Development of Self-Learning Vision-Based Mobile Robots for Acquiring Soccer Robots Behaviors

Takayuki Nakamura

Nara Inst. of Science and Technology Dept. of Information Systems
8916-5, Takayama-cho, Ikoma, Nara 630-01, Japan
takayuki@is.aist-nara.ac.jp

Abstract. An input generalization problem is one of the most important ones in applying reinforcement learning to real robot tasks. To cope with this problem, we propose a self-partitioning state space algorithm which can make non-uniform quantization of the multidimensional continuous state space . This method recursively splits its continuous state space into some coarse spaces called tentative states. It begins by supposing that such tentative states are regarded as the states for Q-learning. It collects Q values and statistical evidence regarding immediate rewards r and Q values within this tentative state space. When it finds out that a tentative state is relevant by the statistical test on minimum description length criterion, it partitions this coarse space into finer spaces. These procedures can make non-uniform quantization of the state space. Our method can be applied to non-deterministic domain because Q-learning is used to find out the optimal policy for accomplishing the given task. To show that our algorithm has generalization capability, we apply our method to two tasks in which a soccer robot shoots a ball into a goal and prevent a ball from entering a goal. To show the validity of this method, the experimental results for computer simulation and a real robot are shown.

Key Words: Self-organizing algorithm, Reinforcement learning, Vision-based mobile robots, Soccer robots.

1 Introduction

RoboCup (The World Cup Robot Soccer) gives a number of research issues for AI and robotics researchers. Such issues involve (1) learning scheme by agents, (2) real-time planning, (3) real-time image processing, (4) coordination or cooperation between agents and so on [1, 2]. Among these technical issues, we currently focus on self-learning scheme by individual agents.

Recently, many researchers in robotics [3] have paid much attention to reinforcement learning methods by which adaptive, reflexive and purposive behavior of robots can be acquired without modeling its environment and its kinematic parameters. A problem in applying reinforcement learning methods to real robot tasks which have continuous state space is that the value function [1] must be

[1] *value function* is a prediction of the return available from each state and is important because the robot can use it to decide a next action. See section 2.1 for more details.

described in a domain consisting of real-valued variables, which means that it should be able to represent the value in terms of infinitely many state and action pairs. For this reason, function approximators are used to represent the value function when a closed-form solution of the optimal policy is not available.

One approach that has been used to represent the value function is to quantize the state and action spaces into a finite number of cells and collect reward and punishment in terms of all states and actions. This is one of the simplest forms of generalization in which all the states and actions within a cell have the same value. In this way, the value function is approximated as a table in which each cell has a specific value (e.g., [3]). However, there is a compromise between the efficiency and accuracy of this table that is difficult to resolve at design time. In order to achieve accuracy, the cell size should be small to provide enough resolution to approximate the value function. But as the cell size gets smaller, the number of cells required to cover the entire state and action spaces grows exponentially, which causes the efficiency of the learning algorithm to become worse because more data is required to estimate the value for all cells. Chapman et. al [4] proposed an input generalization method which splits an input vector consisting of a bit sequence of the states based on the already structured actions such as "shoot a ghost" and "avoid an obstacle." However, the original states have been already abstracted, and therefore it seems difficult to be applied to the continuous raw sensor space of real world. Moore et. al [5] proposed a method to resolve the problem of learning to achieve given tasks in deterministic high-dimensional continuous spaces. It divides the continuous state space into cells such that in each cell the actions available may be aiming at the neighboring cells. This aiming is accomplished by a local controller, which must be provided as a prior knowledge of the given task in advance. The graph of cell transitions is solved for shortest paths in an online incremental manner, but a minimax criterion is used to detect when a group of cells is too coarse to prevent movement between obstacles or to avoid limit cycles. The offending cells are split to higher resolution. Eventually, the environment is divided up just enough to choose appropriate actions for achieving the goal. However, the restriction of this method to deterministic environments might limit its applicability since the real environment is often non-deterministic.

Another approach for representing the value function is to use other types of function approximators, such as neural networks (e.g., [6]), statistical models [7, 8, 9, 10] and so on. The approach consists of associating one function approximator to represent the value of all the states and one specific action. Many researchers have experimented with this approach. For examples, Boyan and Moore [11] used local memory-based methods in conjunction with value iteration; Lin [6] used back-propagation networks for Q-learning; Watkins [12] used CMAC for Q-learning; Tesauro [13] used back-propagation for learning the value function in backgammon. Asada et al. [9] used a concentration ellipsoid as a model of cluster (state) of input vectors, inside which a uniform distribution is assumed. They define a state as a cluster of input vectors from which the robot can reach the goal state or the state already obtained by a sequence of one kind action primitive regardless of its length. However, actual distributions are not always uniform. Ideally, situations that input vectors to be included in their model are not included and vise versa should be avoided.

This paper proposes a new method for incrementally dividing a multidimensional continuous state space into some discrete states. This method recursively splits its continuous state space into some coarse spaces called tentative states. It begins by supposing that such tentative states are regarded as the states for Q-learning. It collects Q values and statistical evidence regarding immediate rewards r and Q values within this tentative state space. When it finds out that a

tentative state is relevant by the statistical test on minimum description length (hereafter, MDL) criterion [14], it partitions this coarse space into finer spaces. These procedures can make non-uniform quantization of the state space. Our method can be applied to non-deterministic domain because Q-learning is used to find out the optimal policy for accomplishing the given task.

The remainder of this article is structured as follows: In the next section, We outline the generalization techniques for reinforcement learning algorithm and give our motivation to the approach described in this paper. In section 3.2, we describe our method to automatically construct the sensor spaces. In section 5, we show the results of the experiments with a simple computer simulation and real robot in which a vision-based mobile robot tries to shoot a ball into a goal and tries to prevent a ball from entering a goal. Finally, we give discussion and concluding remarks.

2 Generalization Techniques for Reinforcement Learning Algorithm

2.1 Basics of Reinforcement Learning Algorithm

One step Q-learning [12] has attracted much attention as an implementation of reinforcement learning because it is derived from dynamic programming [15]. Here, we briefly review the basics of Q-learning [16].

In Q-learning algorithm, it is assumed that the robot can discriminate the set $\boldsymbol{S}$ of distinct world states, and can take one from the set $\boldsymbol{A}$ of actions on the world. The world is modeled as a Markovian 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 $\boldsymbol{A}$. The value function $V^f(s_t)$ associated with a given policy f is defined as:

$$V^f(s_t) \equiv E\left[\sum_{n=0}^{\infty} \gamma^n r_{t+n}\right],$$

where s_t is the state of the system at step t and r_t is the reward received at step t given that the agent started in state s_t 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. The value function measures the expected discounted sum of rewards or expected rewards the robot will receive when it starts from the given state and follows the given policy.

