Machine Learning, 23, 279–303 (1996)

Purposive Behavior Acquisition for a Real Robot by Vision-Based Reinforcement Learning

MINORU ASADA, SHOICHI NODA, SUKOYA TAWARATSUMIDA, AND KOH HOSODA

asada@robotics.ccm.eng.osaka-u.ac.jp

Dept. of Mech. Eng. for Computer-Controlled Machinery
Osaka University, 2-1, Yamadaoka, Suita, Osaka 565, Japan

Abstract. This paper presents a method of vision-based reinforcement learning by which a robot learns to shoot a ball into a goal. We discuss several issues in applying the reinforcement learning method to a real robot with vision sensor by which the robot can obtain information about the changes in an environment. First, we construct a state space in terms of size, position, and orientation of a ball and a goal in an image, and an action space is designed in terms of the action commands to be sent to the left and right motors of a mobile robot. This causes a "state-action deviation" problem in constructing the state and action spaces that reflect the outputs from physical sensors and actuators, respectively. To deal with this issue, an action set is constructed in 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 from Easy Missions* (or LEM) is implemented. LEM reduces the learning time from exponential to almost linear order in the size of the state space. The results of computer simulations and real robot experiments are given.

Keywords: reinforcement learning, vision, learning from easy mission, state-action deviation

1. Introduction

Realization of autonomous agents that organize their own internal structure in order to take actions towards achieving their goals 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 method, a robot and its 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 reinforcement learning is very important to realize autonomous systems, the prominence of that role is largely dependent on the extent to which the learning can be scaled to solve larger and more complex robot learning tasks. Many researchers in the field of machine learning have been concerned with the convergence time of the learning, and have developed methods to speed it up. They have also extended these techniques from solving single goal tasks to multiple goal ones [2]. However, almost all of them have only shown computer simulations in which they assume ideal sensors and actuators, where they can easily construct the state and action spaces consistent with each other. A typical example is the

2-D grid environment in which the robot can take an action of going forward, backward, left, or right, and its state is encoded by the coordinate of the grid (i.e., an absolute (global) positioning system is assumed). Although the uncertainties of sensor and actuator outputs are considered by a stochastic transition model in the state space, such a model cannot account for the accumulation of sensor errors in estimating the robot position. Further, from the 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 a great deal of knowledge about the environment and itself. We intend to apply the reinforcement learning algorithm to the task of purposive behavior acquisition in the real world with less knowledge about the environment and the robot.

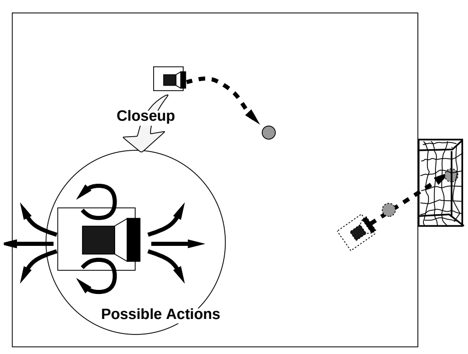
Mahadevan and Connel [5] proposed a method of rapid task learning on a real robot. They separated a pushing task into three subtasks of "finding a box", "pushing a box", and "getting unwedged", and applied Q-learning, a widely used reinforcement learning method, to each of them. Since only proximity 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 more global and goal-directed tasks such as carrying a box to a specified location. For such tasks, visual sensors could be more useful because they might be able to capture the image of the goal in a distant place. However, there are very few examples of use of visual information in reinforcement learning [1], probably because of the cost of visual processing.

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 the only source of information on the changes in an environment. Image positions and sizes of the ball and the goal are used as a state vector. We discuss several 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) starting with easy missions (rather than task decomposition) for rapid task learning.

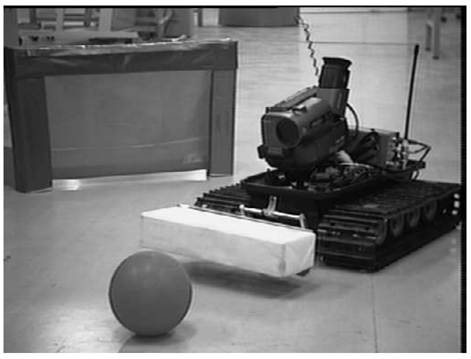
The remainder of this article is structured as follows: In the next section, we explain the task and assumptions, and give a brief overview of Q-learning. Next, we show how to construct the state and action spaces for the task at hand, and describe ways to reduce the learning time by Learning from Easy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

2. Task and Assumptions

The task for a mobile robot is to shoot a ball into a goal as shown in Figure 1(a). The problem we address here is how to develop a method which automatically acquires strategies for doing this. We assume that the environment consists of a ball and a goal; the mobile robot has a single TV camera; and that the robot does not know the location/size of the goal, the



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



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

Figure 1. Task and our real robot.

size/weight of the ball, any camera parameters such as the focal length and tilt angle, or the 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 will briefly review the basics of Q-learning. For a more thorough treatment, see [6]. We follow the explanation of Q-learning by Kaelbling [7].

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 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.

The general reinforcement learning problem is typically stated as finding a policy [2] that maximizes the discounted sum of rewards received over time. This sum is 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. The value of γ is usually slightly less than 1.

Given definitions of the transition probabilities and the reward distribution, we can solve for the optimal policy, using methods from dynamic programming [8]. 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 $Q(s,a)$ equal to an arbitrary value (usually 0), every time an action is taken, the Q-value is updated as follows:

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

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 learning rate (between 0 and 1).

4. Construction of State and Action Sets

Traditional notions of state in the existing applications of the reinforcement learning algorithms fit nicely into deterministic state transition models (e.g. one action is forward, backward, left, or right, and the states are encoded by the locations of the agent). However, this is not always the case in the real world, where everything changes asynchronously [9]. Thus, we need to have the following principles 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 world. 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 world, therefore it is difficult to construct the state and action spaces in which one action always corresponds to one state transition. We call this the **"state-action deviation problem"** as one of the so-called "perceptual aliasing problem" [10] (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. In this section, we first show how to construct the state and action spaces, and then how to cope with the state-action deviation problem.

4.1. Construction of Each Space

(a) a state set $\boldsymbol{S}$

The image, supposed to capture the ball and/or the goal, is the only source of information the robot can obtain about the environment. The ball image is classified into 9 sub-states, combinations of three classifications of positions (left, center, or right) and three types 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 towards 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 the cases in which only the ball or only the goal is captured in the image. In all, we have 319 states in the set $\boldsymbol{S}$.

