Doctor Thesis, The University of Tokyo

State-Action Map Compression by using Vector Quantization for Decision Making of Autonomous Robots

Ryuichi Ueda

Supervisor: Prof. Tamio Arai

February, 2007

This thesis is put in type with LaTeX 2_ε and $\mathcal{AMS}$-LaTeX.

To Rie and our baby

Acknowledgements

Prof. Tamio Arai gives me the chance of this research, and his advice is gratefully acknowledged.

Mr. Takeshi Fukase and Dr. Yuichi Kobayashi have its roots in the study of vector quantization for dynamic programming. I have inherited their initial studies and have been able to advance this research as this thesis.

The persons who belong to and belonged to Team ARAIBO have gratefully helped me to conduct the experiments in this paper, and have provided many cutting-edge software applications for computers and robots. Mr. Kazunori Asanuma and Mr. Yoshiaki Jitsukawa provided me superb simulators of ERS-210 and ERS-7 respectively. Mr.Shogo Kamiya and Mr. Masaki Komura created and maintained the walking actions of ERS-210. Mr. Toshifumi Kikuchi maintained look-up tables for color recognition by the camera of ERS-210. Mr.Kazutaka Takeshita and Mr.Kohei Sakamoto coded some algorithms that became the prototypes of my codes.

This paper is reviewed by Prof. Kanji Ueda with Research into Artifacts, Center for Engineering, Prof. Ichiro Sakuma with Department of Precision Engineering, School of Engineering, Prof. Hitoshi Iba with Department of Frontier Informatics, Graduate School of Frontier Sciences, and Prof. Jun Ota with Department of Precision Engineering, School of Engineering. I received lots of advice and feedback from them.

This research is partially supported by the Ministry of Education, Science, Sports and Culture, Grant-in-Aids for Young Scientists (B) (No. 17760199), and for Scientific Research (B) (No. 16300050).

Abstract

In this thesis, we propose a methodology for implementing decision making rules (policies) of robots on their limited amount of random access memory (RAM). Decision making policies are computed by a dynamic programming (DP) method and they are compressed by vector quantization (VQ).

In Chapter 1, the background, related works, and purpose of our study are described. An optimal control problem and the finite Markov decision process that are handled in this paper are formulated so that we explain the general idea of the optimality of policies. After that, we set up the problem of how to represent policies that fulfill the following conditions: 1) they are **reflexive policies** that can be read only with some operations for data access, 2) they are efficient from the standpoint of optimal control, and 3) they can be installed with small amount of memory on robots. Those policies are useful for various robots, especially for autonomous robots that have poor computing resources, or that must move in real-time. This problem has never been tackled directly even though there are meny previous studies that handle problems of how to solve policies with a small number of data elements. We therefore fix the purpose of this thesis to propose a methodology of creating small and efficient reflexive policies. In other word, we try creating reflexive policies whose **efficiency per bit** are superior.

In Chapter 2, look-up tables called **state-action maps** are defined as one of the most suitable formats for reflexive policies. It is defined as a binary sequence that records a suitable action for every state of robots and their surroundings. When the state is represented by some variables (*state variables*), a state-action map can also be regarded as a multidimensional array on the state space that are spanned by the state variables. As a method for building state-action maps, the value iteration algorithm, which is one of the most popular algorithms of DP, is explained. Simple use of look-up tables for solving decision making problems is regarded as an inefficient and nonresponsive way. This criticism is partly true. We however show some actual examples that the efficiency per bit of policies on a look-up table is sometimes more efficient than that of policies with tricky representation. From this chapter to Chapter 4, we explain and evaluate algorithms with the puddle world task, which is a standard problem of artificial intelligence.

In Chapter 3, we propose the method for compressing the state-action maps. Vector quantization (VQ) algorithms are utilized for compression.

The compressed state-action maps are called **vector quantized state-action maps (VQ maps)**. For compression of state-action maps by VQ, we propose a novel **distortion measure**, which is called the **state-value distortion**. VQ has been successfully used for lossy compression of digital images and sounds. In a VQ method, the loss of information of compressed data is measured by a distortion measure which can be easily defined as difference of values on each pixel or each element of wave. On the other hand, the definition of a distortion measure is much more difficult because a change of an action on a state-action map sometimes influences quite different areas of the state-action maps. Therefore, compression of state-action maps is also difficult. Robots sometimes cannot reach states where tasks are completed due to only one change. The state-value distortion can reduce the risk of such a destruction of state-action maps.

The VQ method for state-action maps and the state-value distortion are evaluated on the puddle world task. The method is also compared to a compression method that utilizes a tree-structure.

In Chapter 4, we propose some methods that can enhance the efficiency per bit of VQ maps. **Value iteration methods for VQ maps** and **vector quantization of VQ maps** are applied to VQ maps of the puddle world task. We verify the effectiveness of the methods by the comparison to the simulation results in Chapter 3. Other techniques as partitioning, methods for finiding suitable ways of creating vectors are also introduced.

In Chapter 5, we try creating VQ maps for the height task of **the Acrobot**. The Acrobot is an underactuated manipulator that has one actuator and two links. The task of the Acrobot has different properties from the puddle world task. Especially, the Acrobot's nonlinear dynamics makes the problem difficult. However, we can show that the VQ method can be applied to it with the common formulation of Markov decision processes. By a simulation for evaluation, we find that the VQ method can reduce the dimension of a state-action map from four to two even though the obtained state-action map is chaotic.

In Chapter 6, the VQ method is applied to two kinds of task on robot soccer (**RoboCup**), which is nowadays the most popular standard problem of robotics and artificial intelligence. In the first task, the VQ method is applied to an actual robot, which is a quadruped robot ERS-210 made by SONY, for evaluation experiment. The task is to approach a ball within minimum steps. In this experiment, we verify that a VQ map whose size

is 1.5[%] of an original state-action map can work without conspicuous efficiency loss.

In the latter part of this chapter, a task in which two ERS-210s try scoring within minimum time is handled in simulation. Multi-agent systems are challenging problems not only for the VQ method but also for DP. By our discretization of the state-space, the nunber of elements in the state-action map reaches 610 million. Firstly, we demonstrate that such a large problem can be solved by DP with a high-performance desktop computer. Secondly, we solve the problem that the state-action map cannot be installed on an ERS-210 due to the shotage of memory. The VQ method is used to reduce the size of the state-action map to a smaller size than the amount of memory (16[MB]) on an ERS-210. The efficiency of the created VQ map is evaluated in simulation. The flow of the process of DP and VQ in this task will suggest that our methodology can give an efficient, elaborate, and reflexive policies.

In Chapter 7, we summarize and discuss each proposed method and concept in the thesis. From the results of simulation and experiment on every task, the ability of the VQ method is evaluated. The versatility of the state-value distortion is revealed.

In Chapter 8, we conclude this thesis. After that, some possible extensions of our method are indicated.

Contents

Chapter 1

Introduction

This thesis devotes all of its pages to deal with a problem of **how to implement software controller for a robot with a small amount of random access memory (RAM) on its computer**. Though this problem would not sound up-to-date, it would be true that many researchers have considered the problem noted above under the bright key-words intentionally or unintentionally. In other words, the problem is still important subject and they try to solve it by using various techniques or approaches.

In this chapter, we make preparations for explaining our methodology. In Sec. 1.1, some problems of optimal control, artificial intelligence, and robotics are introduced. It is helpful for readers to realize characterization of the subject of this thesis in these fields. In Sec. 1.2, a problem of policy implementation is defined. Related studies around the problem are explained in Sec. 1.3. We set the purpose of this thesis in Sec. 1.4. The structure of this thesis is stated in Sec. 1.5.

1.1 Background

1.1.1 Multistage Decision Problems on Robotics

In the course of day-to-day activities, we encounter a lot of cases where we decide our to-do list so as to fulfill some purposes. If we give an example, a marathon runner must vary his/her pace in order to mark his/her best time. The result of the variation of pacing is evaluated when the runner arrives at the goal. Some kinds of board game are more complicated. Players of chess, for example, are thinking sequences of their moves in the game. After dozens of moves, the sequence of moves are evaluated. One of them becomes the winner and another becomes the loser. Literally speaking, the loser cannot be obtained any reward even though he/she has thought a sequence of moves seriously.

As the above examples, there are many cases where not a decision but a sequence of decisions at every stage is evaluated at the end. These problems are called optimal control problems or multistage decision problems. In the above examples, the former can be regarded as an optimal control problem for minimizing the time. The latter is a typical multistage decision problem in artificial intelligence.

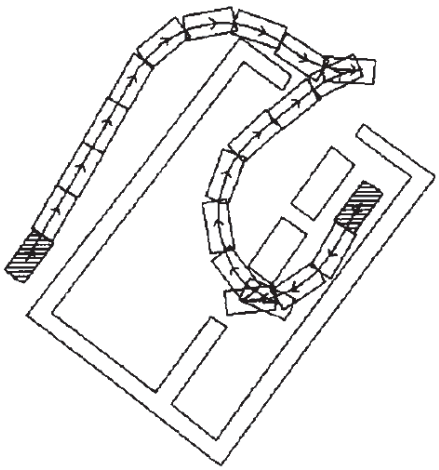


Fig. 1.1: Motion planning of a non-holonomic vehicle [Laumond, 1994]

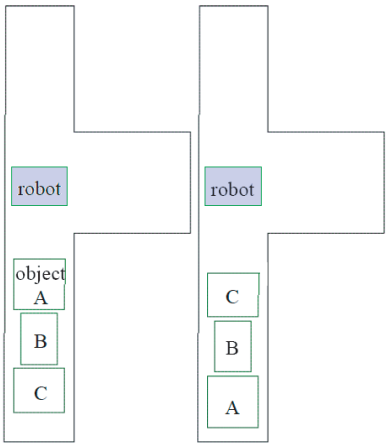


Fig. 1.2: A rearrangement task [Ota, 2004]

In this thesis, multistage decision processes and optimal control problems are argued for robots. In robotics, decision making problems have both

characteristics of control engineering and artificial intelligence. Every robot has its body, which is ruled by the physical law. On the one hand, it should compose a procedure of a task if the task is not finished without following the procedure.

Figure 1.1 and 1.2 illustrate two research subjects in robotics: control of a nonholonomic robot [Laumond, 1994; Ferbach, 1998], and a rearrangement task of movable objects [Fukazawa, 2003; Ota, 2004]. The former is rather a control problem. The motion of the nonholonomic robot is composed of smooth trajectories. Each trajectory is determined according to the physical characteristic of the robot. However, we should also pay attention to the cusps that link the trajectories. When watching the cusps, we think that this path will not be solved by classical methods for control problems.

On the other hand, the rearrangement task seems to be an issue of AI. There are three objects: A, B, and C in the "⊢" shape environment. These objects have to be rearranged from the state (a) to the state (b). When a person does this task instead of the robot, he/she thinks where each object should be temporally put at first. This process is not control but a way of AI. However, the objects and the robot are suffered from various limitations on rigid dynamics.