Given definitions of the transition probabilities and the reward distribution, we can solve the optimal policy, using methods from dynamic programming [15]. 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:

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

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:

$$Q(s,a) \Leftarrow (1-\alpha)Q(s,a) + \alpha\left[r(s,a) + \gamma \max_{a' \in A} Q(s',a')\right],$$

where α is a leaning rate (between 0 and 1) and γ is the discounting factor which controls to what degree rewards in the distant future affect the total value of a policy (between 0 and 1).

2.2 Previous Generalization Techniques and Motivation

As shown in the previous section, basic reinforcement learning algorithm assumed that it is possible to enumerate the state and action spaces and store tables of values over them. In a large smooth state space, we generally expect similar states to have similar values and similar optimal actions. Surely, therefore, there should be some more compact representation than a table. Most problems will have continuous or large discrete state spaces; some will have large or continuous action spaces. The problem of learning in large spaces is addressed through generalization techniques, which allow compact storage of learned information and transfer of knowledge between "similar" states and actions.

One method to allow reinforcement-learning techniques to be applied in large state spaces is to use a function approximator so as to represent the value function by mapping a state description to a value. The following explanation originated from [17]. We follow this literature in order to explain the necessity of non-uniform resolution model for representing a function approximator.

Although there have been some positive examples, in general there are unfortunate interactions between function approximation and the learning rules. In discrete environments there is a guarantee that any operation that updates the value function (according to the Bellman equations) can only reduce the error between the current value function and the optimal value function. This guarantee no longer holds when generalization is used. These issues are discussed by Boyan and Moore [11], who give some simple examples of value function errors growing arbitrarily large when generalization is used with value iteration. Their solution to this, applicable only to certain classes of problems, discourages such divergence by only permitting updates whose estimated values can be shown to be near-optimal via a battery of Monte-Carlo experiments. Several recent results [18, 19] show how the appropriate choice of function approximator can guarantee convergence, though not necessarily to the optimal values. Baird's residual gradient technique [20] provides guaranteed convergence to locally optimal solutions.

Perhaps the gloominess of these counter-examples is misplaced. Boyan and Moore [11] report that their counter-examples can be made to work with problem-specific hand-tuning despite the unreliability of untuned algorithms that provably converge in discrete domains. Sutton [21] shows how modified versions of Boyan and Moore's examples can converge successfully. An open question is whether general principles, ideally supported by theory, can help us understand when value function approximation will succeed. In Sutton's comparative experiments with Boyan and Moore's counter-examples, he changes some aspects of the experiments. Boyan and Moore sampled states uniformly in state space, whereas Sutton's method sampled along empirical trajectories. This change must cause different results. Therefore, more careful research associated with this point is needed.

3 Self-Partitioning State Space Algorithm

3.1 Function Approximator with Non-uniform Resolution Model

There are some reasons why designing non-uniform function approximators may be more beneficial than designing uniform ones.

- In case that the designers know, up to a certain degree, prior knowledge of the system (for example, what regions of the state-action space will be used more often.), it may be efficient to design the function approximator such that it may use many resources in more heavily transited regions than in regions of the state space that are known to be visited rarely.
- If the amount of resources is limited, a non-uniform function approximator may make better performance and learning efficiency than that achieved with a uniform function approximator just because the former is able to exploit the resources more efficiently than the later.
- It may be possible to design function approximators that dynamically allocate more resources in certain regions of the state-action space and increase the resolution in such regions as required to perform on-line.

3.2 Details of Our Algorithm

In this work, we define the sensor inputs, actions and rewards as follows:

- Sensor input $\boldsymbol{d}$ is described by a N dimensional vector $\boldsymbol{d} = (d_1, d_2, \cdots, d_N)$, each component $d_i (i = 1 \sim N)$ of which represents the measurement provided by the sensor i. The continuous value d_i is provided by the sensor i. Its range $Range(d_i)$ is known in advance. Based on $Range(d_i)$, a measurement d_i is normalized in such a way that d_i can take values in the semi open interval $[0, 1)$.
- The agent has a set $\boldsymbol{A}$ of possible actions $a_j, \quad j = 1 \sim M$. Such a set is called the action space.
- One of the discrete rewards $r = r_k, \quad k = 1 \sim C$ is given to the agent depending on the evaluation of the action taken at a state.

Our algorithm works as follows:

1. It starts by assuming that the entire environment is as if it were one state. Initially, the total number of the states $|\boldsymbol{S}|$ is one.
2. We utilize a segment tree to classify N dimensional input vector. The inner node at i th depth in the j th level keeps the range $b_i(j) = [t_i^{low}, t_i^{high})$ of a measurement provided by each sensor i. (Actually, j corresponds to the number of iteration of this algorithm.) At each inner node in the j the level, the range of a measurement is partitioned into two equal intervals $b_i^0(j) = [t_i^{low}, (t_i^{low} + t_i^{high})/2)$ and $b_i^1(j) = [(t_i^{low} + t_i^{high})/2, t_i^{high})$. For example, initially $j = 0$, the range of each dimension i is divided into two equal intervals $b_i^0(0) = [0.0, 0.5)$ and $b_i^1(0) = [0.5, 1.0)$. When sensor input vector $\boldsymbol{d}$ has N dimensions, a segment tree whose depth is N is built (see **Fig.1**). The leaf node corresponds to the result of classification for observed sensor input vector $\boldsymbol{d}$. As a result, 2^N leaf nodes are generated. These leaf nodes can represent the situations in the agent's environment. The state space represented by the leaf nodes is called "tentative state space" $\boldsymbol{TS}$. Let $ts_k, \quad k = 1 \sim 2^N$ be the component of the tentative state space which is called "tentative state."

3. Based on this tentative state space $\boldsymbol{TS}$, our algorithm begins Q-learning. In parallel with this process, it gathers statistics in terms of $r(a_i|ts_k = on)$, $r(a_i|ts_k = off)$, $Q(a_i|ts_k = on)$ and $Q(a_i|ts_k = off)$, which indicate immediate rewards r and discounted future rewards Q in case that individual state is "on" or "off," respectively. In this work, it is supposed that if a N dimensional sensor vector $\boldsymbol{d}$ is classified into a leaf node ts_k, the condition of this node ts_k is regarded as "on," otherwise (this means the case that d is classified into the leaf node except ts_k), it is regarded as "off."
4. After Q-learning based on the tentative state space is converging, our algorithm asks the question whether there are some states in the state description such that the r and Q values for states "on"are significantly different from such values for states "off." When the distributions of statistics of ts_k in case of "on" and "off" are different, it is determined that ts_k is *relevant* to the given task. In order to discover the difference between two distributions, our algorithm performs the statistical test based on MDL criterion. In the section 4.1 and 4.2, these procedures are explained.
5. (a) If there is the state ts'_k adjoining the state ts_k which is shown to be relevant such that the statistical characteristic of Q values and actions assigned at the adjoining state are same, merge these two states into one state.
   (b) Otherwise, skip this step.
6. Each leaf nodes ts_k is represented by a combination of intervals each of which corresponds to the range of a measurement provided by each sensor i. These intervals in ts_k which is shown to be relevant are bisected. As a result, in terms of one ts_k, 2^N leaf nodes are generated and correspond to tentative states.
7. Supposing that these tentative states are regarded as the states in Q-learning, our algorithm performs Q-learning again.
8. Until our algorithm can't find out the relevant leaf nodes, the procedures $2 \sim 7$ are repeated. Finally, a hierarchical segment tree is constructed to represent the partitioning of the state space for achievement of a given task.