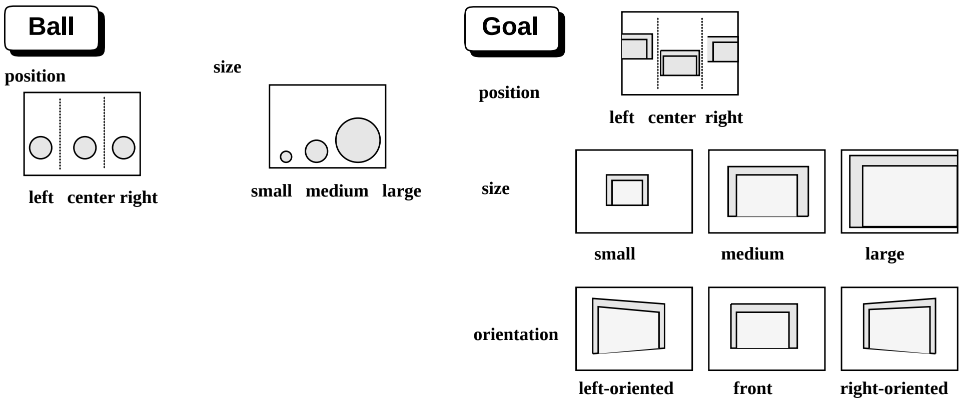


Figure 2. The ball sub-states and the goal sub-states

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 into the goal. However, once it cannot view the ball, the robot moves randomly because it does not know in which direction it should move to find the ball. This occurs because “ball-lost” is just one state, therefore the robot cannot discriminate between the various directions in which the ball may be lost. Thus, we separate the ball-lost state into two states; the ball-lost-into-right and ball-lost-into-left states. Similarly, we set up the goal-lost-into-right and goal-lost-into-left states. These additional classifications improved the robot behavior.

(b) an action set $\boldsymbol{A}$

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 to each of the two motors separately, we construct 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. All together, we have 9 actions in the action set $\boldsymbol{A}$.

(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. Solving A State-Action Deviation Problem

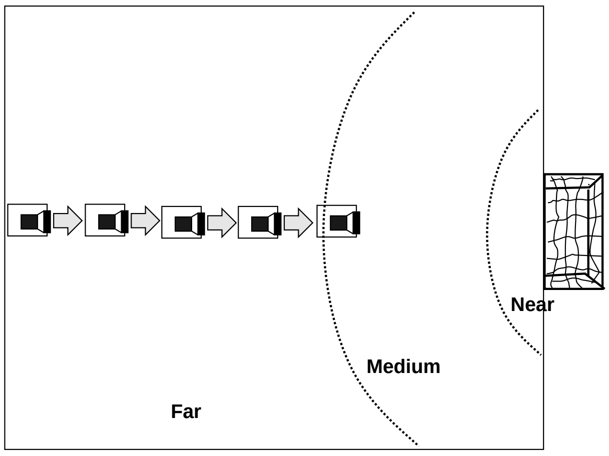


Figure 3. A state-action deviation problem

In the previous section, we constructed the state space so that the position and the size of the ball or goal are naturally and coarsely classified into each state. Due to the peculiarity of visual information, that is, a small change near the observer results in a large change in the image and a large change far from the observer may result in a small change in the image, one action does not always correspond to one state transition. We call this the **"state-action deviation problem"**: Figure 3 indicates this problem, the area representing the state "the goal is far" is large, therefore the robot frequently returns to this state if the action is forward. This is highly undesirable because the variations in the state transitions is very large, consequently the learning does not converge correctly.

To avoid this problem, 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 at a time until the current state changes. This sequence of the action primitives 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 medium". The number of action primitives needed for one state change is not used in the algorithm. Once the state has changed, we update the action-value function by Eqn.(3).

5. Learning from Easy Missions

In order to improve the learning rate, the whole task was separated into different parts in [5]. By contrast, we do not decompose the whole task into subtasks of finding, driblling, 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 (execution time of one week on SGI Elan with R4000). This situation resembles a case where a small child tries to shoot a ball into a goal, but (s)he cannot imagine in which direction and how far the goal is because a reward is received only after the ball is kicked into the goal. Further, (s)he does not know which action is 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. 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).

In the following, we first show the complexity analysis of LEM assuming that the agent knows the order of state transitions completely. Although it does not make any sense if we know it because the learning is no longer necessary, we intend to show the difference in learning times between with and without LEM. Then, we describe how to cope with two problems in implementing the LEM in real tasks: how to decide which state is easier to achieve the goal and when to shift from easy situations to more difficult ones.

5.1. Complexity analysis of LEM

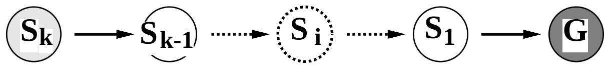


Figure 4. The simplest state space.

We roughly estimate the time complexity for LEM following the complexity analysis by Whitehead [11]. In order to show the difference in the learning time with and without LEM, we assume complete knowledge of the order of the states to the goal. However, as mentioned above, the LEM does not always need completeness of the knowledge.

We assume a "homogeneous" state space uniformly k-bounded with polynomial width of 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 of actions with equal probability. In order to figure out how many steps are needed for Q-learning to converge, we use $O(k)$ state space and simplify the convergence such that the action-value function in each state converges if it is updated from the initial value (0).

Figure 4 shows an example of such a state space. Since we consider a problem solving task, the unbiased Q-learning takes a long time. From the above formulation, it needs m trials [3] 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 action-value function for only the state S_1 is updated. Next, it needs m^{k-1} trials to update the action-value function for the state S_2, and in total it needs $(m^k + m^{k-1} + \cdots + m)$ trials for the action value function to converge for all the states. Therefore, the unbiased Q-learning can be expected to take execution time exponential in the size of k [11].

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

5.2. Implementing LEM

In actual situations like our task, the state transition is stochastic, and the state space is not homogeneous. 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. In the following, we show how to cope with this problem.

(a) Rough ordering of easy situations

Generally, it seems difficult to know which state is easier than others to achieve the goal, unless *a priori* knowledge on ease of achieving the goal is given. However, in real applications, we can assume that the sensor space is smooth and the goal state is known. Therefore, the agent roughly knows the order of state transitions such as the size of a goal or a ball in the image regardless of other sub-states in our task: "small" $\rightarrow$ "medium" $\rightarrow$ "large" $\rightarrow$ "goal state". Similarly, the position of the goal and ball are also ordered such as "left (or right)" $\rightarrow$ "center" $\rightarrow$ "goal state". However, the orientation of the goal is not ordered because the agent can achieve the goal from any direction.