It is easy-to-understand for us to regard a process for completing the task as a sequence of *states and actions*. This sequence is written as

$$s_0, a(t_0), s(t_1), a(t_1), s(t_2), a(t_2), \ldots, a(t_{T-1}), s_{\mathrm{f}} \quad (s_0 = s(t_0), s_{\mathrm{f}} = s(t_T)). \tag{1.1}$$

Each symbol is used with the following definitions:

- $t = t_0, t_1, \ldots, t_T$: a sequence of time instants (no need to be fixed),
- $s(t)$: state of the task at time instant t, and
- $a(t)$: action which is executed by the robot at t.

t_0 and t_T denote time instants at start and finish of the task respectively. s_0 and s_T are named the initial state and the final state respectively. A state contains all kinds of information that relate to the task. An action then can be regarded as the operation of the robot to the state. We assume here that the robot does not need to observe states. We only want to describe how a

robot behaves to complete a task.

We can settle the symbolized states with an arbitrary step size. We can give the symbol s to every state where the robot stops at a cusp in the case of Fig. 1.1. In the case of the task in Fig. 1.2, several halfway arrangements of objects can be given the symbols. Since the aspect of dynamics is concealed behind the symbols in this case, every action $a(t_i)$ is symbolization of the answer of the control problem from $s(t_i)$ to $s(t_{i+1})$. The merit of this idea is that decision in a task can be separated to a control problem and an AI problem.

On the other hand, we can also symbolize the states with a minimum step. The step is determined by cycle time of computation for decision making. In this case, the action is more primitive than that of the above case. For example, it can be input current for actuators. With this representation, behavior of a robot in a task is handled as just a result of control of actuators. This representation is more general than the former one. If every action in the former is decomposed into the sequence of input current for actuators, and if the state at every time instant of input is observed, the state-action sequence has a structure with the latter representation.

1.1.2 State-Action Controller

In either case, behavior of a robot can be represented by the state-action sequence. This idea is frequently utilized to implement controllers of robots, or machines when they can recognize the state at every time instant. To explain those controller, we define the symbols of state and action more definitely.

- States are symbolized in the state-action sequence when they are observed or estimated.
- One symbol of action is given to the motion of actuators between an observation/estimation and the next one.

The controller that we argue above has the following property.

- An action is chosen reflexively when the state is observed/estimated.

We can symbolize the state-action sequence with this controller as:

$$s_0, \pi(s_0), s(t_1), \pi(s(t_1)), s(t_2), \pi(s(t_2)), \ldots, \pi(s(t_{T-1})), s_{\mathrm{f}} \tag{1.2}$$

if the task is successfully finished. π denotes a mapping from a state to an action.

$$\pi : \mathcal{S} \rightarrow \mathcal{A} \tag{1.3}$$

where $\mathcal{S} = \{s(t)|t = t_0, t_1, \ldots, t_{T-1}\}$ and $\mathcal{A} = \{a(t)|t = t_0, t_1, \ldots, t_{T-1}\}$. The set of states $\mathcal{S}$ does not contain identical states. We such a case can be avoided if the time is included as the state. π is called a policy. Here we call this controller a state-action controller since every action is chosen based on the state.

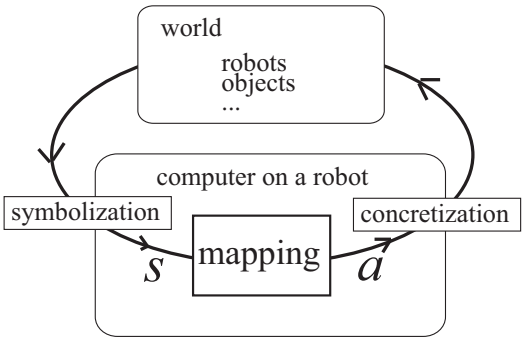


Fig. 1.3: A state-action controller

Figure 1.3 illustrates relation between a state-action controller and the world. We assume that a robot, machine, or system has a computer, which is a von Neumann-type one. It also has sensors and actuators. This computer converts the sensor readings into $s \in \mathcal{S}$, s into $\pi(s) = a \in \mathcal{A}$, and a into motion of actuators.

Sequential controllers can be the simplest examples of state-action controller. Robots and machines with sequential controllers have been used for manufacturing in factories. State of a manufacturing line is recognized through mechanical switches. When some states of switches are changed, inputs for actuators are converted by a controller. The motion of machines changes from the state to another. The mapping from a state to an action is programmed by engineers.

We can utilize a state-action controller for autonomous robots. A policy can be defined based on a state-action sequence that is planned by a programmer. For example, LEGO Mindstorm [Baum, 2000] is a popular

commercial product for building autonomous robots and vehicles. Figure 1.4 illustrates a vehicle composed of LEGO blocks and a computer block that is called RCX. A programming environment, which is called RCX Code, is attached to this product. We can program a sequential controller with RCX Code on a computer and send the executable code to the RCX. In the box of Mindstorm, paper on which a track is drawn is also attached as shown in the figure. Many people who buy Mindstorm may build up an autonomous vehicle and try programming a code for the vehicle for tracing the track at first.

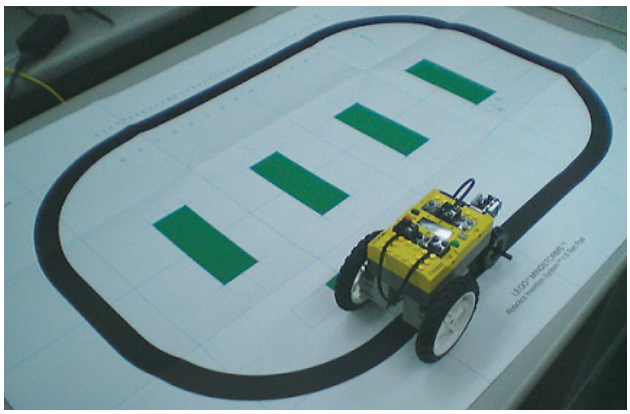


Fig. 1.4: A LEGO Vehicle

It is not easy to create a program that can make a vehicle trace the track hundreds of laps without failure, while manufacturing lines can make thousands of products in serial order. The reason is not that their programming for LEGO robots is unserious. In the case of manufacturing lines, various factors that disturb state-action sequences are removed. On the other hand, it seems that we expect LEGO robots to be robust against factors of disturbance. It is then impossible to program motion of robots with a perfect prediction of all of the factors.

1.1.3 Mapping from State to Action

Thus, robustness toward the disturbance is very important when we want to make an autonomous robot work beyond environments of factories. In control engineering, a controller is regarded as robust if it can pull back a state in the state-action sequence when a perturbation is given. In the case of the robots mentioned above, however, it is not enough because envisioned disturbances are much larger than perturbations assumed in

control engineering.

Moreover, an autonomous robot must build the state-action sequence by itself when the initial state is not told beforehand. Paradoxically speaking, we can judge whether a robot is autonomous or not from this ability. The robot in Fig. 1.2, for example, should arrange the objects even if the initial arrangement is not defined previously.

The robustness toward disturbances is included in this function. When a state-action sequence is cut by an accident, the state after that can be regarded as an initial state. The robot must compose an appropriate sequence from the new initial state to a final state. If we want to realize such a robot, every and each state should be able to be an initial state.

We define the following state-action controller as follows so as to consider robots that have the above function. A state space and a set of actions are defined as

$$\mathcal{S} = \{s_i | i = 0, 1, 2, \ldots, N-1\}, \text{ and} \tag{1.4}$$
$$\mathcal{A} = \{a_j | j = 0, 1, 2, \ldots, M-1\} \tag{1.5}$$

respectively. We assume that each and every condition of an environment belongs to a state in $\mathcal{S}$. Some of the states in $\mathcal{S}$ are final states, whose set is written as $\mathcal{S}_{\rm f}$. The state is changed from one to another by an action. Since the state-transition is not always deterministic, we represent each state transition as $\mathcal{P}^a_{ss'} \in [0, 1]$ toward all of the set of $s \in \mathcal{S} - \mathcal{S}_{\rm f}$, $a \in \mathcal{A}$, and $s' \in \mathcal{S}$. On the definition, a policy is defined as

$$\pi : \mathcal{S} - \mathcal{S}_{\rm f} \to \mathcal{A}. \tag{1.6}$$

This policy is effective all over the state space, while the policy in Eq. (1.3) is a piece of string in the space.

The robot with this policy starts its task as soon as the robot is put at an arbitrary state; as if it is the instinct of the robot. The robot continues the task from the posterior state even if the robot drops something, skids somewhere, or is obstructed by someone. Even if this idea is an impractical proposition, to research entirely autonomous robots means to realize such a robot.

1.2 Policy Implementation Problem

Though it is difficult to imagine autonomous robots that have the robustness mentioned above, polices that can react all kinds of state in the world exist certainly. For example, a toy car with a motor and a sensor has a kind of policy. When the car has one sensor that outputs one or zero, the car can distinguish two states and react to each state. If it has an electrical circit for memory, moreover, the car distinguish more than two states with combinations of the sequence of binary. For the car or its policies, simply stated, state of the world is equal to the stimulation of the sensor. Even if the car does not know the variety of this world, it can be moving.

What is the difference between robots and the toy car? The important difference is that robots in research are given tasks. Their performance is evaluated by some ways of evaluation. The other difference is that people place excessively high hopes on robots, while there is no essential difference between their hardware. Even if a robot has a high performance computer, the entity of the computer is an electrical circuit.

From that pessimistic opinion, however, we can find a challenging problem of how to make bundles of electrical circuits with sensors and actuators behave intelligently. In this thesis, we want to give robust and extraordinarily-complex policies to robots beyond the limitation of their computers.

Three Evaluation Criteria of A Policy

Here we assume a robot that has a von Neumann computer. A policy is implemented on memory of the computer. The ability of a von Neumann computer is evaluated by its processing speed and its amount of memory. A policy on the computer is also evaluated by

- time complexity: calculation amount for accessing the policy, and
- space complexity: the amount of memory for its representation.

These criteria are called *the on-line costs* in this thesis. The computational costs should also be considered when a policy is created. The costs are called *the off-line costs*. Hereafter, we use *off-line* and *on-line* to represent the phase when a policy is computed and the phase when it is used for decision making respectively.

Another criterion is required for evaluating a policy. As mentioned above, a robot is given a task and evaluated. Therefore,

- efficiency of control: its optimality on an evaluation

is an important criterion. A policy on a robot can be evaluated by the above three criteria at on-line.

Our Scope in This Thesis

We can think various implementation problems of policies by the change of emphasis on each of the above criteria. In this thesis, we handle policies that make robots possible to decide its action reflexively. In such a policy, an action at an arbitrary state on a policy must be accessible only with some steps of four arithmetic operations or address calculations. We name such policies *reflexive policies*. A reflexive policy enables a robot to react the result of state recognition in real-time. For robots that have poor computers or micro computers, reflexive policies are inevitable. Autonomous robots and machines that must move very fast also require them.