After the learning, based on Q values stored at leaf nodes, the agent takes actions for accomplishing the given task.

4 The Relevance Test Based on MDL Criterion

Here, we explain how to determine whether a state is *relevant* to the task or not. **Fig. 2** shows the difference between the distributions of r or Q values regarding to the state ts_k in case that ts_k is *relevant* or *irrelevant*. As shown in upper part of this figure, when two peaks of the distributions of r or Q values ,which correspond to pair of $r(a_i|ts_k = on)$ and $r(a_i|ts_k = off)$, or pair of $Q(a_i|ts_k = on)$ and $Q(a_i|ts_k = off)$), can be clearly discriminated, it is supposed that ts_k is relevant to the given task because the such state affects the value of state more heavily than the other states does, therefore, it affects how the robot should act at next time step. On the contrary, in case that two peaks are ambiguous as shown in bottom part of this figure, it is considered that ts_k is *irrelevant* to the given task. Actually, we perform the statistical test with respect r and Q values based on MDL criterion [14] in order to distinguish the distribution of such reinforcement values.

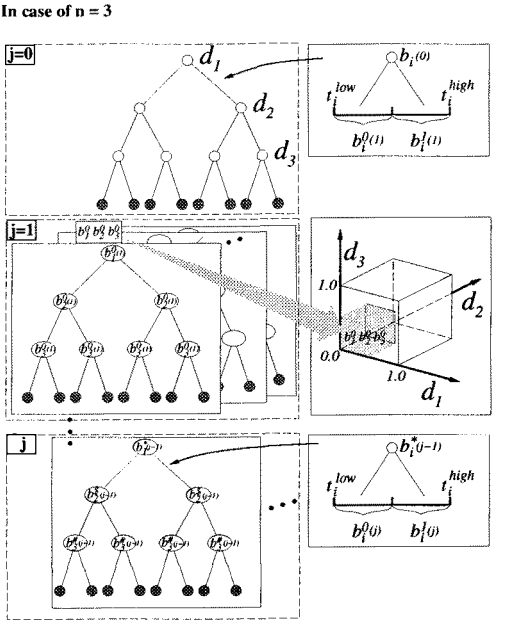


Fig. 1. Representation of state space by a hierarchical segment tree

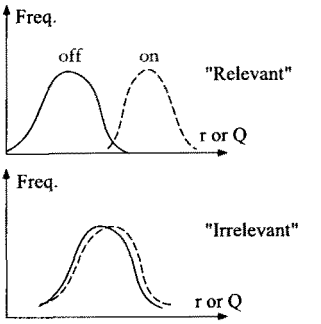


Fig. 2. Criterion for determining the relevance of the state

4.1 The statistical test for r value

Since the immediate reward r_j is given at each trial among one of C mutually exclusive rewards $r_j \quad j = 1, 2, \cdots C$, the distribution of r_j in the state ts_k follows a multi-nominal distribution. Let n be the number of independent trials, k_i be the number of event E_i and p_i be the probability that the event E_i occurs where $\sum_{i=1}^{c} p_i = 1$. The probability that E_1 occurs k_1 times, E_2 occurs k_2 times, ... E_c occurs k_c times, can be represented by the multi-nominal distribution as follows:

$$p(k_1, \cdots, k_c | p_1, \cdots, p_c) = \frac{n!}{k_1! \cdots k_c!} p_1^{k_1} \cdots p_c^{k_c}$$

where, $0 \leq k_i \leq n (i = 1 \sim c), \quad \sum_{i=1}^{c} = n.$

Supposing **Table 1** shows the distribution of immediate rewards in case that the state ts_k is "*on*" or "*off*," our algorithm tries to find the difference between the distributions of rewards in two cases of ts_k "on" and "off" based on this table.

Table 1. The distribution of rewards in ts_k

Rewards	On	Off
R_1	$n(On, R_1)$	$n(Off, R_1)$
R_2	$n(On, R_2)$	$n(Off, R_2)$
:	:	:
R_C	$n(On, R_C)$	$n(Off, R_C)$
	$n(On)$	$n(Off)$

$n(i)$: the frequency of sample data in the state i $(i = 1, \cdots, S)$

$n(i, r_j)$: the frequency of reward r_j $(j = 1, \cdots, C)$ given in the state i

$p(r_j|i)$: the probability that reward r_j is given in the state i

$$\sum_{j=1}^{C} n(i, r_j) = n(i), \quad \sum_{j=1}^{C} p(r_j|i) = 1, \quad (i = 1, \cdots S).$$

The probability $P(\{n(i, r_j)\}|\{p(r_j|i)\})$ that the distribution of the immediate rewards are acquired as shown in **Tab. 1** $\{n(i, r_j)\}$, $(i = 1 \cdots, S, \quad j = 1, \cdots, C)$, can be described as follows:

$$P(\{n(i, r_j)\}|\{p(r_j|i)\}) = \prod_{i=1}^{S} \left\{ \frac{n(i)!}{\prod_{j=1}^{C} n(i, r_j)!} \prod_{j=1}^{C} p(r_j|i)^{n(i, r_j)} \right\}$$

The likelihood function L of this multi-nominal distribution can be written as follows:

$$L(\{p(r_j|i)\}) = \sum_{i=1}^{S} \sum_{j=1}^{C} n(i, r_j) \log p(r_j|i) + K, \text{where} K = \log \left\{ \frac{\prod_{i=1}^{S} n(i)!}{\prod_{i=1}^{S} \prod_{j=1}^{C} n(i, r_j)!} \right\}.$$

When two multi-nominal distributions in case that each tentative state ts_k is "on" and "off" can be considered to be same, the probability $p(r_j|i)$ that the immediate reward r_j is given in the state i can be modeled as follows:

$$\text{M1: } p(r_j|i) = \theta(r_j) \quad i = 1, \cdots, S, \quad j = 1, \cdots, C.$$

Furthermore, its likelihood function L_1 can be written by

$$L_1(\{\theta(r_j)\}) = K + \sum_{j=1}^{C} \left[\left\{ \sum_{i=1}^{S} n(i, r_j) \right\} \log \theta(r_j) \right].$$