This ordering is possible if the state space is categorized into sub-states such as ball size, ball position, and so on. Figure 5 shows a state space in which a number of states are ordered along the sub-state axis. The LEM paradigm can be applied when such a sub-state axis exists or partial ordering along the axis is known. Let n and m be the size of the state space (the number of states) and the number of ordered sets (e.g., 3, when the sub-state axis is the goal size and states are "small", "medium", and "larg e"). When the agent learns without LEM it takes $O(e^n)$. If we apply the LEM with m ordered state sets, it would take $O(m \times \sqrt[m]{e^n})$.

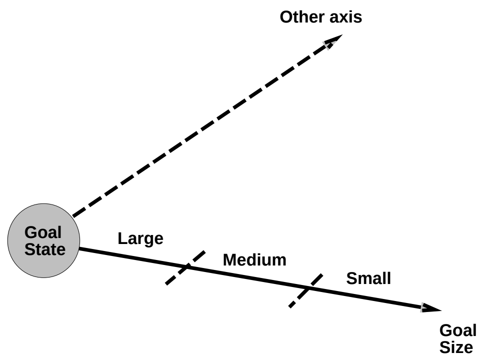


Figure 5. State space part of which divided into sub-states

(b) When to shift?

Another problem in implementing the LEM paradigm is to decide when to shift the initial situations into more difficult ones. From the above discussion, we suppose that the agent knows the ordering of situations along one sub-state axis according to the closeness of the goal state. We call a set of states $\mathbf{S_1}$ that is closest to the goal along the current sub-state axis regardless of other sub-state axes, the set of second closest states $\mathbf{S_2}$, the set of third closest states $\mathbf{S_3}$, and so on. Shifting to the next set occurs when

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

where

$$\Delta Q_t(\mathbf{S_k}, a) = \sum_{s \in \mathbf{S_k}} \left| \max_{a \in \mathbf{A}} Q_t(s,a) - \max_{a \in \mathbf{A}} Q_{t-\Delta t}(s,a) \right| \qquad (k = 1, 2, 3, \cdots). \tag{5}$$

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

The closer to zero ϵ is in inequality (4), 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 than the case where $\epsilon \approx 0$. 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 appropriate value for ϵ.

We suppose that the current state set $\mathbf{S_{k-1}}$ cannot transit to any state but is limited to transitions to states in the neighborhood of sets $\mathbf{S_{k-2}}$ and $\mathbf{S_k}$. When the learning has converged sufficiently, we have the following from Eqn. (3).

$$\Delta Q(s,a) = \left| \alpha(r(s,a) + \gamma \max_{a' \in \mathbf{A}} Q(s',a') - Q(s,a)) \right|. \tag{6}$$

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

$$\Delta Q(s,a) = \begin{cases} \alpha \frac{1-\gamma}{\gamma} Q(s,a), & \text{for } s' \in \mathbf{S_{k-2}} \\ 0, & \text{for } s' \in \mathbf{S_{k-1}} \\ \alpha(1-\gamma)Q(s,a). & \text{for } s' \in \mathbf{S_k} \end{cases} \tag{7}$$

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

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

Taking summation over elements in the state set $\mathbf{S_k}$ leads to:

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

Note that the sampling time Δt should be long enough to check the convergence exactly without being affected by small fluctuations of the summation of Q-values, but also short enough to reduce the total learning time. Since the above inequality supposes that the Q-values for all $s \in \mathbf{S_k}$ are updated, an adequate Δt should be longer than $|\mathbf{S_k}|$ attempts [4] to make attempts starting from every $s \in \mathbf{S_k}$. Following the above discussions, we adopt:

$$\epsilon = \alpha \frac{1-\gamma}{\gamma} \sum_{s \in \mathbf{S_k}} \max_{a \in \mathbf{A}} Q(s,a). \tag{8}$$

From the above, the LEM algorithm is formulated as follows:

1. Find a sub-state axis along which rough ordering of state transition to a goal state is known.
2. Let $\mathbf{S_i}$ $(i = 1, 2, .., m)$ be a set of states that is closest to the goal along the current sub-state axis regardless of other sub-state axes, a set of second closest states, a set of third closest states, and so on.
3. Set $i = 1$.
4. Repeat many attempts starting with various kinds of initial situations which are classified into the state set $\mathbf{S_i}$ until the inequality (4) is satisfied (Action selection is random).
5. If $i = m$, then terminate. Else, $i = i + 1$ and go to step 4.

5.3. Simulation results

To start with 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 to one of four principle directions, left, right, up, or down. The robot is reset after it reaches the goal or crosses over the grid world boundary. In order to simulate the stochastic state transitions,

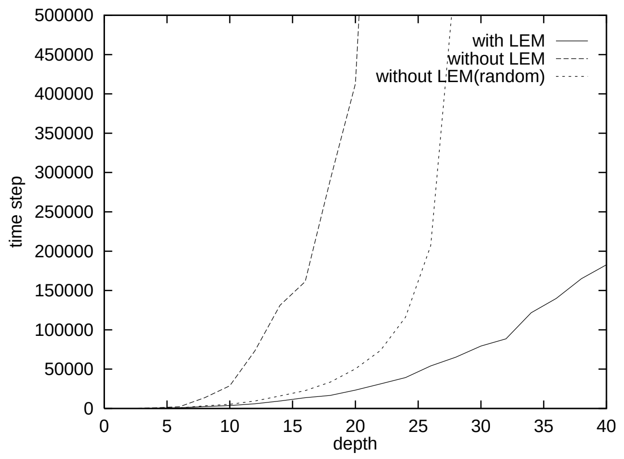


Figure 6. Search time complexity in terms of the size of state space.

the robot can move to a desired grid with probability 3/4 but stays at the current grid with probability 1/4. 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 and 0 otherwise. The depth of the state space, the maximum distance to the goal of the optimal path, is $2(n-1)$.

We implemented the LEM as follows. The state set $\mathbf{S_1}$ consists of two states (each of which is a grid neighbor to the goal), $\mathbf{S_2}$ consists of three states (each of which is a 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 (8) with $\Delta t = |\mathbf{S_j}|$ attempts.