On the other hand, the amount of memory and the efficiency will be sacrificed by the direct representation of policies. However, a policy cannot be larger than the amount of memory on the robot. For robots with poor computing resources, compact representation of reflexive polices is inevitable. Small size policies are also welcomed by other robots because there is a possibility that the robots can consider the more larger number of states with high efficiency beyond the sizes of their memory. Therefore, we handle a problem of how to implement reflexive polices as small as possible in this thesis.

Optimal Control Problem

Before that, we must define the efficiency of a policy. As mentioned above, the efficiency is measured based on an evaluation, which is represented by a scalar. If there is no evaluation, we cannot discuss which policy is better or worse.

The optimal control theory is the most general formulation for solving or evaluating policies with an evaluation. We consider the following equation:

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}[\boldsymbol{x}(t), \boldsymbol{u}(t)], \quad \boldsymbol{x}(0) = \boldsymbol{x}_0, \quad t \in [0, t_{\mathrm{f}}]. \tag{1.7}$$

This equation is called a state equation. $\boldsymbol{x}(t) = (x_1(t), x_2(t), \ldots, x_n(t)) \in \mathcal{X} \subset \Re^n$ is named a state vector. Each element in the vector is called a state variable. $\boldsymbol{u}(t) = (u_1(t), u_2(t), \ldots, u_m(t)) \in \mathcal{U} \subset \Re^m$ is a control input vector and its element is named a control parameter. We assume that the state vector $\boldsymbol{x}(t)$ is known toward $\forall t$. This state equation defines or models the dynamics of the system that is controlled. The state of the system $\boldsymbol{x}(t)$ changes according to control input $\boldsymbol{u}(t)$ with the low of dynamics: $\boldsymbol{f}$. Though time t_f is defined as the time at the end of control, it does not have to be fixed. Hereafter $\boldsymbol{x}(t)$ and $\boldsymbol{u}(t)$ are sometimes called a state and a control input respectively. $\boldsymbol{x}_0$ is called an initial state. If the state can be led from $\boldsymbol{x}_0$ to $\boldsymbol{x}_\mathrm{f}$ by the sequence of control inputs from $t = 0$ to $t = t_\mathrm{f}$, a path of the state in $\Re^n$ space is fixed as shown in Fig. 1.5.

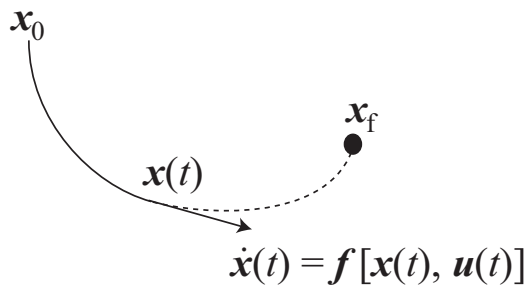


Fig. 1.5: A state transition of the system

Such a sequence of control inputs can be defined as a function of control input $\boldsymbol{u} : [0, t_\mathrm{f}) \to \Re^m$. If there is a purpose of control, this purpose should be mathematically defined. In an optimal control problem, the purpose is given as a functional $J[\boldsymbol{u}] \in \Re$, which should be maximized or minimized. In other word, all control inputs from $t = 0$ to $t = t_\mathrm{f}$ and their results are evaluated by a real number. The functional is the summation of an evaluation function:

$$g[\boldsymbol{x}(t), \boldsymbol{u}(t)] \in \Re \quad (t \in [0, t_\mathrm{f}]). \tag{1.8}$$

This function gives a real number toward the occasion at time t. Another evaluation, a value of a final state, is sometimes added to the functional. A final state is the state at t_f and written as $\boldsymbol{x}_\mathrm{f}$. The value of a final state is then represented as $V(\boldsymbol{x}_\mathrm{f})$. With the evaluation function and the value for the final state, the functional is written as

$$J[\boldsymbol{u}] = \int_0^{t_\mathrm{f}} g[\boldsymbol{x}(t), \boldsymbol{u}(t)]dt + V(\boldsymbol{x}_\mathrm{f}). \tag{1.9}$$

The function $\boldsymbol{u}$, which maximizes the functional, is called the optimal control. The function of the optimal control is written as $\boldsymbol{u}^*$. A problem to find the optimal control can be represented as

$$\max_{\boldsymbol{u}:[0,t_\mathrm{f})\to\Re^m} J[\boldsymbol{u};\boldsymbol{x}_0]. \tag{1.10}$$

When an initial state $\boldsymbol{x}_0$ and $\boldsymbol{u}^*$ are given, a path of the state transition is fixed. This path is called the optimal path and its function is written as $\boldsymbol{x}^* : [0, t_\mathrm{f}] \to \Re^n$.

When the optimal path is known, $\boldsymbol{u}^*$ can be a function from the state on the path to the optimal control input. We write this function

$$\boldsymbol{\pi}^* : \Re^n \to \Re^m \tag{1.11}$$

specially. We call $\boldsymbol{\pi}^*$ the optimal policy. However, this function can be defined only on the optimal path at this stage. In other words, this function depends to the initial state $\boldsymbol{x}_0$.

In the case of the time-invariant system that can be represented by Eq.(1.7), we can recognize that the optimal policy is independent of the initial state. Equation (1.9) can be rewritten as

$$\begin{aligned}\max_{\boldsymbol{u}:[0,t_\mathrm{f})\to\Re^m} J[\boldsymbol{u};\boldsymbol{x}_0] &= \max_{\boldsymbol{u}:[0,t')\to\Re^m} \int_0^{t'} g[\boldsymbol{x}(t),\boldsymbol{u}(t)]dt \\ &+ \max_{\boldsymbol{u}:[t',t_\mathrm{f})\to\Re^m} \int_{t'}^{t_\mathrm{f}} g[\boldsymbol{x}(t),\boldsymbol{u}(t)]dt + V(\boldsymbol{x}_\mathrm{f}) \\ &= \max_{\boldsymbol{u}:[0,t')\to\Re^m} \int_0^{t'} g[\boldsymbol{x}(t),\boldsymbol{u}(t)]dt + \max_{\boldsymbol{u}:[t',t_\mathrm{f})\to\Re^m} J[\boldsymbol{u};\boldsymbol{x}(t')].\end{aligned} \tag{1.12}$$

This equation is fulfilled with $\forall t' \in [0, t_\mathrm{f})$. As shown in this equation, the optimal control problem can be separated at $\forall t' \in [0, t_\mathrm{f})$. $\boldsymbol{u}^*(t')$ should be the solution of the problem in $[0, t_\mathrm{f})$, while it should be that in $[t', t_\mathrm{f})$. Moreover, the solutions should be equal to each other. It means that $\boldsymbol{u}^*(t')$ is independent of the initial state $\boldsymbol{x}_0$ and that $\boldsymbol{u}^*(t')$ is depended only on the state $\boldsymbol{x}(t')$. Therefore, the policy, which is the mapping from every state to a control input, can be defined.

When a policy $\boldsymbol{\pi}$, which does not need to be optimal, is given, the functional can be regarded as a function, $V^{\boldsymbol{\pi}} : \Re^n \to \Re$. Each value of the

function is defined as

$$V^{\boldsymbol{\pi}}(\boldsymbol{x}) = J[\boldsymbol{u}; \boldsymbol{x}], \quad \text{where } \boldsymbol{u}(t) = \boldsymbol{\pi}(\boldsymbol{x}(t)),\ 0 \leq t \leq t_{\mathrm{f}}. \tag{1.13}$$

This function is called a state-value function. This function can be regarded as an extension of the function that gives the value of the final state in Eq.(1.9). The state-value function for the optimal policy $\boldsymbol{\pi}^*$ is called the optimal state-value function and is written as V^*.

Finite Markov Decision Processes

Any robot exists in continuous space as the above definition of optimal control problems. On the other hands, a computer of the robot calculates everything with discrete approach. Therefore, the problem is sometimes discretized or quantized. Here we define the above discretization rigidly with the manner of finite Markov decision processes (finite MDPs) [Sutton, 1996].

At first, we quantize time as t_i $(i = 0, 1, 2, \ldots,)$. Though we do not need to fix the width of each time interval, we define it as the minimum cycle time of decision for purposes of illustration. The minimum cycle time will depend on the various hardwares of the robot. When $\boldsymbol{u} \in \mathcal{U}$ is chosen at t_i, the robot cannot change it until t_{i+1} comes. The control input changes a state $\boldsymbol{x}$ to another $\boldsymbol{x}'$ while the computer is preparing next decision. For the computer, control between a time interval is only a operator that changes states and its physical aspect is not important. Therefore, we define a set of actions $\mathcal{A} = \{a_0, a_1, \ldots, a_{M-1}\}$ in place of $\boldsymbol{u} \in \mathcal{U}$ for the operation of the computer. M is the number of actions that are considered.

The state space $\mathcal{X}$ is also discretized or quantized. A symbol of discrete state s and their set $\mathcal{S} = \{s_0, s_1 \ldots, s_{N-1}\}$ are prepared. N is the number of discrete state. Every state $\boldsymbol{x}$ in the physical space should be related to one of the discrete states on the computer. As defined in Sec. 1.1.3, some states in $\mathcal{S}$ belong to a set of final states $\mathcal{S}_{\mathrm{f}}$.

Though $a \in \mathcal{A}$ and $s \in \mathcal{S}$ are only symbols on the computer, every state transition from a state s to another state s' by an action a is dominated by the state equation of Eq. (1.7). Moreover, these state transitions become probabilistic due to the discretization of the state space whether the state equation is deterministic or stochastic. When a state $s(t_i)$ jumps to $s(t_{i+1})$

with an action at time t_i, which is written as $a(t_i)$, this dynamics is given by probabilities:

$$\mathcal{P}^a_{ss'} = P[s(t_{i+1}) = s' | s(t) = s, a(t) = a], \quad (\forall t \in \{t_0, t_1, \ldots, t_{T-1}\}, \forall s \in \mathcal{S} - \mathcal{S}_\mathrm{f}, \text{ and } \forall s' \in \mathcal{S}). \tag{1.14}$$

$\mathcal{P}^a_{ss'}$ are called state transition probabilities.

Toward the above set of s, a, s', then, an evaluation

$$\mathcal{R}^a_{ss'} \in \Re \tag{1.15}$$

is given. This value is called a reward. In the continuous system of the optimal control problem, this value relates to the integral of the evaluation function g at the time interval.