Therefore, the maximum likelihood ML_1 can be written by

$$ML_1 = L(\theta(\hat{r}_j)) = L\left(\frac{\sum_{i=1}^{S} n(i, r_j)}{\sum_{i=1}^{S} n(i)} \right).$$

where $\theta(\hat{r}_j)$ is the maximum likelihood estimation of $\theta(r_j)$. Therefore, the description length $l_{MDL}(M1)$ of the model $M1$ based on MDL principle is

$$l_{MDL}(M1) = -ML_1 + \frac{C-1}{2} \log \sum_{i=1}^{S} n(i).$$

On the contrary, when two multi-nominal distributions in case of "on" and "off" can be considered to be different, the probability $p(r_j|i)$ can be modeled as follows:

$$\text{M2: } p(r_j|i) = \theta(r_j|i) \quad i = 1, \cdots, S, \quad j = 1, \cdots, C.$$

Furthermore, its likelihood function L_2 can be written by

$$L_2(\{\theta(r_j|i)\}) = K + \sum_{i=1}^{S} \sum_{j=1}^{C} n(i, r_j) \log \theta(r_j|i).$$

In the same way, the maximum likelihood ML_2 can be written by

$$ML_2 = L(\theta(\hat{r_j})) = L\left(\frac{n(i, r_j)}{n(i)}\right).$$

Therefore, the description length $l_{MDL}(M2)$ of the model $M2$ based on MDL principle is

$$l_{MDL}(M2) = -ML_2 + \frac{S(C-1)}{2} \log \sum_{i=1}^{S} n(i).$$

Based on MDL criterion, we can suppose that discovering the difference between two distributions is equivalent to determining which model is appropriate for representing the distribution of data. In this work, the difference between the distributions is found based on the following conditions:

if $l_{MDL}(M1) > l_{MDL}(M2)$
Two distributions are different.
if $l_{MDL}(M2) \geq l_{MDL}(M1)$
Two distributions are same.

4.2 The statistical test for Q value

In order to distinguish the distribution of sampled data of Q values, we perform the statistical test based on MDL criterion. Let $\boldsymbol{x}^n$ and $\boldsymbol{y}^m$ be the sample data $(x_1, x_2 \cdots, x_n)$ and $(y_1, y_2 \cdots, y_m)$, respectively. $\boldsymbol{x}^n$ and $\boldsymbol{y}^m$ indicate a history of $Q(a_i|ts_k = on)$ and $Q(a_i|ts_k = off)$, respectively. We'd like to know whether these two sample data $\boldsymbol{x}^n$ and $\boldsymbol{y}^m$ come from the two different distributions or the same distribution. Here, we assume the following two model for the distribution of sampled data are $M1$ based on one normal distribution and $M2$ based on two normal distributions.

$$\text{M1:} \quad N(\boldsymbol{\mu}, \boldsymbol{\sigma}^2), \qquad \text{M2:} \quad N(\boldsymbol{\mu}_1, \boldsymbol{\sigma}'^2), N(\boldsymbol{\mu}_2, \boldsymbol{\sigma}'^2)$$

The normal distribution with mean μ and variance σ is defined by

$$f(x : \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{\frac{-(x-\mu)^2}{2\sigma^2}\right\}.$$

If both $\boldsymbol{x}^n$ and $\boldsymbol{y}^m$ follow the model $M1$, the probabilistic density function of these sampled data can be written by

$$\prod_{i=1}^{n} f(x_i : \mu, \sigma^2) + \prod_{i=1}^{m} f(y_i : \mu, \sigma^2).$$

The likelihood function LL_1 of the model $M1$ can be written by

$$LL_1 = \sum_{i=1}^{n} \log f(x_i : \mu, \sigma^2) + \sum_{i=1}^{m} \log f(y_i : \mu, \sigma^2).$$

Therefore, the maximum likelihood MLL_1 can be derived as follows:

$$MLL_1 = -\frac{n+m}{2}(1 + \log 2\pi\hat{\sigma}^2),$$

where $\hat{\sigma}^2$ is the maximum likelihood estimated variance of σ^2 and can be estimated as follows:

$$\hat{\mu} = \frac{1}{n+m}\left\{\sum_{i=1}^{n} x_i + \sum_{i=1}^{m} y_i\right\}, \qquad \hat{\sigma^2} = \frac{1}{n+m}\left\{\sum_{i=1}^{n}(x_i - \hat{\mu})^2 + \sum_{i=1}^{m}(y_i - \hat{\mu})^2\right\},$$

where $\hat{\mu}$ is the maximum likelihood estimated mean of μ. Therefore, the description length $l_{MDL}(M1)$ of the model $M1$ based on MDL principle is

$$l_{MDL}(M1) = -MLL_1 + \log(n+m).$$

On the contrary, if $\boldsymbol{x}^n$ and $\boldsymbol{y}^m$ follow the model $M2$, the probabilistic density function of these sampled data can be written by

$$\prod_{i=1}^{n} f(x_i : \mu_1, \sigma'^2) + \prod_{i=1}^{m} f(y_i : \mu_2, \sigma'^2).$$

The likelihood function LL_2 of the model $M2$ can be written by

$$LL_2 = \sum_{i=1}^{n} \log f(x_i : \mu_1, \sigma'^2) + \sum_{i=1}^{m} \log f(y_i : \mu_2, \sigma'^2).$$

Therefore, the maximum likelihood MLL_2 can be derived as follows:

$$MLL_2 = -\frac{n+m}{2}(1 + \log 2\pi\hat{\sigma'}^2),$$

where $\hat{\sigma_1}^2$ and $\hat{\sigma_2}^2$ is the maximum likelihood estimated variance of σ_1^2 and σ_2^2, respectively. These statistics can be estimated as follows:

$$\hat{\mu_1} = \frac{1}{n}\sum_{i=1}^{n} x_i, \qquad \hat{\mu_2} = \frac{1}{m}\sum_{i=1}^{m} y_i,$$
$$\hat{\sigma'^2} = \frac{1}{n+m}\left\{\sum_{i=1}^{n}(x_i - \hat{\mu_1})^2 + \sum_{i=1}^{m}(y_i - \hat{\mu_2})^2\right\},$$

where $\hat{\mu_1}$ and $\hat{\mu_2}$ is the maximum likelihood estimated mean of μ_1 and μ_2, respectively. Therefore, the description length $l_{MDL}(M2)$ of the model $M2$ based on MDL principle is

$$l_{MDL}(M2) = -MLL_2 + \frac{3}{2}\log(n+m).$$

We can recognize the difference between the distributions based on the following condition:

if $l_{MDL}(M1) > l_{MDL}(M2)$
$\boldsymbol{x}$ and $\boldsymbol{y}$ arise from the different normal distributions
if $l_{MDL}(M2) \geq l_{MDL}(M1)$
$\boldsymbol{x}$ and $\boldsymbol{y}$ arise from the same normal distribution

5 Experimental Results

T show that our algorithm has a generalization capability, we apply it to acquire two different behaviors: one is a shooting behavior and the other is a defending behavior for soccer robots.