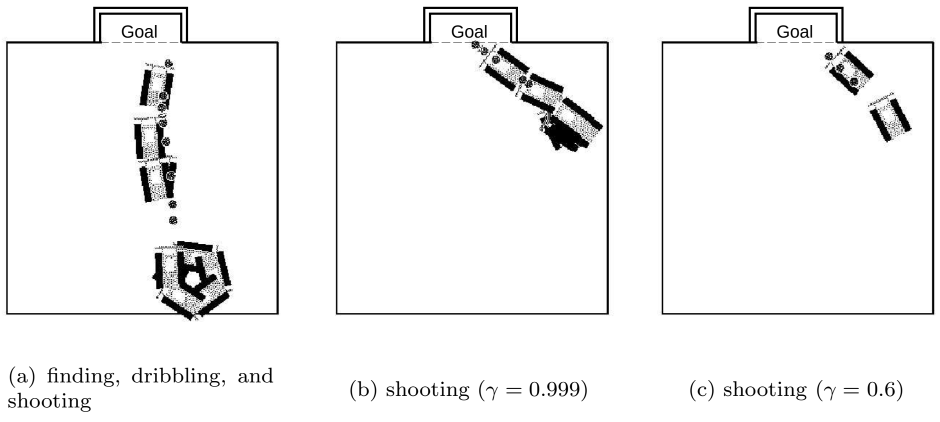
Figure 6 shows a plot of the search time (number of steps) versus maximum distance k, where the one step Q-learning algorithm is applied with the learning rate $\alpha = 0.25$ and the discounting factor $\gamma = 0.9$. As we expected, the search time of the normal Q-learning without LEM (dotted and broken 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). In comparison with Q-learning without LEM, 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 attempts wasted much time due to useless search such as backward motions.

6. Experiments

The experiment consists of two parts: first, learning the optimal policy f through the computer simulation, then applying the learned policy to a real situation.

6.1. Computer simulation

We performed the computer simulation with the following specifications. The field is a square of 3.0m $\times$ 3.0m. The goal post is located at the center of the top line of the square (see Figure 1) and its height and width are 0.23m and 0.9m, respectively. The robot is 0.31m wide and 0.45m long and kicks a ball of diameter 0.09m. The maximum translation velocity is 1.1m/s, and the maximum angular velocity is 4.8rad/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 speed of the ball 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.



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

Figure 7. Some kinds of behaviors obtained by our method

Figure 7 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, driblled 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. ϵ is determined by Eqns. (4) and (8). In order to ensure the convergence, we fix $\Delta t = 3000$ that is much larger than $|\mathbf{S_j}|$.

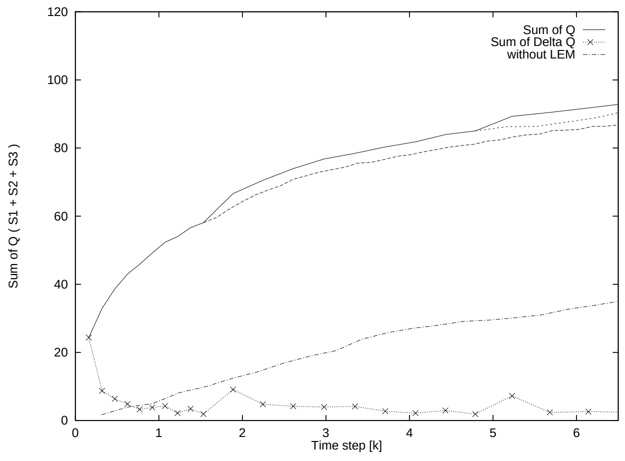


Figure 8. Change of the sum of Q-values with LEM in terms of goal size

Figure 8 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 world 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 empty squares 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 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 8. The summation of Q-values is slightly less than that of the LEM with more knowledge, but much better than that without LEM.

Similarly, we applied the LEM algorithm with other orderings in terms of ball position. Figure 9 show a graph similar to Figure 8. Since the ball position is classified into two categories such as “center” and “left (or right)”, the number of arrows indicating the time step to shift from a set of easier states to a set of more difficult ones is just one. Although the difference of LEM with partial and complete knowledge on the ordering along this sub-axis is smaller than the case of the goal size, the usefulness of the LEM algorithm can be seen.

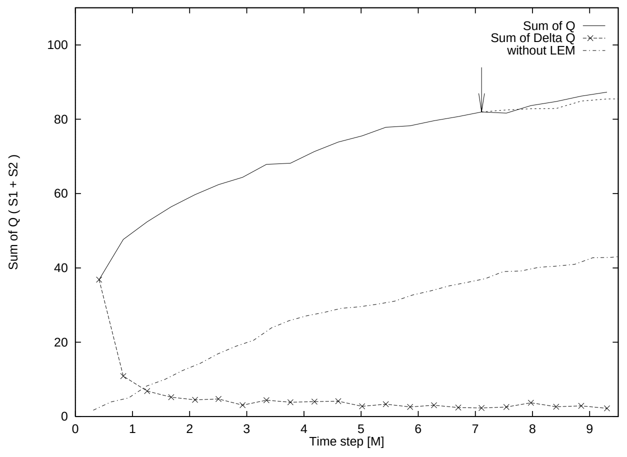


Figure 9. Change of the sum of Q-values with LEM in terms of ball position

6.2. Experimental results on a real robot

Figure 10 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 [12]. 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).

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 further speed up the image processing time, and the boundaries of the ball and goal regions are extracted for the state discrimination. The results from the image processing operations are sent to the host CPU which decides an optimal action for the current state. Figure 11 shows an example of the input image captured by the robot and the result of the ball and the goal detection for the input image, respectively. Their positions, sizes, and the goal orientation are calculated in real time (in total, 33ms). The image processing program can handle the occlusion by using knowledge of the object shapes such as a sphere and a rectangle.

We tested the learned policy on the real robot almost 100 times. The shooting rate in the real robot system was about 60% which was about 20% worse than the simulation. The main reason for this is that the ball often moves towards unpredictable directions due to the eccentricity of its centroid. The second cause for poorer performance is noise in the image processing results, explained in the following paragraph.

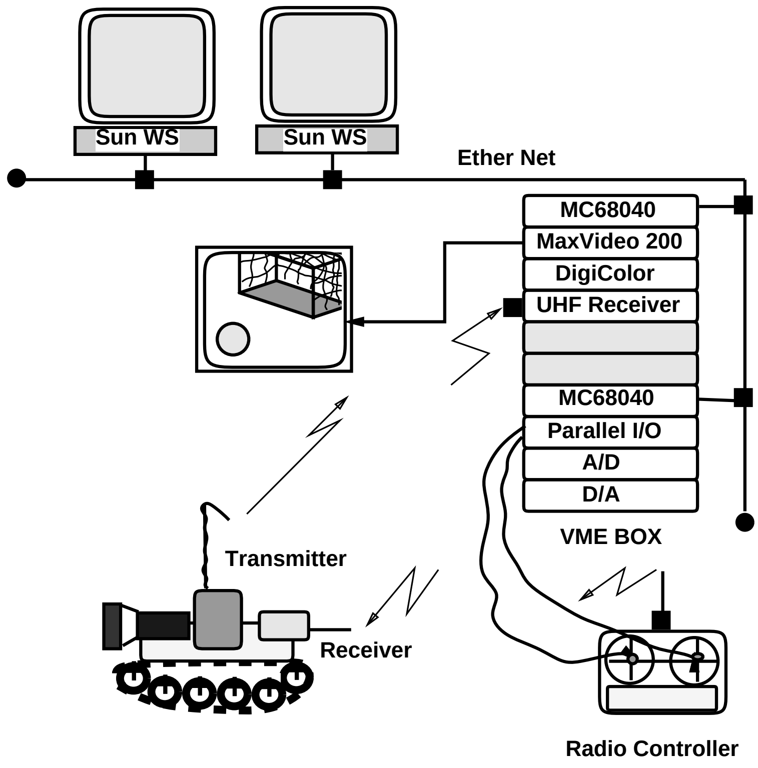


Figure 10. A configuration of the real system.