The purpose of the task is to maximize the following summation of values

$$J[a; s(t_0)] = J[a(0), a(1), \ldots, a(t_{T-1})] = \sum_{i=0}^{T-1} \mathcal{R}^{a(t_i)}_{s(t_i)s(t_{i+1})} + V(s(t_T)), \tag{1.16}$$

where $s(t_{T_\mathrm{f}}) \in \mathcal{S}_\mathrm{f}$. V is the value of a final state, which is also defined for the continuous system. The problem is represented as

$$\max J[a; s(t_0)]. \tag{1.17}$$

A policy is defined as Eq. (1.6), which is $\pi : \mathcal{S} - \mathcal{S}_\mathrm{f} \to \mathcal{A}$. As the same with the continuous system, we can assume the existence of an optimal policy π^* and the optimal state value function V^*.

Relation of Policy Implementation to Optimal Control Problem, and to Finite Markov Decision Process

By the definition of the optimal control problem, the efficiency of a policy can be defined rigidly as:

$$J^{\boldsymbol{\pi}} = \int_{\mathcal{X}} p(\boldsymbol{x}_0) J[\boldsymbol{u}; \boldsymbol{x}_0] d\boldsymbol{x}_0 \quad \Big(\boldsymbol{u}(t) = \boldsymbol{\pi}(\boldsymbol{x}(t))\Big), \tag{1.18}$$

where $p(\boldsymbol{x}_0)$ denotes a probability density that $\boldsymbol{x}$ is chosen as the initial state. If π is optimal, incidentally, the above equation is maximized independently of $p(\boldsymbol{x}_0)$. When an optimal control problem is solved as a finite MDP with an approximation, the policy π should be evaluated by the above equation.

π is replaced by $\boldsymbol{\pi}$ that represents the input-output relation in the actual world more directly.

Though the efficiency should be high, we also consider the amount of memory that is used for representing a policy. The size of a policy on memory can be measured by the number of bits whether the policy is continuous one: $\boldsymbol{\pi}$ or not: π. The relation of efficiency and the size can be compared on such a graph shown in Fig. 1.6. The horizontal axis and vertical axis indicate the size of a policy and the efficiency respectively. The axis of efficiency is reversed so that the origin of the graph makes the ideal point. The origin denotes a point of 0[bit] and the maximum value of $J^{\boldsymbol{\pi}}$.

In studies of minimum description length (MDL) principle [Rissanen, 1999; Barron, 1998], the vertical axis indicates the ability to discriminate. From this point of view, this graph suggests that this thesis handle a problem of minimum description of a policy.

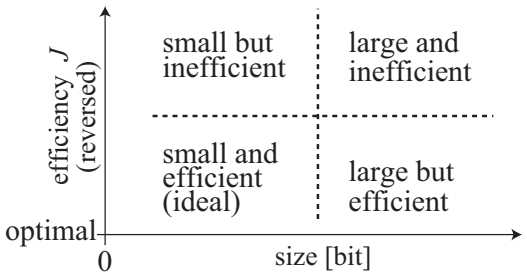


Fig. 1.6: Size-Efficiency Graph

In this thesis, we try bringing reflexive policies close to the origin. When there is a limitation of the amount of memory or a target amount of memory, we try maximizing the efficiency on the limitation or the target amount.

1.3 Related Studies and Works

In this section, important studies for us are referred to so that our concrete objective is fixed. We start from some methods for solving policies because a policy must be solved before it is implemented.

1.3.1 Fundamental Solutions of Optimal Control Problems

The conventional technique for finding a policy in a continuous system is calculus of variations [Ewing, 1985]. Here it is means the group of methods to solve the Hamilton-Jacobi-Bellman equation:

$$\frac{\partial V(\boldsymbol{x})}{\partial t} = \max_{\boldsymbol{u}\in\mathcal{U}} \left[g[\boldsymbol{x},\boldsymbol{u}] + \frac{\partial V(\boldsymbol{x})}{\partial \boldsymbol{x}} \boldsymbol{f}[\boldsymbol{x},\boldsymbol{u}] \right]. \tag{1.19}$$

In [Pierre, 1986], such methods are called classical calculus of variations. If the optimal control and the optimal path are obtained by calculus of variations, it is composed of several equations. In other words, the policy that is not represented by several equations cannot be obtained by these methods. When we see the above equation, the difficulty can be easily understood.

Dynamic Programming and Reinforcement Learning

Therefore, *direct methods* are applied to systems in robotics. This term is used for indicating the methods that solve optimal control problems not by formula manipulation but by numerical calculation on computers. Some kinds of continuous quantity are discretized or quantized in the methods. The finer the discretization or quantization is in a well implemented algorithm, the more the calculation result closes to that of the Hamilton-Jacobi-Bellman equation.

Dynamic programming (DP), which has been proposed by Bellman [Bellman, 1957], is a representative method of direct methods. Though DP is a classical method of the 1950's, it comes to be used for various applications [Hu, 1997; Ferbach, 1998; Fukase, 2002; Delft, 1994] in this ten years due to the remarkable progression of computers. If the dynamics of a system are known, DP can solve an optimal control for a purpose based on the discrete representation of the problem.

DP has been used for path planning of mobile robots [Fukase, 2002; Roy, 1999; Hu, 1997]. In their studies, the pose of this robot is represented

by two or three state variables: (x, y) or (x, y, θ). (x, y) denotes the position of a robot on a floor. θ is the orientation of the robot. In these cases, a system is approximated to a finite MDP, and DP is applied to. The numbers of states vary from some billion to some hundreds depending on the number of state variables and necessary granularity of discretization.

Researchers have dealt with the cases where the robot does not know the dynamics of the system on ahead [Takahashi, 1999; Ito, 2002; Kleiner, 2003; Tuyls, 2003; Buck, 2002]. Methods of reinforcement learning (RL) [Sutton, 1998; Watkins, 1992] are used for such problems. In RL, the dynamics and the policy are gradually obtained through the experience of a robot. We can regard as RL as a kind of DP that is applied to when the state equation in Eq. (1.7) or state transition probabilities $\mathcal{P}^a_{ss'}$ are unknown.

DP is sometimes compared to RL since RL can solve sevaral problems of DP. As mentioned above, RL methods can be appled to a problem without state transition probabilities $\mathcal{P}^a_{ss'}$. Even if $\mathcal{P}^a_{ss'}$ can be measured, difference between the actual system and measured $\mathcal{P}^a_{ss'}$ is inevitable. Therefore, DP is sometimes regarded as too conventional to be studied.

However, the ability of a reinforcement learning method is largely changed according to circumstances as shown in Table 1.1. Whether dynamics (state transition probabilities) are known or not influences the difficulty of a problem so much. Therefore, we should try to obtain the dynamics of a target of control once at least. In reinforcement learning, how to discretize state space is frequently discussed. That is because state is sometimes not changed after a state transition in trials for experience when the discretization is coarse. Some additional algorithms to deal with this problem are therefore required. If the discretization is too fine, on the other hand, much experience is required for convergence. If we want to obtain optimal policy for robots by a reinforcement learning method, moreover, we must watch for them until convergence. In the case of DP, the influence of coarseness is much less than reinforcement learning since DP regards state transitions as a stream of probability.

Potential Field Methods

A potential field method means to create a state-value function, $V : \mathcal{X} \to \Re$, with heuristics if we regard them as a solution of optimal control problems. From the viewpoint of optimal control, this method is quite rough. However, this method is a popular one in robotics [Khatib, 1986;

Table 1.1: Comparison of Dynamic Programming and Reinforcement Learning

	DP (value iteration algorithm) (with simple discretization)	RL (with simple discretization)
computation load	huge	small if complete convergence is not required
human labor	to measure accurate state transition probabilities	to help robots in learning
convergence of policy	high confidence	low confidence
range of policy	all over the state space	states in which much experience is obtained
flexibility of discretization	large	poor (depending on *width* of each state transition)

Latombe, 1991], especially for navigation task [Laue, 2004; Ge, 2000]. This method is frequently used when some moving obstacles exist in the environment. If states of all obstacles are considered in the state space, the optical control problem is annoyed by the curse of dimensionality. In the case of a potential field method, the shape of potential field can be instantly changed regardless of the number of state parameters.

A typical potential field method uses an attractive potential and repulsive potentials. When there is the only final state $\boldsymbol{x}_{\mathrm{f}}$ in the state space $\mathcal{X}$, an attractive potential, $U_{\mathrm{att}} : \mathcal{X} \to \Re$, is defined as a function whose global minimum exists at $\boldsymbol{x}_{\mathrm{f}}$. $\boldsymbol{x}_{\mathrm{f}}$ is the only stationary point of U_{att}. According to [Latombe, 1991], this function can be defined as a parabolic well, i.e.:

$$U_{\mathrm{att}}(\boldsymbol{x}) = \frac{1}{2}\xi\rho^2(\boldsymbol{x}) \tag{1.20}$$

where ξ is a positive scaling factor and $\rho(\boldsymbol{x})$ denotes a distance between $\boldsymbol{x}$ and $\boldsymbol{x}_{\mathrm{f}}$. Then the following is an example of repulsive potentials:

$$U_{\mathrm{rep}}(\boldsymbol{x}) = \begin{cases} \dfrac{1}{2}\eta\left(\dfrac{1}{\rho(\boldsymbol{x})} - \dfrac{1}{\rho_0}\right)^2 & \text{if } \rho(\boldsymbol{x}) \leq \rho_0, \\ 0 & \text{if } \rho(\boldsymbol{x}) > \rho_0, \end{cases} \tag{1.21}$$

where η is a positive scaling factor, $\rho(\boldsymbol{x})$ denotes the distance from $\boldsymbol{x}$ to an obstacle that is represented by a point in $\Re^n - \mathcal{X}$. The sum of the attractive potential and repulsive potentials of some obstacles:

$$U(\boldsymbol{x}) = U_{\mathrm{att}}(\boldsymbol{x}) + U_{\mathrm{rep}}(\boldsymbol{x}) \tag{1.22}$$

where U_{rep} is sum of the repulsive potentials of all obstacles. When the robot moves so as to reduce the value of U, the most appropriate direction can be obtained by

$$\begin{aligned} \boldsymbol{F}(\boldsymbol{x}) &= -(\partial U/\partial x_1, \partial U/\partial x_2, \ldots, \partial U/\partial x_n)^T \\ &= -\nabla U(\boldsymbol{x}). \end{aligned} \tag{1.23}$$

Potential field methods have some problems. Koren and Borenstein have summarized them in [Koren, 1991] as 1) trap situations due to local minima (cyclic behavior), 2) no passage between closely spaced obstacles, 3) oscillations in the presence of obstacles, and 4) oscillations in narrow passages. Ge and Cui have argued another problem that the minimum of U is off the goal state due to obstacles near the goal. Though many researchers have tried to solve these problems, an almighty method has never appeared. It seems that almighty methods will be DP-like methods.

1.3.2 Function Approximation Methods

We can utilize a variety of function approximation methods for representing a state-value function. Here we introduce some of the popular methods.