5.1 Simulation

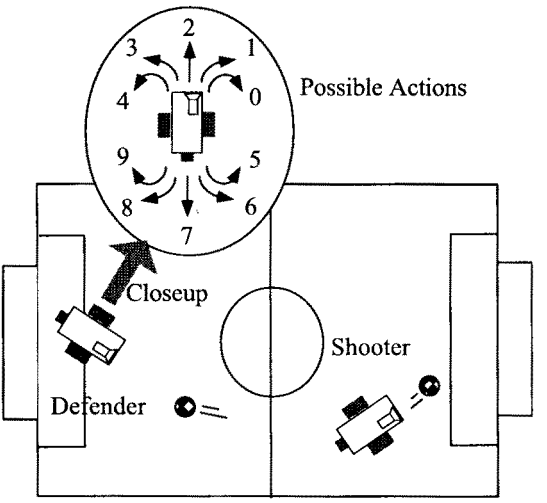


Fig. 3. Simulation environment

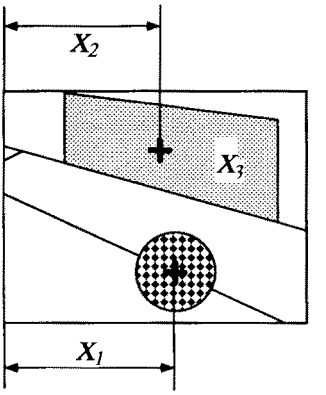


Fig. 4. Input vector as sensor information

We consider an environment shown in **Fig. 3** where the task for a mobile robot is to shoot a ball into a goal or to defend a shot ball. The environment consists of a ball and a goal, and the mobile robot has a single CCD camera. 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.

We performed the computer simulation with the following specifications. The field is 1.52m × 2.74m. The goal posts are located at the center of the left and right line of the rectangle (see **Fig. 3**) and its height and width are 0.20m and 0.5m, respectively. The robot is 0.10m wide and 0.20m long and kicks a ball of diameter 0.05m. The maximum translation velocity is 5cm/s. The camera is

horizontally mounted on the robot (no tilt) and is in off-centered position. 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.

The robot is driven by two independent motors and steered by front and rear wheels which is driven by one motor. Since we can send the motor control commands such as "move forward or backward in the given direction," all together, we have 10 actions in the action primitive set $\boldsymbol{A}$ as shown in **Fig. 3**. 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. Actually, a stop motion does not causes any changes in the environment, we do not take into account this action primitive.

In computer simulation, we take into account the delay due to sensory information processing. The contents of the image processing are color filtering (a ball and a goal are painted in red and blue, respectively), localizing and counting red and blue pixels, and vector calculation. We are using a general graphic workstation for image processing in the real robot experiments. Since it is not specialized on an image processing, it takes about 66 ms to perform these processes. As a result, in the simulation, the robot is assumed to take an action primitive every 66ms.

The size of the image taken by the camera is 256 × 240 pixels. An input vector $\boldsymbol{x}$ to our algorithm consists of:

- x_1: the horizontal position of the ball in the image, that ranges from 0 to 256 pixels,
- x_2: the horizontal position of the goal in the image ranging from 0 to 256,
- x_3: the area of the goal region in the image, that ranges from 0 to 256×240 pixels^2.

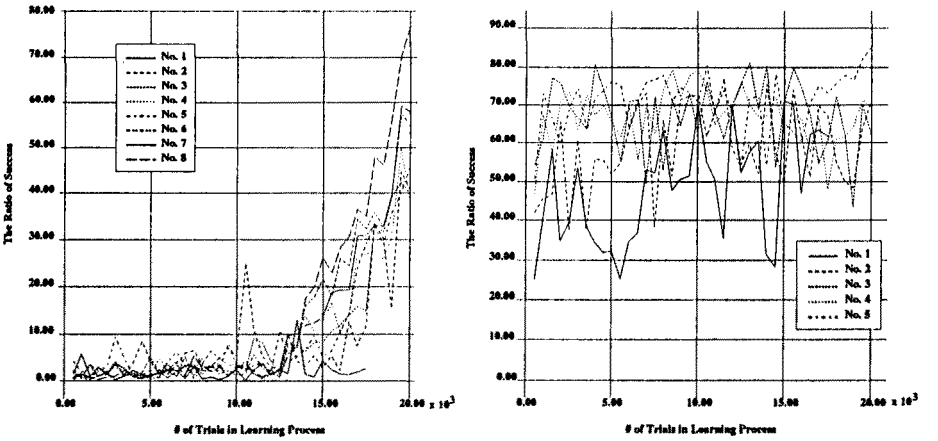
After the range of these values is normalized in such a way that the range may become the semi open interval $[0, 1)$, they are used as inputs of our method.

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 ($\gamma = 0.9$). In this work, we set the learning rate $\alpha = 0.25$. In case that the shooting behavior tried to be acquired by our method, as a reward value, 1 is given when the robot succeeded in shooting a ball into a goal, 0.3 is given when the robot just kicked a ball, -0.01 is given when the robot went out of field, 0 is given otherwise. In the same way, in case that the defending behavior tried to be acquired by our method, as a reward value, -0.7 is given when the robot failed in preventing a ball from entering a goal, 1.0 is given when the robot just kicked a ball, -0.01 is given when the robot went out of field, 0 is given otherwise.

In the learning process, Q-learning continues until the sum of estimated Q values seems to be almost convergent. When our algorithm tried to acquire the shooting behavior, our algorithms ended after it iterated the process (Q-learning + statistical test) 8 times. In this case, about $160K$ trials were required to converge our algorithm and the total number of the states is 246. In case that our algorithm tried to acquire the defending behavior, our algorithms ended after it iterated the process (Q-learning + statistical test) 5 times. In this case, about $100K$ trials were required to converge our algorithm and the total number of the states is 141.

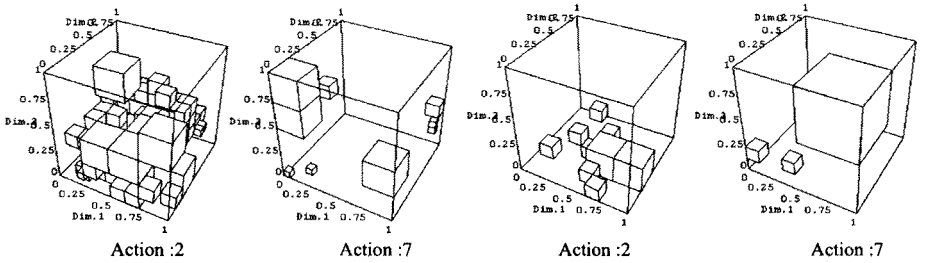
Fig. 5 shows the success ratio versus the step of trials in the two learning processes that one is for acquiring a shooting behavior the other is for a defending behavior. We define the success rate as (# of successes)/(# of trials) × 100(%).

As you can see, the bigger the number of iteration is, the higher the success ratio at the final step in each iteration. This means that our algorithm gradually made better segmentation of state space for accomplishing the given task.