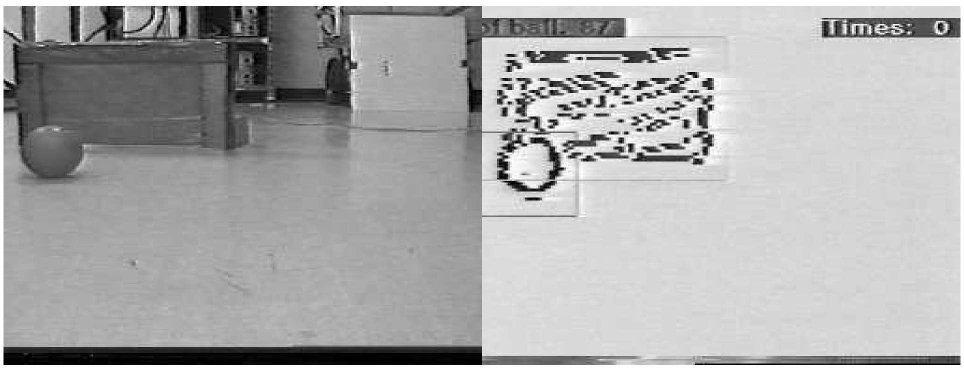


Figure 11. Result of image processing.

Figure 12 shows a sequence of the images (raster order from top left to bottom right) in which the robot succeeded in shooting a ball into the goal by using our method. The images

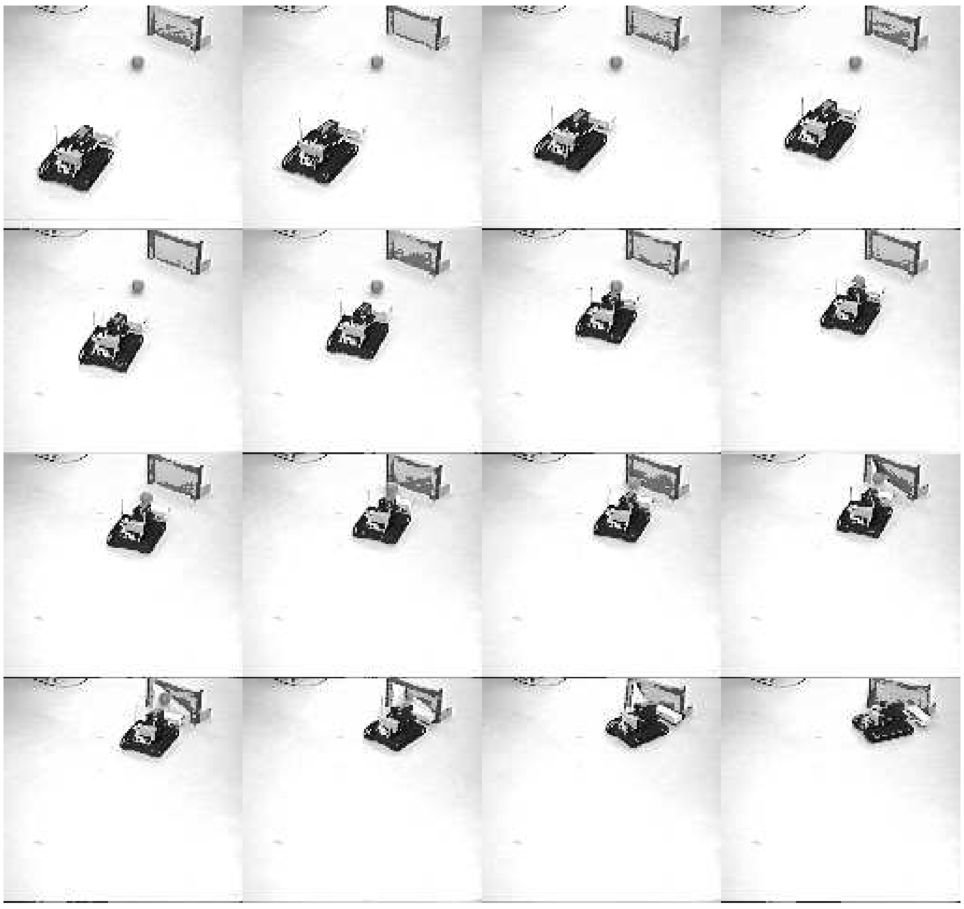


Figure 12. The robot succeeded in shooting a ball into the goal(I).

are taken every five frames (166ms). Table 1 shows the image processing and state mapping result in each time step (33ms) for the sequence of images captured by the robot, examples of which are shown in Figure 12. Each column indicates the 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 sub-states: 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 sub-states are marked with "*"s, and the number of these sub-states 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).

Amazingly, the ratio of completely correct mappings is about 60%. Most of the incorrect mappings occur when the size of the ball is misjudged as smaller due to failures in edge