Any function approximator can be represented as

$$V(\boldsymbol{x}; \theta_1, \theta_2, \ldots, \theta_{N_\theta})$$

where $\boldsymbol{x}$ is a state and $\theta_1, \theta_2, \ldots, \theta_{N_\theta}$ are the parameters of this function. For solving a control problem, DP or RL must find the appropriate parameters that can approximate a hidden optimal state-value function. Function approximation cannot be used for representing a policy in the case where actions are discretized. In this case, an algorithm that chooses actions directly from the function.

There exist two types of function approximator: global one and local one. In the case of a global function approximator, a change of a parameter makes an impact all over the state space. Global methods have potential ability for solving the curse of dimensionality. On the other hand, each parameter of local one has its own area in the space. That means local function approximators require much more parameters than the global one. However, the more global an approximator is, the more instable it becomes when it is applied to DP or RL. The discussion of convergence is frequently discussed as shown in [Boyan, 1995; Sutton, 1996;

Sutton, 1998].

In practical use, a function approximator must be chosen carefully when a control problem is given because a function on an approximator does not converge or does not approximate the actual function well in many cases. Moreover, we must choose the number of parameters and other specifications. This operation is much difficult than adjustment of the granularity of discretization at the simple discretization methods.

Artificial neural network

Artificial neural networks (ANNs) have been used for function approximation or representing the direct relation between states to actions. There is a famous application of this method in the field of AI. Tesauro has applied an ANN to backgammon [Tesauro, 1995]. A multilayered perceptron is used for learning the ratio of victory or defeat from each state of the game. The ratio is the value of each state. This application is named TD-Gammon. It is a worthy opponent for human champions as well as Deep Blue. The core of TD-Gammon is an artificial neural network (ANN) that learns the game by using temporal difference (TD) learning methodology [Sutton, 1988], which belongs to reinforcement learning (RL). In the case of the version 3.1, though tens of thousands of training games is required for learning, it worked in an ordinary computer. On average, it requires only $10 - 12$[s] per move decision when it is running on a 400 MHz Pentium II processor.

As mentioned before, this approach has achieved a successful outcome. The reason why a multilayered perceptron is suitable for backgammon is derived from the fact that the random dice rolls produce a degree of variability of states. As a result, the state-value function becomes smooth and continuous, and becomes easy to be learned.

Tile Coding

In this technique, various kinds of discretization are applied to state space. When all of them are layered, pseudo cells, which are surrounded with some partition walls of the various kinds of discrete space, are created. The number of pseudo cells can be larger than that of cells on all kinds of discrete space. Figure 1.8 shows an example. Imagine that the two two-dimensional grids at the left side of this figure are piled up and one of them is shifted to right and upper direction slightly as shown in the right hand of the figure.

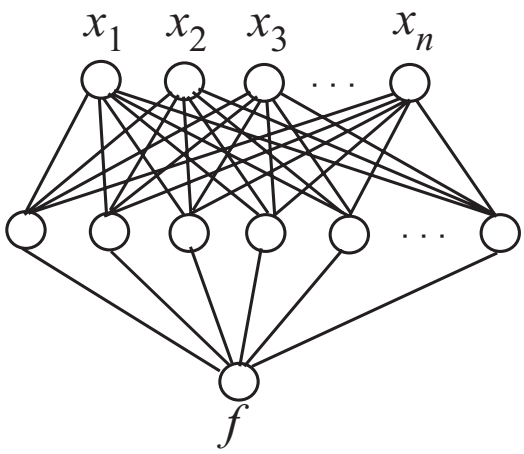


Fig. 1.7: A typical multilayered neural network for function approximation

The number of pseudo cells becomes two times larger than that of cells in the two grids.

In DP and RL methods, each layer records its state-value function. The value of a state in continuous state space is computed as the average of the values of discrete states that contain the state.

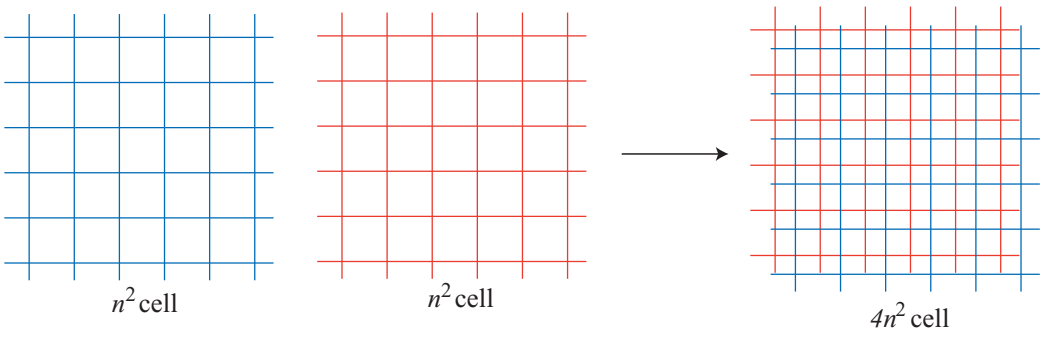


Fig. 1.8: Tile coding

The archetype of this technique has been tried by Albus [Albus, 1975a; Albus, 1975b]. He named the archetype a cerebellar model articulation controller (CMAC). Tile coding is another name of this technique given by Sutton [Sutton, 1996].

Radial Basis Function Approach

The methods [Broomhead, 1988; Moody, 1989; Samejima, 1999] that belong to this category scatter Gaussian radial basis functions, and approximate a state-value function with their summation. A radial basis function (RBF) for state space $\mathcal{X} \subset \Re^n$ is represented as

$$\phi_i(\boldsymbol{x}) = \exp\left\{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{c}_i)^t M_i(\boldsymbol{x}-\boldsymbol{c}_i)\right\}, \tag{1.24}$$

where M_i is a $n \times n$ scaling matrix and $\boldsymbol{c}_i \in \Re^n$ is the center vector of this function. $\boldsymbol{x}$ is a state vector defined in advance. In the case where the normalized Gaussian radial basis function [Moody, 1989] is used, this function is normalized as

$$b_i(\boldsymbol{x}) = \frac{\phi_i(\boldsymbol{x})}{\sum_{j=1}^{N_\phi}\phi_j(\boldsymbol{x})}, \ (N_\phi : \text{ number of RBFs in the space}) \tag{1.25}$$

and a state-value function is represented by

$$V(\boldsymbol{x}) = \sum_{i=1}^{N_\phi} \nu_i b_i(\boldsymbol{x}). \tag{1.26}$$

Figure 1.9 shows an example. There are five RBFs: $\phi_1, \phi_2, \ldots, \phi_5$, whose

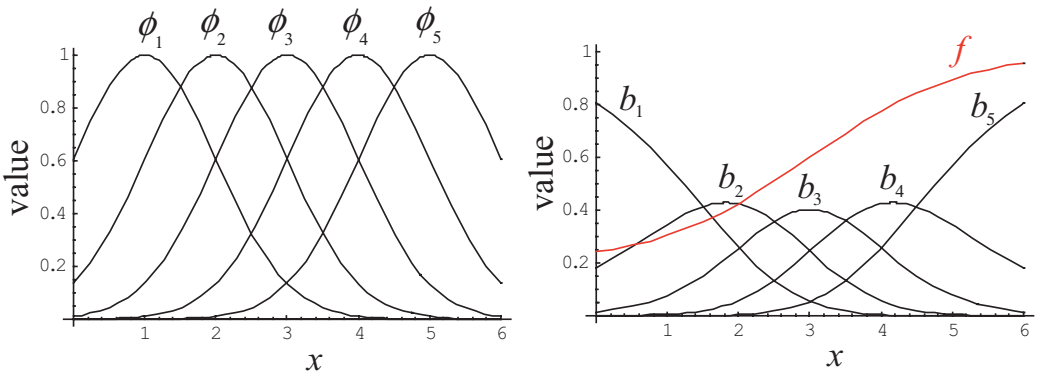


Fig. 1.9: Function Approximation by Using Radial Basis Functions

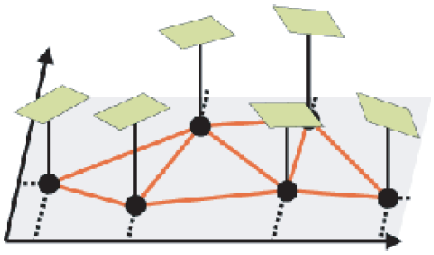
equation is

$$\phi_i(x) = \exp\left\{-\frac{1}{2}(x-i)^2\right\}$$

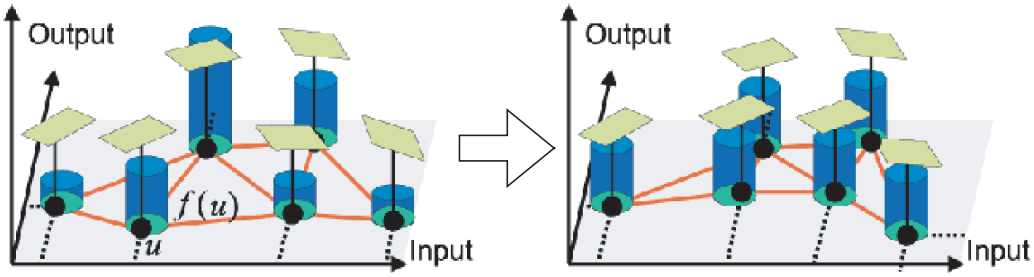
in the left figure. $b_1, b_2, \ldots, b_5$ in the right figure are their normalized functions respectively. A function is represented with the weighted sum of the normalized functions by Eq. 1.26. Function f shown in the right figure is an example with $\nu_i = i/5$.

Reaction-Diffusion Equation on a Graph

Kobayashi and Yuasa *et al.* have proposed to utilize reaction-diffusion equation on a graph for reinforcement learning [Kobayashi, 2002]. In this method, a function is represented by some elements that are connected by arcs as shown in Fig. 1.10(a). Parameters of each element are its position of state space, the value of the function at the position, and the gradient of the function around the position. The position is a black point in Fig. 1.10(a). The two dimensional coordinate system is regarded as state space, which does not have to be two dimensional. The height of the rod that stands from each position represents the value. The gradient of the panel that is top of each rod signifies the gradient.



(a) elements for function approximation and their connection



(b) evaluation of complexity of the function and modification of the elements

Fig. 1.10: Function Approximation by Reaction-Diffusion Equation on a Graph [Kobayashi, 2002]

In this method, the ability of function approximation depends on how

to distribute the elements. The more a part of the function has change of gradient, the more the elements should be allocated to the part. Information of the requirement of elements is exchanged on the graph, and a scalar function that represents the overs and shorts of elements is composed on the graph. Each elements move so that the scalar function becomes flat. Figure 1.10(b) shows an example of the move of elements. The height of a cylinder at each element represents the value of the function.

The objective of the study is parallel computation by distributed autonomous systems. As well as artificial neural networks, it is not always suitable for a von Neumann-type computer.