(a) In case of shooting behavior (b) In case of defending behavior

Fig. 5. The success ratio versus the step of trial



(a) The state space for shooting behavior (b) The state space for defending behavior

Fig. 6. The partitioned state space

Fig. 6 shows the partitioned state spaces obtained by our method. The upper part of this figure shows the state space for the shooting behavior, the bottom part shows one for defending behavior. In each figure, *Dim.*1, *Dim.*2 and *Dim.*3 shows the position of ball, the position of goal and the area of goal region,

respectively. Furthermore, in each figure, *Action2* and *Action7* corresponds to "moving forward" and "moving backward," respectively.

For the sake of readers understanding, one cube in the partitioned state space corresponds to one state. For example, the top left shows a group of the cube where the action 2 is assigned as an optimal action. As shown in this figure, many cubes where forward actions are assigned concentrate around the center of the entire state space. This means that the robot will take an forward action if the ball and goal are observed around the center of field of its view. This shows very natural behavior for shooting a ball into a goal.

In the bottom right figure, there is one large cube. This means that the the robot will take an backward action if large goal are observed around the center of field of its view. This strategy is plausible behavior for preventing a ball from entering a goal because the robot will have to go back in front of own goal after it moved out there in order to kick a ball.

Fig.7 shows a typical sequence of the shooting behavior in the simulation acquired by our method. The bottom part of each figure in the **Fig. 7** shows the processed image where the white rectangle and dark gray circle indicates the goal and ball region, respectively. Once the robot found the ball, the robot ran to the ball quickly. After that, the robot kicked the ball into the goal.

Fig.8 shows the defending behavior in the simulation acquired by our method. The robot initially looked around because the ball was not observed from its initial position. Then, once the robot found the ball, the robot ran to the ball quickly. After that, the robot kicked the ball to prevent a ball from approaching the goal. The robot followed the ball until the area of opponent's goal grows to a certain extent. Then in order to go back to the front of own goal, the robot took a backward action. It seems that the robot utilizes the area of the opponent's goal to localize its own position in the field. Note that this behavior is just the result of learning and not hand-coded.

5.2 Real Robot Experiments

Fig. 9 shows our real robot system which we have developed to take part in RoboCup-97 competition where several robotic teams are competing on a field. So, the system includes two robots which have the same structure: one for a shooter, the other for a defender. Off-board computer SGI ONYX (R4400/250MHz) perceives the environment through on-board cameras, performs the decision making based on the learned policy and sends motor commands to each robot. A CCD camera is set at bottom of each robot in off-centered position . Each robot is controlled by SGI ONYX through radio RS232C. The maximum vehicle speed is about 5cm/s. The images taken by the CCD camera on each robot are transmitted to a video signal receiver. In order to process two images (one is sent from the shooter robot, the other from the defender robot) simultaneously, two video signals are combined into one video signal by a video combiner on PC. Then, the video signal is sent to SGI ONYX for image processing. The color-based visual tracking routine is implemented for tracking and finding a ball and a goal in the image. In our current system, it takes 66 ms to do this image processing for one frame.

Simple Color-Based Tracking

Our simple tracking method is based on tracking regions with similar color information from frame to frame. We assume the existence of the color models (CM_{ball}, CM_{goal}) for tracking targets, which are estimated at initialization. Actually, we use only hue values (H_{ball} and H_{goal}) of targets as (CM_{ball} and

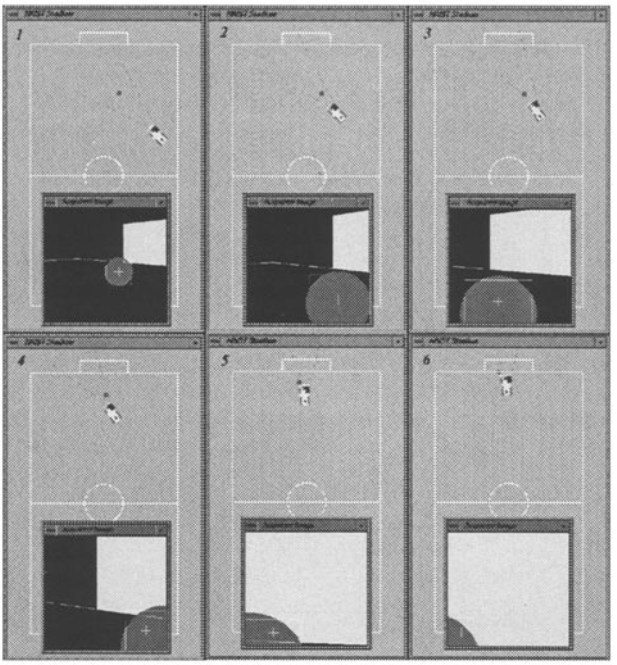


Fig. 7. Shooting behavior in simulation

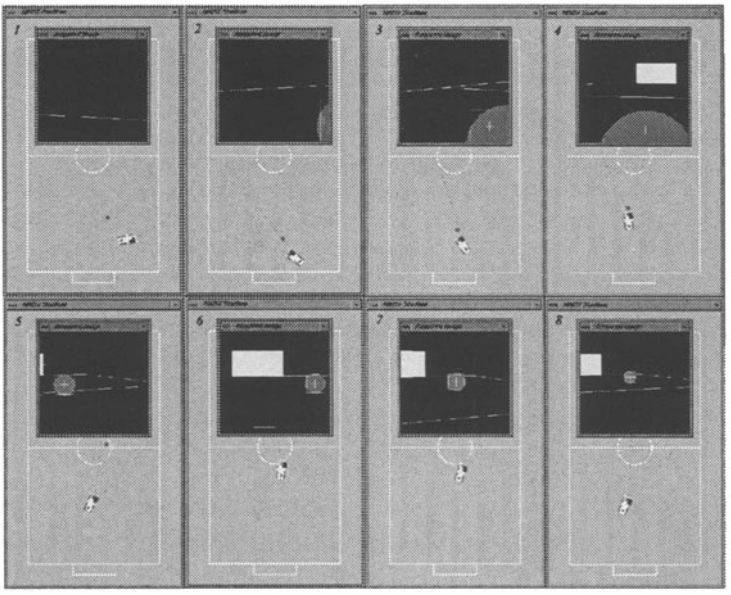


Fig. 8. Defending behavior in simulation

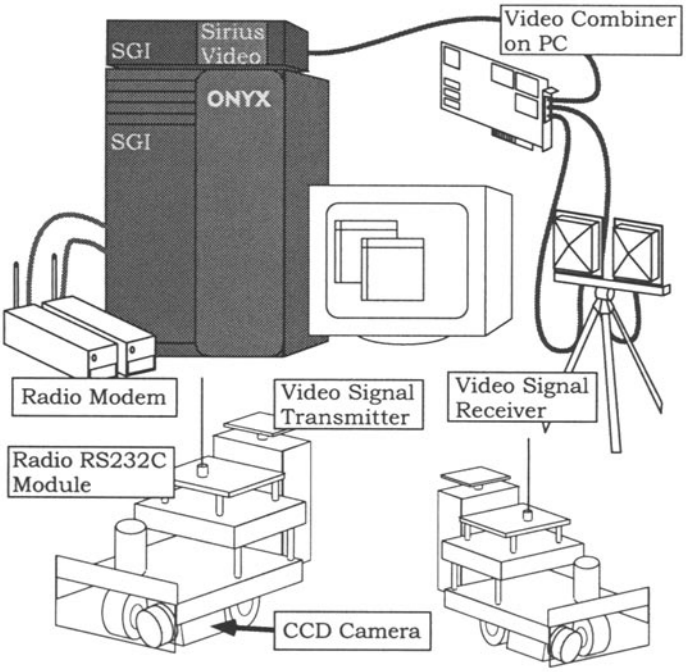


Fig. 9. Our robots and system

CM_{goal}) to reduce its sensitivity due to changes in illumination. We define a fitness function $\Phi_{target}(x, y)$ at a pixel (x, y) as a criterion for extracting a target region in the image,