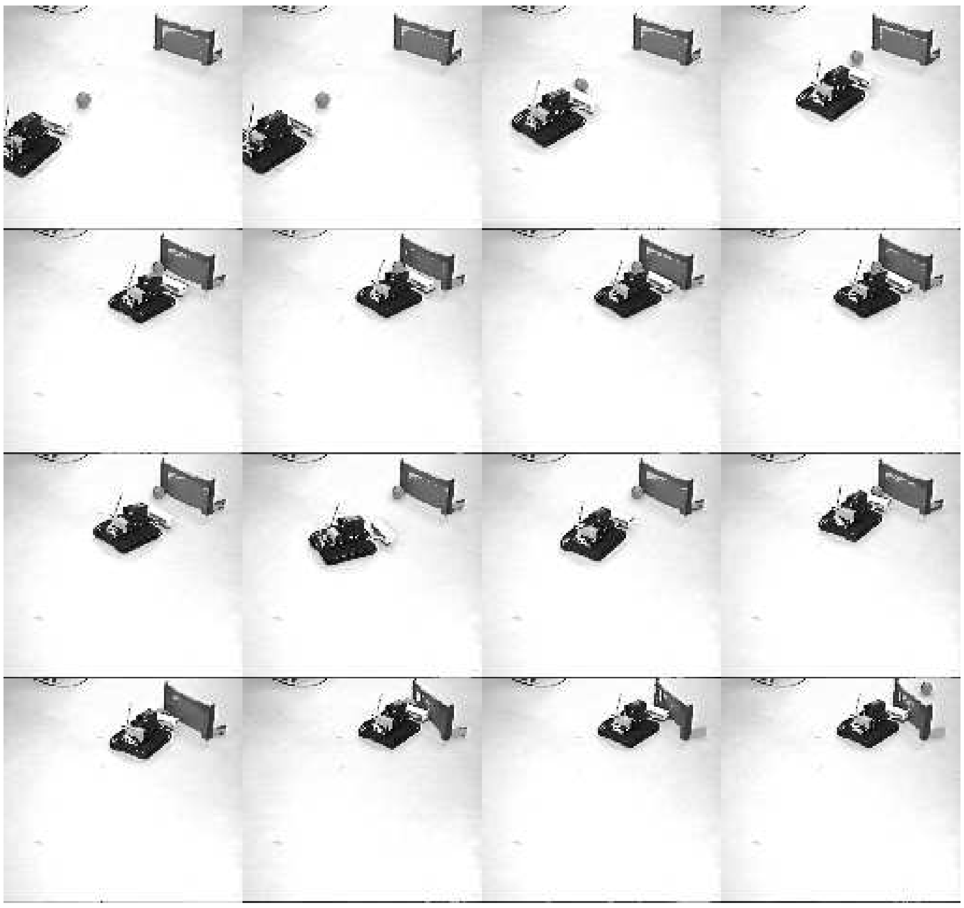


Figure 13. The robot failed to shoot a ball into the goal.

detection. As long as the ball and the goal are captured at the center of the image, this does not cause a serious problem because the optimal action is forward in any case. However, the robot 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 problem because the robot can take the correct action once the proper state mapping is obtained.

Other examples are shown in Figures 13 and 14. In the former, the robot failed to shoot a ball. The images are taken every 12 frames (about 400ms). First, the robot tried to shoot a ball into a goal, but the ball moved in an unpredicted direction. Then, the robot moved backward, and tried again. Still it could not shoot the ball into the goal, but it seems that the robot had an intention to do that even though it failed. Note that, the shooting is not

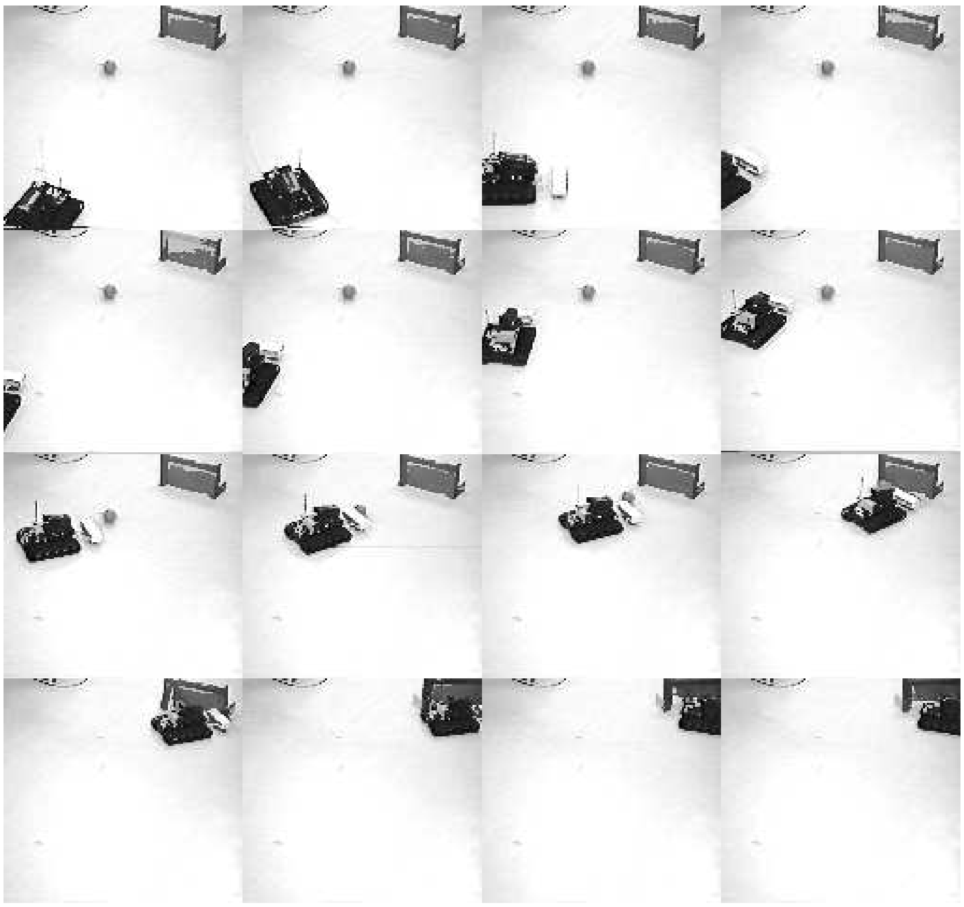


Figure 14. The robot succeeded in shooting a ball into the goal(II).

a programmed action but just a result of learning. In Figure 14, the images are taken every second. First, the robot lost the ball due to noise, and then it turned around to find the ball, and finally it succeeded in shooting.

7. Discussion

We discuss several issues related to applying the reinforcement learning to real robot tasks.

Table 1. State-Action data

time	state	state		action		# of
step	step	ball	goal	L	R	erros
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	

(a) comparison with conventional approaches

In order to compare the performance of our method with conventional approaches, we first implemented a servo mechanism with a controller for angular and forward motion velocities (ω, v). The following equation shows the relationship between (v, ω) and two angular velocities (ω_r, ω_l) to be sent to the right and left motors.

$$\begin{pmatrix} v \\ \omega \end{pmatrix} = \begin{pmatrix} \frac{R_r}{2} & \frac{R_l}{2} \\ \frac{R_r}{T} & -\frac{R_l}{T} \end{pmatrix} \begin{pmatrix} \omega_r \\ \omega_l \end{pmatrix} \tag{9}$$

where R_r, R_l, and T denote the radii of the right and left wheels, and the distance between two wheels, respectively.