Interpolation

Takahashi *et al.* have used interpolation for reinforcement learning [Takahashi, 2001] so as to accelerate learning speed. In this method, representative states are placed in a reticular pattern as shown in Fig. 1.11. The value of state $\boldsymbol{x} \in \mathcal{X}$ is calculated as the weighed mean of the values of the representative states around $\boldsymbol{x}$. When the weight and the value of a representative state are w_i and x_i respectively, the value is calculated as

$$V(\boldsymbol{x}) = \sum_{i=0}^{3} w_i V(\boldsymbol{x}_i) \tag{1.27}$$

in the case of two dimensional state space. A weight is in proportion to the size of the opposite space from each representative state as shown in the figure. In this study, the interpolation technique is used not only for values but also for control input vectors $\boldsymbol{u} \in \mathcal{U}$ so that a robot can decide torques of actuators with continuous values.

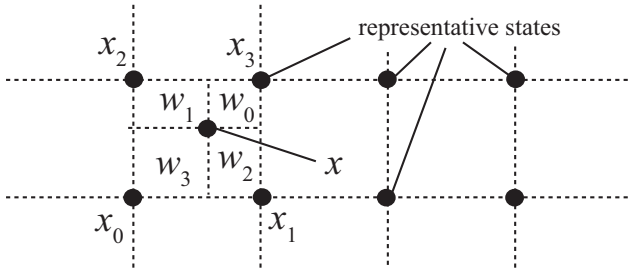


Fig. 1.11: Interpolation (An Example in Two Dimensional State Space)

Munos and Moore have presented more complex interpolation [Munos, 1998; Munos, 2002]. They use a tree structure shown in Fig. 1.12. They have indicate that this structure is useful not only reinforcement learning but also any method for solving optimal control problems.

As shown in Fig. (a), representative states are placed as they cut the state-space with different resolutions. The value of a state is computed from the values of three representative states that compose a triangle in which the state belongs to. Representative states are added to the space when some criteria indicate their necessity. They are registered on a tree structure shown in Fig. (b). A leaf has a cell (two triangles) that is not divided into small cells.

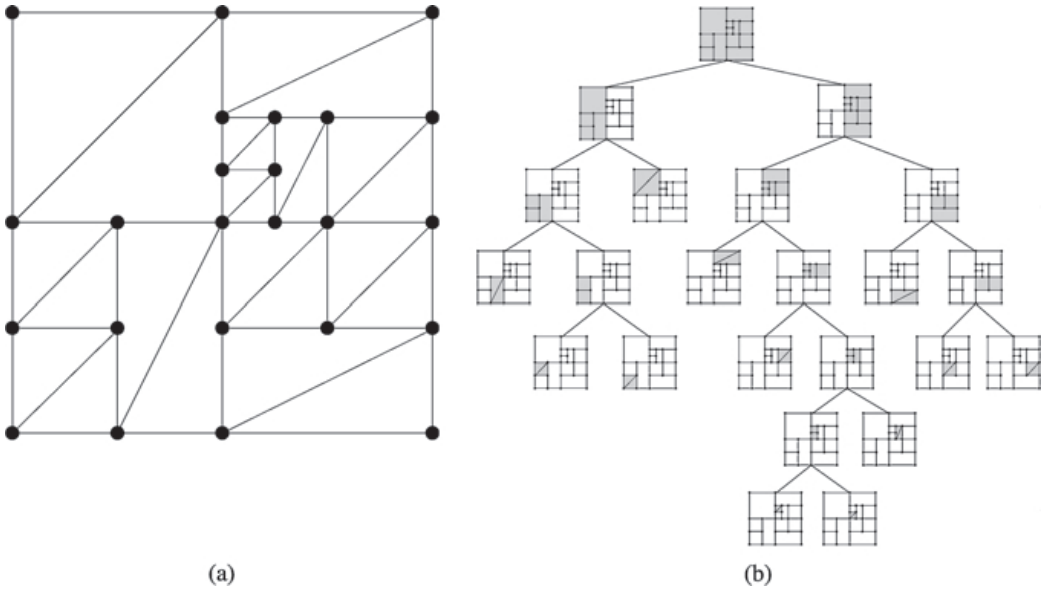


Fig. 1.12: Interpolation (Tree Structure) [Munos, 2002]

In the first part of [Munos, 2002], they have presented the following criteria of where representative states should be added.

- Edges of triangles in a cell have large gradient of values. The gradient of an edge means the difference of values of representative states at its end points.
- The values of representative states in a cell have non-linearity when each of them is compared to values of the other representative states

that are connected to by edges.

These criteria are suitable for representing a function with a small number of cells. However, as the authors have been mentioned, they never relate to represent a better policy.

Moreover, we must point out that the tree structure consumes a much larger amount of memory for representing the structure of branches than that for representing values at leaves. Actually, the ability of approximation has been evaluated not by the amount of memory but by the number of cells in the paper. From the viewpoint of memory consumption, well-arranged representative states as shown in [Takahashi, 2001] are sometime superior to some complicated arrangement.

1.3.3 Memory Economization for Policy Creation and Implementation

Studies of memory economization for policies are less than that of function approximation. In an optimal control problem or a finite MDP, a policy becomes known after we can obtain its state-value function, which consumes memory not less than the policy does. Therefore, memory economization of state-value functions is regarded more important than that of policies.

When we want to represent a policy with a compact format, however, a study of memory economization for polices is necessary. In this case, some techniques for function approximation sometimes yield redundant memory consumption.

Search Methods

If we must give some examples, search methods should be referred to. As a matter of course, they are not reflexive policies. Though these methods do not try solving state-value functions, they utilize some evaluation functions, which are heuristics in many cases, for finding appropriate state-action sequences.

Search methods have been studied in the field of artificial intelligence for board games as chess [Greer, 2000; Campbell, 2002], shogi [Iida, 2002], go [Bouzy, 2001; Müller, 2002] and so on. That is because the number of states is huge. According to [Iida, 2002], the numbers of states at chess, shogi, and go are 10^{43}, 10^{71}, and 10^{172} respectively. Since

the state space is discrete, search algorithms are suitable to the board games.

In the case of robotics, also, search methods are applied to a problem with many state variables. Since problems of robotics must be solved in continuous space, some states should be sampled from continuous state space (or configuration space [Latombe, 1991]) for using a search method. The sampled states are connected each other and search is done on the network. Such methods are called *sampling-based algorithms* [Choset, 2005]. Probabilistic roadmaps (PRMs) proposed by [Kavraki, 1996] and Rapidly-exploring Random Trees (RRTs) by [LaValle, 1999] are famous in these kinds of search method. They are frequently utilized for navigation of mazy environments [Choset, 2005], for motion planning of manipulators [Miyazawa, 2005].

As mentioned above, search methods are suitable to solve a problem in huge state space in which DP is useless. Alternatively, we should not expect that they can give an appropriate action instantly on a usual computer. We afford an instance of Deep Blue, which defeated a human champion of chess in 1997 [Campbell, 2002]. In this case, the policy is given as a search algorithm on a supercomputer. The computer is composed of 30 processors and 480 special LSI chips so as to search 100 million positions in feasible time.

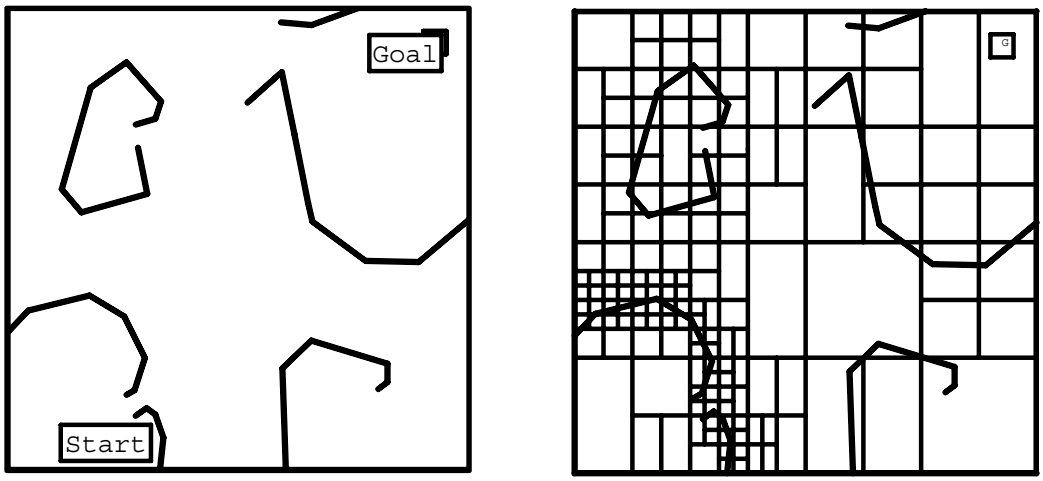


Fig. 1.13: Partitioning of State Space by the Parti-Game Algorithm
[Moore, 1995]

Parti-Game

The above search methods answer one or several state-action sequences. They do not aim to solve a policy completely in state space. On the other hand, the parti-game algorithm proposed by Moore and Atkeson in [Moore, 1995] can solve a policy though its coverage of state-space is not complete. This algorithm has been proposed for on-line reinforcement learning, and has been improved [Al-Ansari, 1998; Likhachev, 2002].

At the beginning of this algorithm, state space is divided into only two discrete states: the final state, and the other state. Then an action is allocated to the non-final state. However, in almost all cases, the agent cannot reach the final state from some areas in the non-final state. When the algorithm notices such an incomplete state, referred to as *a losing cell* associated with game theory in [Moore, 1995; Al-Ansari, 1998], it is divided into two states. By this means, the parti-game algorithm aims to keep the number of discrete states as small as possible.

As mentioned in [Moore, 1995], this algorithm does not attach importance to solving an optimal policy, but does it to finding a feasible path at on-line quickly. Actually, this algorithm is not good for computing state-value functions since this algorithm does not divide a not-losing state. Even if the values at some points in a large discrete state are different each other, these values are represented by one value of this state. This wild approximation gives negative effect to calculation of the values of neighboring states.

Interpolation with Tree-Structure based on Policy

In the center part of [Munos, 2002], which has been referred to with Fig. 1.12, a criterion for creating the tree structure for representing a policy accurately. In this criterion, a cell is divided into two when its representative states cannot give suitable control inputs at any state in the cell. According to them, this lack of resolution can be found by the comparison of the value obtained from the Hamilton-Jacobi-Bellman equation shown in Eq. (1.19) and the value obtained from the approximated value function.

1.3.4 Relation to The Policy Implementation

The previous methods have their merits respectively. A policy that is crated by one of these methods, however, never becomes a perfect policy, which means the most reflexive, the smallest, and the optimal policy toward an optimal control problem.

