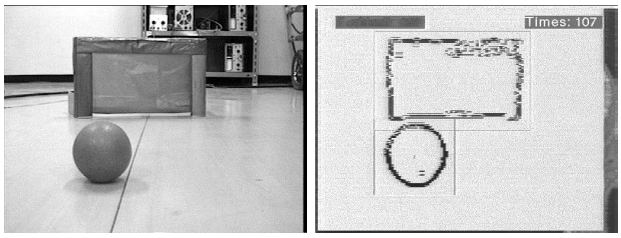


Figure 8: A configuration of the real system.



(a) input image (b) detected ball and goal

Figure 9: Result of image processing.

Figure 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. We constructed

the radio control system of the vehicle [11]. The image processing and the vehicle control system are operated by VxWorks OS on MC68040 CPUs which are connected with host Sun workstations via Ether net. We have shown a picture of the real robot in Figure 1(b).

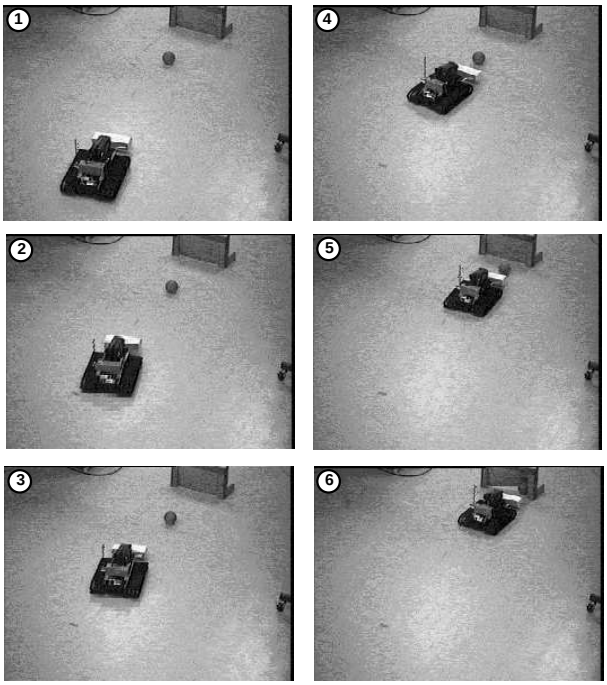


Figure 10: The robot succeeded in shooting a ball into the goal.

In order to simplify and speed up the image processing time, we painted the ball in red and the goal in blue. The input NTSC color video signal is first converted into HSV color components in order to make the extraction of the ball and the goal easy. Then, the image size is reduced to speed up the image processing time more, and the boundaries of the ball and goal regions are extracted for the state discrimination. The result of image processing is sent to the host CPU to decide an optimal action against the current state. Figure 9 (a) and (b) show an example of input image captured by the robot and the result of the ball and the goal detection for the input image, respectively. Their positions and orientation are calculated in real time (totally, about 30ms).

The shooting rate in the real robot system was less than 50% which was about 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.

Figure 10 shows a sequence of the images in which the robot succeeded in shooting a ball into the goal by the method. Table 1 shows the image processing and state mapping result in each time step (about 30ms) for the sequence of the images captured by the robot, examples of which are shown in Figure 10. Each column indicates time step, the state transition step, the mapped state, the action command, and the number of errors in the state discrimination process, respectively. The mapped state consists of five substates: two for the 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-**o**riented, 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 the error box. An action command consists of a combination of two independent motor commands (**F**orward, **S**top, or **B**ackward).

Table 1: State-Action data

time step	state step	state		action		error
		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	

Amazingly, the ratio of the completely correct mappings is about 60%. Most of the incorrect mappings occur when the size of the ball is misjudged as smaller one due to failures in edge detection. As long as the ball and the goal are captured at the center of the image, this does not cause a serious situation 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 misjudges the distance

to the ball. Due to the noise of the transmitter, completely incorrect mappings occur at the ratio of 15%. Unless this situation continues two or more time steps, this does not cause a serious situation because the robot can take the correct action once the correct state mapping is obtained.

7 Discussion

Coarse segmentation of the state space ("right" or "left" and "small" or "large") is useful in two folds: one is to reduce the size of the state space and the other one to absorb the image processing noise. Ideally, the agent should find these properties necessary for goal achievement automatically from the visual information. Although some works are reported on this problem [12, 13], we need much more works to make clear what kind of information is necessary to achieve the goal and how to construct (segment) the state space from it.

The LEM algorithm differs in some aspects from the existing approaches to speed up the search time. In the task decomposition approach [5], the Q-learning is closed inside each subtask. In LEM, however, the robot wanders around the field crossing over the states easy to achieve the goal even if we initially place it at such states. We just advise the positions of the easy states. 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, and the partial knowledge about the easiness of the missions can be used in LEM scheme. However such knowledge seems difficult to be applied in the task decomposition scheme.

In the Learning with an External Critic (or LEC) [10], the robot receives an advise in each state from the external critic. In order to let LEC work correctly, the complete knowledge about the evaluation for the action taken in any state 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.

8 Concluding Remarks

We have shown a vision-based reinforcement learning method which is, to the best of our knowledge, the first attempt at applying reinforcement learning (Q-learning) to a real robot task with real time vision system. We adopted the Learning from Easy Missions algorithm similar to a "shaping" technique in animal learning in order to speed up the learning time 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 Q-learning to real robot tasks, and we constructed an action space to cope with this problem.

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 learn with the policy obtained by computer simulation as the initial values of the action-value function to be determined.

References

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

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

[3] F. Saito and T. Fukuda. "Learning architecture for real robot systems – extension of connectionist q-learning for continuous robot control domain". In *Proc. of 1994 IEEE Int. Conf. on Robotics and Automation*, pages 27–32, 1994.

[4] A. H. Fagg, D. Lotspeich, and G. A. Bekey. "A reinforcement learning approach to reactive control policy design for autonomous robots". In *Proc. of 1994 IEEE Int. Conf. on Robotics and Automation*, pages 39–44, 1994.

[5] 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.

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

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

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

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

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

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

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

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