We first implemented the following proportional controller which tries to reduce the differences in visual angle between the lines of sight to the ball center (x_b) and to the goal center (x_g), and in distance between the robot and the ball (l):

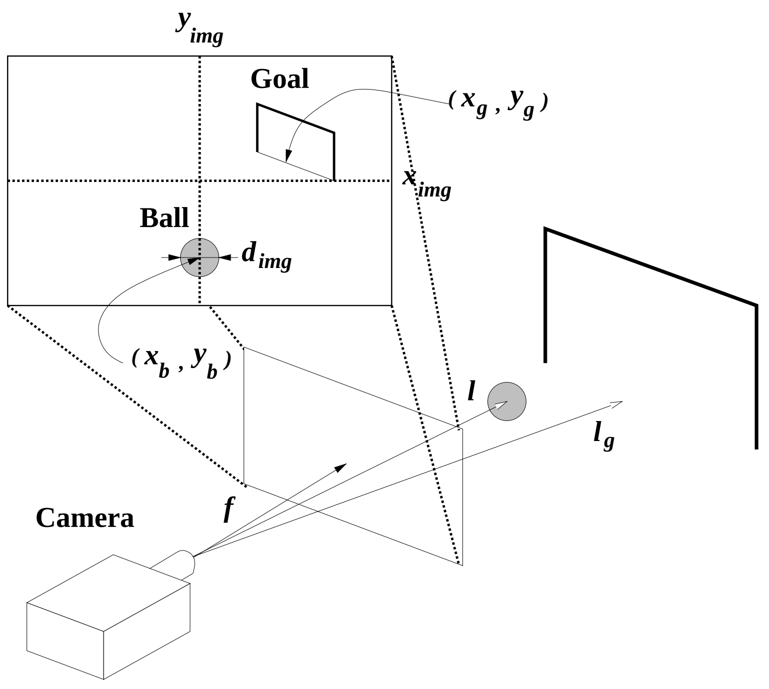


Figure 15. Relationship between camera, ball, and goal

$$\omega = k_\theta(-x_b + x_g)/f, \quad \text{and} \quad v = k_l(l - l_d),$$

where f, x_b, x_g, l indicate the focal length of camera, the horizontal coordinate of the ball center in the image, the horizontal coordinate of the goal center, and the distance from the camera to the goal center, respectively, as shown in Figure 15. l_d shows the desired value of the distance between the ball and the robot, and therefore it is zero.

With only this controller, the success rate is worse than the learning method. Then, we add the following controller when the ball is observed between two goal posts:

$$\omega = \dot{\theta} + k_\theta(\theta - \theta_d), \quad v = \dot{l} + k_l(l - l_d), \quad \text{and} \quad \theta = -x_b/f,$$

where θ_d denotes the desired value of the line of sight viewed from the robot. Table 2 compares the performances by the servo control ($k_\theta = k_l = 0.3$) and our learning method. The success rate by the servo mechanism with two controllers is better than that by our learning method because the servo control mechanism assumed *a priori* knowledge on the external and internal parameters of the camera and the robot such as the focal length of camera, the distance to the ball, the distance between two wheels, and so on. However, the mean steps necessary for getting a shoot by the two servo controls are longer than that by the learning method. Because, the servo control tries to turn its body so that the ball and the goal can be observed in a line,

Table 2. Comaprison with servo control

	Servo Control with one controller	Servo Control with two controllers	Learning Method
Success Rate	72.3%	98.6%	78.5%
Mean Steps	64.3	73.2	56.6

and therefore the trajectory of the robot are oscillating a little bit. While, the trajectory by the learning method does not show such a trajectory. It seems possible to obtain the straight path by the servo control method by selecting appropriate gains k_θ and k_l. However, this needs much more knowledge on dynamics of the robot and other parameters such the so-called "viscous friction."

Next, we tested Fuzzy rules which were programmed by a student who has observed the robot behavior by computer simulation for a long time. The rules are as follows:

```
 1. IF ball-is-observed
 2.     IF ball-size-is-large
 3.         IF goal-position-is-center
 4.             forward;
 5.         ELSE IF goal-position-is-left
 6.             right-backward;
 7.         ELSE IF goal-position-is-right
 8.             left-backward;
 9.         ELSE                                /*goal-is-not-observed*/
10.             IF goal-is-lost-into-left
11.                 right-backward;
12.             ELSE                            /*goal-is-lost-into-right*/
13.                 left-backward;
14.     ELSE                                    /*approach-to-ball*/
15.         forward;
16. ELSE                                        /*ball-is-not-observed*/
17.     IF ball-is-lost-into-left
18.         left-turn;
19.     ELSE                                    /*ball-is-lost-into-right*/
20.         right-turn;
```

We tested the above program with various kinds of conditions in terms of the maximum velocity of the robot and the rolling velocity of the ball (quick, normal, and slow). The success rates by the above program and the learning method are not so different from each other. However, the mean steps to the goal state by the above program is about 10% shorter than that by the learning method. The main reason seems that in the learning method the robot might have experienced useless trials such as kicking a ball by backward motion, which causes inappropriate updates of Q-values in some states, and therefore the optimal path obtained by the learning method might include detours. Note again, the above program is carefully designed by a programmer.

Although the learning method is not superior to the servo control in success rate, or to the Fuzzy rules in mean steps to the goal state, the performance by the learning method is encouraging since the difference does not seem large in spite of not requiring human intervention.

(b) state space construction

The "state-action deviation" problem can be considered as one form of the "perceptual aliasing problem" [10]. It is generally defined as "a problem caused by multiple projections of different actual situations into one observed state". The multiple projections make it very difficult for a robot to take an optimal action. There are several causes for it:

1. Sensor noise: physical sensors are not free from noise, because of which one to many and/or many to one correspondences between actual situations and observed states occur. In our case, the edges of the ball and/or the goal were often not detected, and phantom edges were sometimes detected. Due to such failures, incorrect mapping from one observation to a mismatched state occurs.

2. Sensor processing delay: sensor processing needs non-negligible time to take an adequate action for the observed state. This means that what the robot perceives (state discrimination) is what the environment was some time before. In our case, we have a delay of at least two video frames: one is for image acquisition and another for image processing. We compared the performance of the learned policies with and without delay assuming that the real robot needs two video frames time (66ms). The success rates are 75% by the policy without delay and 60% by the policy with delay. Therefore, we used the learned policy with no delay on a real robot.

3. Difficulty in constructing the consistent staqte and action spaces: as mentioned above, the state and action space are not discrete but continuous in the real world, therefore it is difficult to construct state and action spaces where one action always corresponds to one state transition. We called this the **"state-action deviation problem"**. In utilizing the vision sensor(s), the state-action deviation problem occurs due to the peculiarities of visual information. If we try to construct the state and action spaces in which one action always corresponds to one state transition, we need a large state space that is almost continuous, and suffer from very large variances in the state transition probabilities. Both of these are highly undesirable for the learning method to converge to the optimal solution. We first constructed the state and action spaces which naturally segment the observed state and the effect of the actuator(s), respectively. These coarse segmentation of the state space ("right" or "left" and "small" or "large") is useful in two ways: by reducing the size of the state space and by absorbing the image processing noise. Then, we reconstructed the action space in such a way that one action consists of a sequence of the same action primitive

which is successively executed until the current state changes. This also contributed to reducing the delay in sensor information processing.