The methods then have been proposed for different purposes from ours. The policies obtained by the studies become large and inefficient due to each purpose. The most typical example is a group of studies to reduce the number of elements (e.g. number of parameters, number of RBFs, and so on) for representing a state-value function or a policy. This attempt aims to reducing learning or planning time with the generalization capability of each method. Some of the methods actually can reduce the time for creating a policy. In some of those studies, however, they have said that not only the time but also the amount of memory use can be reduced. However, we think that this proposition has never proved. That is because the relation between memory consumption and numbers of elements is unobvious. If an element uses a large number of bits, the size of the policy (or the state-value function) becomes large for the number of elements.

In the case of reinforcement learning, a robot must have a state-value function or another value function. Therefore, many function approximation methods have been proposed. Though a robot can decide its action from a state-value function with an evaluation function, the representation of functions is redundant when it is regarded as a policy. Moreover, the decision making is not reflexive.

Some methods are then specialized in particular control problems or decision making problems. For example, some studies have been tried getting rid of the curse of dimensionality when a policy is solved. Those studies are very interesting since it is the ultimate theme of AI. However, those attempts have never been achieved unless some characteristics of each control or decision making problem should be considered.

From the above discussion, we must propose another methodology so as to concentrate the small and efficient representation of policies. For versatility of the methodology, moreover, we should handle each problem as an optimal control problem or a finite MDP without consideration of its characters. As mentioned in Sec. 1.2, robots that have poor computing resources or that work in real-time are handled in this thesis. For this purpose, we think that

the creation problem and the implementation problem of a policy should be divided in a definite manner. The creation of a policy, or calculation of a state-value function should be done on a high-performance computer. After that, a policy should be processed into a small one. This methodology can be applied only to the case where the dynamics of the system is known. Since we handle a versatile method, the curse of dimensionality is not solved. However, a study to quest the small and efficient representation of policies will give important knowledge for the studies of learning or high dimensional decision making problems.

1.4 Purpose of This Study

We therefore introduce a novel concept: **compression of policies** in this thesis. The compression means the secondary processing for a policy to reduce its number of bits after it is obtained. Since high compression ratios cannot be obtained by a lossless compression method, we try applying a lossy compression method. In this process, the efficiency of the policy should be kept as high as possible.

In this thesis, a policy is solved by dynamic programming with simple discretization of state-space for fear that the performance decrement occurs due to memory economization. The policy is then compressed. There is the problem of performance decrement in the process of compression. Differently from other methods, however, the compression method can use the dynamic programming result, which is free of the influence of economization, as the information for cutting wasteful use of memory. It will be a great advantage for this approach.

We develop a novel lossy compression method for policies. This method inherits the techniques of vector quantization (VQ) [Gersho, 1992]. VQ is an universal method for compressing digital data of image [Tsai, 2000; Fekri, 2000] and sound [Huang, 2002]. This method can be applied to any binary data. Moreover, random access data can be crated by vector quantization. Therefore, it is suitable for creating reflexive policies.

However, VQ cannot be used for compressing policies without modification. Policies are too easily broken after compression due to some problems. To deal with the problems, we add the novel methodology:

- **reuse of dynamic programming result for compression**.

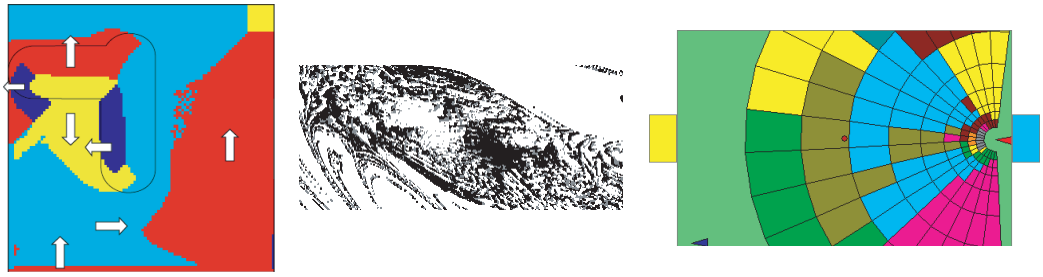
It means that our VQ method is based not only on the theory of information processing, but also on that of optimal control and decision making. From the above basis, we propose a novel definition of distortion measure, which is used in almost all of VQ methods for evaluating loss of information by compression. Our distortion measure is called **state-value distortion**. The state-value distortion is used in compression for measuring and reducing the loss of efficiency of policies. Moreover, some DP algorithms that re-optimize compressed policies are proposed in this thesis. These algorithms are also based on the above concept. We also introduce several methods and algorithms for enhancing compression ratio.

1.5 Contents of This Thesis

This thesis is composed of 8 chapters.

Chapter 2, 3, and 4 are devoted to explanation and proposition of methods and algorithms. Their evaluation is also done with the puddle world task [Sutton, 1996]. This task is a standard problem of artificial intelligence. In Chapter 2, a format of policies that are compressed is fixed. The policies are called state-action maps. We then explain how to create, use, and evaluate. Our novel algorithm for compressing state-action maps is presented in Chapter 3. The method is evaluated. In Chapter 4, some techniques to enhance *efficiency per bit* of policies are proposed and evaluated.

In Chapter 5 and 6, the proposed method is applied to two kinds of robots. Though most experiments in these chapters are simulation with a few exceptions, we can obtain very significant results. The Acrobot [Spong, 1994; Boone, 1997; Xin, 2004] is controlled by the method in Chapter 5. We take up tasks of RoboCup [Asada, 1999; Fujita, 2003] in Chapter 6.



(a) puddle world task

(b) the Acrobot
(a part of a 4D map)

(c) RoboCup
(a part of an 8D map)

Fig. 1.14: Part of State-Action Map for Each Task

In advance of detailed description, we show a (part of) state-action map for each task in Fig. 1.14. Each color is related to an action. In the case of puddle world task and RoboCup, the same action is located in adjacent states. On the other hand, the state-action map has more tangled pattern. That difference is convenient for verifying versatility of the proposed method. As another important difference, numbers of states and actions vary greatly.

In Chapter 7, we comprehensively discuss the proposed methods in consideration of the discussion and experimental results in the previous chapters. The state-value distortion is evaluated with three tasks. We conclude this thesis in Chapter 8 with our vision about expansibility of our algorithm.

Chapter 2

State-Action Map

In this chapter, a format of policies, state-action maps, is defined, and how to create them by dynamic programming (DP) is explained. State-action maps that are introduced in this chapter are simple look-up tables that record an appropriate action toward every state. They say that a state-action map consumes huge memory and it is too conventional to be studied. However, data can be accessed from it quickly in computation thanks to its affinity with von Neumann-type computers. Moreover, DP with look-up tables is one of the most certain methods to solve an optimal control problem.

This chapter is composed of five sections. In Sec. 2.1, the state-action map is defined. Algorithms of dynamic programming are explained is 2.2. In Sec. 2.3, we show an example of creation of state-action maps on the puddle world task, which is a popular standard problem of artificial intelligence. Memory consumption and efficiency of the state-action maps are evaluated. In Sec. 2.4, the evaluated results are compared to some methods of function approximation techniques so that we improve their memory usage is not wasteful. We conclude this chapter in Sec. 2.5 with discussion.

2.1 State-Action Maps

2.1.1 Its Format

In this section, a format of policies, state-action maps, is defined. Assume that a computer of a robot has a random access memory (RAM). Each bit of the memory is given its address. Addresses are numbered from zero to {(the number of bits)−1} on the RAM. A policy, $\pi : \mathcal{S} \to \mathcal{A}$, is given a first address and it is recorded from the bit that has the address without a jump. We assume that the first address is zero for simplicity. Discrete states in $\mathcal{S}$ are numbered from zero to $N-1$. Actions in $\mathcal{A}$ are also given numbers from zero to $M-1$.

Assume that indexes of actions, $\pi(s_0), \pi(s_1), \ldots$, and $\pi(s_{N-1})$, are put in from the bit that has the zero address without any delimiter as show in Fig. 2.1. This binary sequence is called **a state-action map** (or a map) in this thesis. Arbitrary binaries can be put in the bits for final states.

$$\underbrace{011}_{\pi(s_0)}\underbrace{010}_{\pi(s_1)}\underbrace{111}_{\pi(s_2)}\underbrace{010}_{\pi(s_3)}\cdots\underbrace{110}_{\pi(s_{N-1})}$$

Fig. 2.1: An example of a state-action map

All operations on a state-action map are manipulation of indexes. We define the following index sets:

- set of indexes for addresses: $\mathcal{I}_{\text{address}} = \{0, 1, 2, \ldots, B-1\}$,
- set of indexes for states: $\mathcal{I}_{\mathcal{S}} = \{0, 1, 2, \ldots, N-1\}$, and
- set of indexes for actions: $\mathcal{I}_{\mathcal{A}} = \{0, 1, 2, \ldots, M-1\}$.

In mathematical formulae, an arbitrary state-action map is defined as a mapping:

$$\pi_{\text{MAP}} : \mathcal{I}_{\mathcal{S}} \to \mathcal{I}_{\mathcal{A}}. \tag{2.1}$$

Each action index $j \in \mathcal{I}_{\mathcal{A}}$ is recorded on $\lceil \log_2 M \rceil$[bit] of memory. $\lceil \cdot \rceil$ denotes the ceiling function, which rounds up a real number. In the case

of Fig. 2.1, eight kinds of actions can be recorded because three bits are allocated to a state.

If we consider only the process for changing $i \in \mathcal{I}_{\mathcal{S}}$ to $j \in \mathcal{I}_{\mathcal{A}}$, the space complexity for using a state-action map is evaluated based on its memory usage:

$$L = \lceil \log_2 M \rceil N \text{ [bit]}. \tag{2.2}$$

That is because every state uses $\lceil \log_2 M \rceil$ bits for recording an action index. If $L > B$, the state-action map cannot be recorded on the memory. The time complexity for fetching an action can be estimated from the number of operations for $\mathcal{I}_{\mathcal{S}} \to \mathcal{I}_{\mathcal{A}}$. When we compare the time complexities of other formats of policy, a state-action map can be regarded as a reflexive policy.

The coarser the discretization of $\mathcal{X}$, the more a state-action map will decrease its performance. Here we formulate the loss of performance.

When a state-action map π_{MAP} is obtained toward a control problem in continuous state space $\mathcal{X}$, its ability can be measured by Eq. (1.18). Eq. (1.18) can be rewritten as

$$J^{\pi_{\text{MAP}}} = \int_{\mathcal{X}} p(\boldsymbol{x}_0) V^{\pi_{\text{MAP}}}(\boldsymbol{x}_0) d\boldsymbol{x}_0, \tag{2.3}$$

where $V^{\pi_{\text{MAP}}}(\boldsymbol{x})$ is a state-value function of π_{MAP} on $\mathcal{X}$. Note that $V^{\pi_{\text{MAP}}}(\boldsymbol{x})$ is different from the state-value function of π_{MAP} on $\mathcal{S}$. This function should be estimated from actual use of π_{MAP}.