$$\Phi_{target}(x,y) = \begin{cases} 1 & H_{target} - \sigma_{target} \leq H(x,y) \leq H_{target} + \sigma_{target}, \\ 0 & \text{Otherwise} \end{cases}$$

,where $H(x, y)$ and σ_{target} show hue value at (x, y) and a threshold for extraction, respectively. In our current implementation, we set $\sigma_{ball} = 10$ and $\sigma_{goal} = 20$. These values can be calculated based on the standard deviation of H_{ball} and H_{goal} which were estimated at initial estimation process. Based on $\Phi_{target}(x, y)$, the best estimate $(\hat{x}_{target}, \hat{y}_{target})$ for the target's location is calculated as follows:

$$\hat{x}_{target} = \frac{\sum_{(x_i,y_i)\in R} x_i \Phi_{target}(x_i, y_i)}{\sum_{(x_i,y_i)\in R} \Phi_{target}(x_i, y_i)}, \quad \hat{y}_{target} = \frac{\sum_{(x_i,y_i)\in R} y_i \Phi_{target}(x_i, y_i)}{\sum_{(x_i,y_i)\in R} \Phi_{target}(x_i, y_i)},$$

where R shows the search area. Initially, R implies an entire image plane. After initial estimation for the location of the target, we can know the standard deviations $\sigma(\hat{x}_{target})$ and $\sigma(\hat{y}_{target})$ regarding $(\hat{x}_{target}, \hat{y}_{target})$. Therefore, based on the deviations, R is restricted to a local region during the tracking process as follows:

$$R : \{(x,y)|\ \hat{x}_{target} - 2.5\sigma(\hat{x}_{target}) \leq x \leq \hat{x}_{target} + 2.5\sigma(\hat{x}_{target}),$$
$$\hat{y}_{target} - 2.5\sigma(\hat{y}_{target}) \leq y \leq \hat{y}_{target} + 2.5\sigma(\hat{y}_{target})\}.$$

$\sum_{(x_i,y_i)\in R} \Phi_{target}(x_i, y_i)$ shows the area of the target in the image. Based on this value, we judge the appearance of the target. If this value is lower than the pre-defined threshold, the target is considered to be lost, then R is set to be the entire image plane for estimation at next time step. We set this threshold for the target area $= 0.05 * S$, where S shows the area of the entire image. **Fig. 10** shows examples of processed images during the task execution. In these figures, a light gray and a dark gray "X" show the estimated location of the ball and the goal, respectively. In the same way, the white rectangles surrounded with dotted line in each image show the estimated area of the ball and the goal.

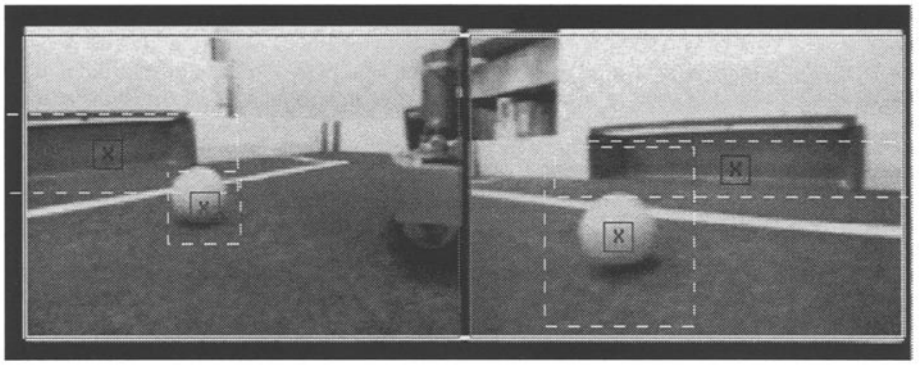


Fig. 10. A example of processed images taken by the robots

Experimental Results

Fig. 11 shows how a real robot shoots a ball into a goal based on the state space obtained by our method. 6 images are shown in raster order from the top left to the bottom right in every 2.0 seconds, in which the robot tried to shoot a ball, but failed, then moved backward so as to find a position to shoot a ball, finally succeeded in shooting. Note that the backward motion for retry is just the result of learning and not hand-coded.

Fig. 12 shows how a real robot prevents a ball shot by the opponent's robot from entering a goal based on the state space obtained by our method. 6 images are shown in raster order from the top left to the bottom right in every 2.0 seconds. This sequence was recorded at the RoboCup-97 competition. In the top left image, the white circle shows the position of the defender robot and the white arrow shows the position of the ball. In the time step 1 and 2, first, the opponent's robot tried to shoot a ball, then the defender robot prevent a ball from entering a goal. In the time step 3 $\sim$ 6, another opponent's robot got a ball cleared by the defender robot and tried to shoot a ball again, then the defender robot defend a goal again. As shown in these figures, the defender robot always moves in front of own goal to find out a shot ball as soon as possible. Note that this behavior is just the result of our learning algorithm and not hand-coded.

6 Concluding Remarks and Discussion

We have proposed a method for self-partitioning the state space which recursively splits continuous state space into some coarse regions based on the relevance test in terms of reward values. Our method utilized a hierarchical segment tree in order to represent the non-uniform partitioning of the state space. This representation has an advantage for approximating the non-uniform distribution of sample data which frequently occurs in real world. We also have shown the validity of the method with computer simulations and real robot experiments at our lab. and the RoboCup-97 competition. We think that segmentation of sensory data from the environment should depend on the purpose (task), capabilities (sensing, acting, and processing) of the robot, and its environment. The state spaces obtained by our method (**Fig.6** indicates such segmentations) correspond to the internal representations of the robot for accomplishing given tasks. Furthermore, we think that the resolution of state spaces indicates the importance that how much attention the robot have to pay to a part of the state space. In these sense, by our method, our robot acquired the the subjective representation for its environment and the given task.