4. Hidden states: if there are hidden states which cannot be physically observed by the robot sensors, the system cannot cope with these states without any *a priori* knowledge on them. In our case, the robot easily looses the ball because of its narrow visual field (only 36 degs.). The state of "ball-lost" has a broad area in the real world, and therefore it is difficult for the robot to take an adequate action for this state. Finally, we separated the "ball-lost" state into two states by memorizing the previous state: "ball-lost into right" or "ball-lost into left." This helps the robot take an adequate action to find the ball.

(c) LEM

The LEM algorithm differs in some aspects from the existing approaches in speeding up the search time. In the task decomposition approach [5], Q-learning is closed inside each subtask. In LEM, however, the robot wanders around the field crossing over the states from which it is easy to achieve the goal, even if the robot is initially placed at such states. The LEM just identifies 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, partial knowledge about the degree of difficulty can be used in the LEM method. However it seems difficult to apply such knowledge in a task decomposition approach.

In the Learning with an External Critic (or LEC) [11], the robot receives an advise in each state from the external critic. In order to let LEC work correctly, complete knowledge about the evaluation for the action taken in any state is needed. While, only partial knowledge is used in LEM, completeness of the knowledge does not have any effect on the correct convergence of Q-learning. Only the search time is reduced with complete knowledge.

LEM differs from other approaches such as supervised learning (e.g., [13]) and teaching (e.g., [14]) in that the LEM does not need any demonstrations by a teacher while the teaching method needs at least one or more demonstrations on how instances of the target task can be achieved from some initial states, and supervised learning needs much more demonstrations. These demonstrations must show correct ways to the goal in order for the correct convergence of the learning. The LEM only provides the knowledge of the order of the situations to attain the goal. Even though this knowledge is partially incorrect, the convergence of the learning is guaranteed in LEM, while the other two methods seriously depend on the correctness of the knowledge (demonstrations). In other words, the LEM does not teach how to achieve the goal but from where to start.

From the above differences, the LEM is less restrictive and therefore seems more applicable to a wider range of tasks than supervised learning and teaching approaches.

The most serious and important problem is how to construct the state space. If the state space is not partitioned into sub-states, the LEM cannot be applied. Ideally, the agent should construct the state space automatically from the sensory information and experience. Although some results are reported on this problem [15], [16], we need much more work to make clear what kind of information is necessary to achieve the goal and how to construct (segment) the state space from it.

(d) Limitations

Since the state space of our learning method consists of only the current situations, and does not reflect the history in the past except the ball-lost states, currently the robot cannot shoot a ball into a goal if the robot is located between the ball and the goal. Task decomposition might be one solution for this problem, that is, find a ball, turn so that the ball and the goal can be observed in a line, and move forward. Another one is to put other references in the filed such as corer posts and field lines. Now, we are investigating on these approaches.

8. Concluding Remarks

We presented 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 a real-time vision system. We adopted the Learning from Easy Missions algorithm instead of task decomposition in order to speed up the learning time. The state-action deviation problem due to the peculiarity of visual information is pointed out as one form of the perceptual aliasing problem in applying Q-learning to real robot tasks. Thus, we constructed an action space to cope with this problem.

Although the real experiments are encouraging, still there is a gap between the computer simulation and the real system. We have not made the real robot learn but have only executed the optimal policy obtained by the computer simulation. We are planning to make the real robot learn, starting with the policy obtained by computer simulation as the initial value of the action-value function which is to be determined.

Acknowledgments

This research was supported by the Japanese Sience Research Program under the Project Number 06650301 of the Grand-in-Aid for sientific research from the Ministry of Education, Sience, and Culture.

The authors thank three reviewers for their constructive and valuable comments. They also thank Mr. Eiji Uchibe and Mr. Yasutake Takahashi for their efforts in preparing the revised version of the paper.

Notes

1. To the best of our knowledge, only Whitehead and Ballard [10] used the active vision system and argued the importance of the so-called "perceptual aliasing" problem. However, they have not shown real experiments.
2. A policy f is a mapping from S to A.
3. Here, we define one trial as one step state transition by one action.
4. An attempt ends when an agent achieves a goal or fails (crosses over the field boundary).

References

1. Connel, J. H. and Mahadevan, S. editors, *Robot Learning.* Kluwer Academic Publishers, 1993.
2. Sutton, R. S., "Special issue on reinforcement learning". In R. S. Sutton(Guest), editor, *Machine Learning*, volume 8, pages –. Kluwer Academic Publishers, 1992.
3. Saito, F. and Fukuda, T., "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. Fagg, A. H., Lotspeich, D., and Bekey, G. A., "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. Connel, J. H. and Mahadevan, S., "Rapid task learning for real robot". In J. H. Connel and S. Mahadevan, editors, *Robot Learning*, chapter 5. Kluwer Academic Publishers, 1993.
6. Watkins, C. J. C. H., *Learning from delayed rewards".* PhD thesis, King's College, University of Cambridge, May 1989.
7. Kaelbling, L. P., "Learning to achieve goals". In *Proc. of IJCAI-93*, pages 1094–1098, 1993.
8. Bellman, R., *Dynamic Programming.* Princeton University Press, Princeton, NJ, 1957.
9. Mataric, M., "Reward functions for accelerated learning". In *Proc. of Conf. on Machine Learning-1994*, pages 181–189, 1994.
10. Whitehead, S. D. and Ballard, D. H., "Active perception and reinforcement learning". In *Proc. of Workshop on Machine Learning-1990*, pages 179–188, 1990.
11. Whitehead, S. D., "A complexity analysis of cooperative mechanisms in reinforcement learning". In *Proc. AAAI-91*, pages 607–613, 1991.
12. Inaba, M., "Remote-brained robotics: Interfacing ai with real world behaviors". In *Preprints of ISRR'93*, Pitsuburg, 1993.
13. Pomerleau, Dean A., Knowledge-based training of aritificial neural networks for autonomous robot driving. In J. H. Connel and S. Mahadevan, editors, *Robot Learning*, chapter 2. Kluwer Academic Publishers, 1993.
14. Lin, Long-Ji, Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8:293–321, 1992.
15. Chapman, D. and Kaelbling, L. P., "Input generalization in delayed reinforcement learning: An alogorithm and performance comparisons". In *Proc. of IJCAI-91*, pages 726–731, 1991.
16. Mahadevan, S. and Connell, J., "Automatic programming of behavior-based robots using reinforcement learning". In *AAAI-'91*, pages 768–773, 1991.