Current version of our algorithm has two kinds of problem:

- If the dimension of the sensor space is higher, the computational time and amount are enormous, and as a result the learning might not converge correctly. The dimension of the sensor space is associated with the number of features which are used to describe all situations in the environment. In our current implementation, we used input vector consisting of three parameters which were enough to acquire the shooting and defending behaviors. However, there is no guarantee that these parameters are enough in case that our method tries to acquire the higher-level behaviors such as strategic coordination between two robots. Generally, selection of features from raw images is really a difficult problem. The selection of feature which is necessary to accomplish a give task might be much harder when such a feature changes depending on situations. Since use of all possible sensory information seems impossible, a learning mechanism for selecting features from the sensory data should be developed. As a result, an effective method for coping with higher dimension space might be developed.
- We currently define the quantization of the action space in advance. Since our robot has only two degrees of freedom (DOFs), abstraction of action space is not very important. However, it is generally important to quantize the action space in case that the dimension of the action space is higher and/or we want to use real-valued motor signals to control the robot precisely. For example, if the robot has an active vision system with pan and tilt mechanism in addition to the two DOFs for driving system, the action space should be abstracted depending on situation in order to coordinate between sensing and driving actions. The abstraction of action space is as necessary as the abstraction of sensor space.

7 Acknowledgments

The main idea of this paper was thought of while I stayed at AI Lab of Comp. Sci. Dept. of Brown University. I would like to thank Prof. L. P. Kaelbling for her helpful comments during my stay. I also would like to thank Prof. M. Imai for providing research fund and S. Morita Japan SGI Cray Corp for lending SGI ONYX to me. Finally, I want to say thank you to my colleague M. Iwase for his help in the development of our robot.

References

1. H. Kitano, M. Tambe, Peter Stone, and et.al. "The robocup synthetic agent challenge 97". In *Proc. of The First International Workshop on RoboCup*, pages 45–50, 1997.
2. M. Asada, Y. Kuniyoshi, A. Drogoul, and et.al. "The robocup physical agent challenge:phase i(draft)". In *Proc. of The First International Workshop on RoboCup*, pages 51–56, 1997.
3. J. H. Connel and S. Mahadevan, editors. *Robot Learning*. Kluwer Academic Publishers, 1993.
4. 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.
5. A. W. Moore and C. G. Atkeson. "The parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces". *Machine Learning*, 21:199–233, 1995.
6. Long-Ji Lin. "Self-improving reactive agents based on reinforcement learning, planning and teaching". *Machine Learning*, 8:293–321, 1992.
7. H. Ishiguro, R. Sato, and T. Ishida. "Robot oriented state space construction". In *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems 1996 (IROS96)*, volume 3, 1996.
8. A. Ueno, K. Hori, and S. Nakasuka. "Simultaneous learning of situation classification based on rewards and behavior selection based on the situation". In *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems 1996 (IROS96)*, volume 3, 1996.
9. M. Asada, S. Noda, and K. Hosoda. "Action-based sensor space categorization for robot learning". In *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems 1996 (IROS96)*, volume 3, 1996.
10. Y. Takahashi, M. Asada, and K. Hosoda. "Reasonable performance in less learning time by real robot based on incremental state space segmentation". In *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems 1996 (IROS96)*, volume 3, pages 1518–1524, 1996.
11. J. Boyan and A. Moore. "Generalization in reinforcement learning: Safely approximating the value function". In *Proceedings of Neural Information Processings Systems 7*. Morgan Kaufmann, January 1995.
12. C. J. C. H. Watkins. *"Learning from delayed rewards"*. PhD thesis, King's College, University of Cambridge, May 1989.
13. G. Tesauro. "Practical issues in temporal difference learning". *Machine Learning*, 8:257–277, 1992.
14. J. Rissanen. *Stochastic Complexity in Statistical Inquiry*. World Scientific, 1989.
15. R. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, 1957.
16. L. P. Kaelbling. "Learning to achieve goals". In *Proc. of IJCAI-93*, pages 1094–1098, 1993.
17. L. P. Kaelbling, M. L. Littman, and A. W. Moore. "Reinforcement learning: A survey". *Journal of Artificial Intelligence Research*, 4, 1996.
18. G. J. Gordon. "Stable function approximation in dynamic programming". In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 261–268, 1995.

19. J. N. Tsitsiklis and B. V. Roy. "Feature-based methods for large scale dynamic programming". *Machine Learning*, 22:1, 1996.
20. L. Baird. "Residual algorithms: Reinforcement learning with function approximation". In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 30–37, 1995.
21. R. S. Sutton. "Generalization in reinforcement learning: Successful examples using sparse coarse coding". In *Proc. of Neural Information Processing Systems 8*, 1996.

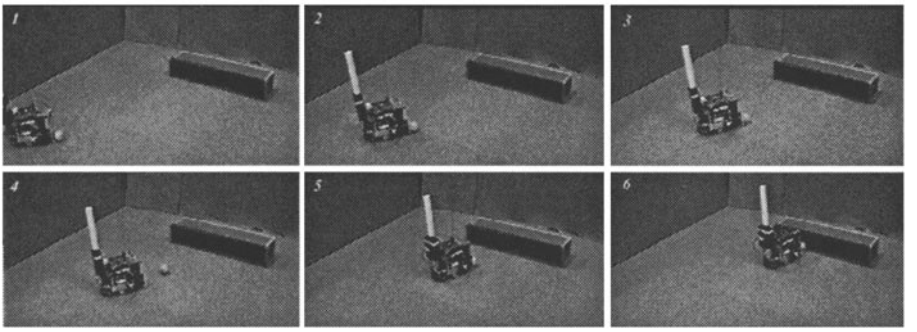


Fig. 11. Shooting behavior on our robot

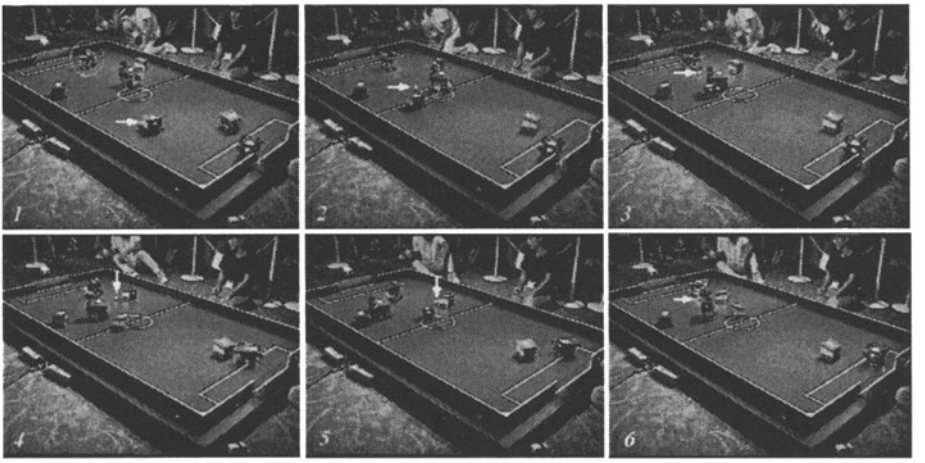


Fig. 12. Defending behavior on our robot