When we assume that the optimal policy π^* on $\mathcal{X}$ is known, the loss can be calculated as

$$\Delta J^{\pi_{\text{MAP}}} = J^{\pi^*} - J^{\pi_{\text{MAP}}}. \tag{2.4}$$

This value is the efficiency loss of π_{MAP}. If π_{MAP} is the optimal policy under the discretization, this value can be regarded as the loss of efficiency by the discretization. As for the Bellman's principle of optimality, incidentally, $\Delta J^{\pi_{\text{MAP}}} \geq 0$ is fulfilled to any probability density function p in the above equation.

2.1.2 Association between Physical Space and State-Action Map

When a policy in continuous state space is recorded as a state-action map, the state space $\mathcal{X}$ is discretized into $\mathcal{S}$. Actually a mapping from a state to a state index

$$^{\mathcal{I}_\mathcal{S}}\mathcal{T}_\mathcal{X} : \mathcal{X} \to \mathcal{I}_\mathcal{S}. \tag{2.5}$$

is required for accessing a state-action map. This mapping actually defines the way of discretization. We should pay attention to the format of this mapping. If it is too complicated, computational cost increases.

If the space of control input is continuous, we quantize it. This quantization is represented by the following mapping:

$$^{\mathcal{U}}\mathcal{T}_{\mathcal{I}_\mathcal{A}} : \mathcal{I}_\mathcal{A} \to \mathcal{U}. \tag{2.6}$$

This quantization is required not for recording a state-action map, but for the value iteration algorithm in Fig. 2.5. In fact, quantization of $\mathcal{U}$ is not required if input parameters for control are directly written in a state-action map. This direct description can be regarded as the finest quantization.

If all use of memory is considered, the amount of memory for recording $^{\mathcal{I}_\mathcal{S}}\mathcal{T}_\mathcal{X}$ and $^{\mathcal{U}}\mathcal{T}_{\mathcal{I}_\mathcal{A}}$ is added to Eq. (2.2). When N is a large number, the additional amount of memory can be vanishingly small in many cases. We therefore use Eq. (2.2) for evaluation of L also in this case.

When a robot perceives a state $\boldsymbol{x} \in \mathcal{X}$, the action is chosen with the procedure in Fig. 2.2 If they are brought together, a policy $\boldsymbol{\pi} : \mathcal{X} \to \mathcal{U}$ can

```
1: i ⟵ ^{I_S}T_X(x)
2: j ⟵ π_MAP(i)
3: u ⟵ ^{U}T_{I_A}(j)
```

Fig. 2.2: Procedure for accessing a state-action map

be written as

$$\boldsymbol{u} = {}^{\mathcal{U}}\mathcal{T}_{\mathcal{I}_\mathcal{A}}\left(\pi_{\mathrm{MAP}}\left({}^{\mathcal{I}_\mathcal{S}}\mathcal{T}_\mathcal{X}(\boldsymbol{x})\right)\right). \tag{2.7}$$

L depends only on the number of discrete states N whichever $^{\mathcal{I}_\mathcal{S}}\mathcal{T}_\mathcal{X}$ is used. We can reduce the number of discrete states with a tricky manner

of division of $\mathcal{X}$. For example, we show two popular manners of division with two dimensional space in Fig. 2.3. In Fig. (a), the space is divided by some lines that are parallel to one of the axes. In Fig. (b), Voronoi tessellation is used for dividing. In the space, some representative points exist. Each discrete state is composed of the set of points that have the same representative point. Voronoi tessellation is more flexible than the use of simple lattice because the lattice in (a) is also a restricted version of Voronoi tessellation. If we tune a Voronoi tessellation to a patchy pattern of actions in state space, the number of discrete states may be reduced.

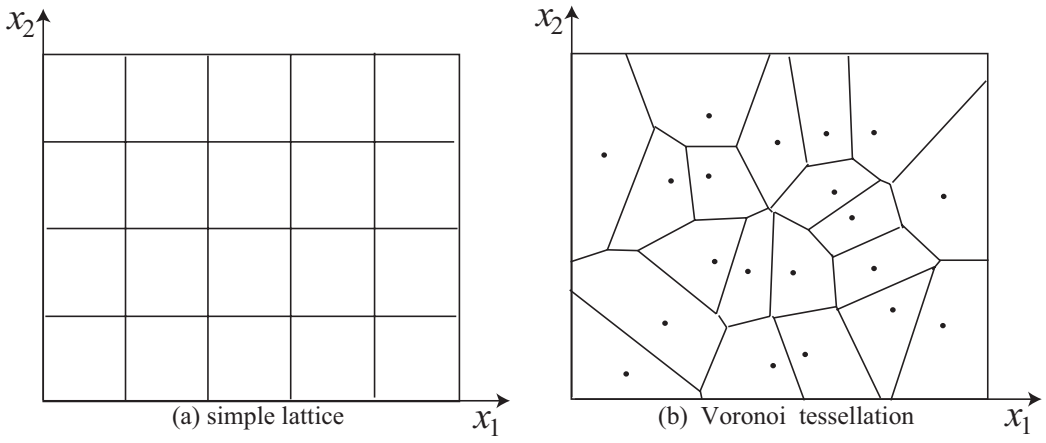


Fig. 2.3: Discretization of State Space

However, we should pay attention to the fact that the definition of this mapping influences time complexity. A transform ${}^{\mathcal{I}_S}\mathcal{T}_{\mathcal{X}}$ for the simple lattice in (a) can be defined as:

$$ {}^{\mathcal{I}_S}\mathcal{T}_{\mathcal{X}}(x_1, x_2) = \lfloor x_1/d_1 \rfloor + \lfloor x_2/d_2 \rfloor d_1, \tag{2.8} $$

where d_1 and d_2 are the numbers of intervals on x_1-axis and x_2-axis respectively. In the case of Voronoi tessellation shown in (b), when a state $\boldsymbol{x}$ is given, its nearest representative point must be chosen from all representative points. This procedure is much more complex than the calculation of Eq. (2.8). If time complexity is considered, we cannot say that a complicated manner of division is always superior to a division with simple lattice.

2.2 Creation of State-Action Map

2.2.1 State-Value Function on Look-Up Table

Though a state-action map can be created by hand, or by decomposition of an *if-then-else* code, we handle it as a solution of an optimal control problem in this thesis. This statement means that the purpose of control by a state-action map is rigidly defined as an evaluation function, and that the state-value function of the state-action map can be calculated. That is because state-value functions are used for compression as it will be explained in the next chapter.

A state-value function V^π of π can be represented by a look-up table that discretizes state space just as the state-action map of policy π. In this case, the state-value function is approximated as each real number $V^\pi(s)$ that is related to each discrete state $s \in \mathcal{S}$. $V^\pi(s)$ is called the value of state s. We need zN[bit] of memory for recording the look-up table of values, where z is the number of bits for representing a value. We can also use one of the function approximation methods that are argued in Sec. 1.3.2 if we want to economize the size of memory. However, we do not use them so as to avoid the ill-effects of function approximation that is mentioned in the section.

A policy and its state-value function on $\mathcal{S}$ has the following equation:

$$V^\pi(s) = \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^{\pi(s)} \left\{ \mathcal{R}_{ss'}^{\pi(s)} + V^\pi(s') \right\} \qquad (s \in \mathcal{S} - \mathcal{S}_\mathrm{f}) \tag{2.9}$$

in the case of a finite MDP. Values at final states: $V^\pi(s)$ $s \in \mathcal{S}_\mathrm{f}$ are fixed in advance. This equation is called a Bellman equation. From this relation, we can solve unknown π from known V^π and unknown V^π from known π.

2.2.2 Dynamic Programming

We use the value iteration algorithm, which belongs to dynamic programming (DP) for obtaining policies and state-value functions. The word *dynamic programming* (DP) is a generic term of methods that solve optimal control problems from the relation of a policy and its state-value function both in continuous and discrete state space. Here we refer to the methods that relate to Eq. (2.9) and the following equation:

$$V^*(s) = \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \left\{ \mathcal{R}_{ss'}^a + V^*(s') \right\} \quad (\forall s \in \mathcal{S} - \mathcal{S}_\mathrm{f}). \tag{2.10}$$

This equation is called a Bellman optimality equation. This equation represents the relation between the optimal policy π^* and the optimal state-value function V^*.

Methods of DP are suitable to computing the optimal action and the value at every discrete state rigidly. They are then quick against all expectations. As Sutton refers to them in [Sutton, 1998], though this method may not be practical for very large problems, DP methods are actually quite efficient compared with other methods for solving MDPs.

A value function can be calculated by the algorithm shown in Table 2.4 from a policy. This algorithm is called a policy evaluation algorithm [Sutton, 1998]. The arrow "$\longleftarrow$" denotes a substitution from the right side to the left side. Θ in the table then denotes a threshold to stop this algorithm.

```
1: initialize V(s) arbitrarily (for all s ∈ S − S_F)
2: repeat
3:         Δ ⟵ 0
4:         for each s ∈ S − S_F
5:                 v ⟵ V(s)
6:                 V(s) ⟵ Σ_{s'} P^{π(s)}_{ss'} [R^{π(s)}_{ss'} + V(s')]
7:                 Δ ⟵ max(|v − V(s)|, Δ)
8: until Δ < Θ (a small number)
```

Fig. 2.4: Policy Evaluation (from [Sutton, 1998] with minor changes)

The value iteration is a popular algorithm in DP when we want to obtain V^* from scratch. Equation (2.9) represents the balance of action $\pi(s)$ and value $V^\pi(s)$. If another action that can enhance $V^\pi(s)$ is found in a state s, π and V^π should change for better ones. The concept of value iteration algorithm is based on the above idea. Figure 2.5 is a pseudocode of the value iteration algorithm. By the iteration of Eq. (2.11), a state-value function comes close to V^*. After V converges to V^*, the optimal policy π^* can be obtained by Eq. (2.12).

When we use the value iteration algorithm in Fig. 2.5, its time complexity is $\mathrm{O}(\iota N_{\mathcal{P}})$, where $N_{\mathcal{P}}$ denotes the number of possible state transitions and ι is the number of *sweeps*. A sweep means the procedure from Line 3 to 7 in Fig. 2.5.

```
1:  initialize V(s) arbitrarily (for all s ∈ S − S_F)
2:  repeat
3:          Δ ⟵ 0
4:          for each s ∈ S − S_F
5:                  v ⟵ V(s)
6:                  Proc. 2.11
7:                  Δ ⟵ max(|v − V(s)|, Δ)
8:  until Δ < Θ (a small number)
9:  Proc. 2.12 (for all s ∈ S − S_F)
```

procedures:

$$V(s) \longleftarrow \max_a \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + V(s')] \tag{2.11}$$

$$\pi^*(s) \longleftarrow \operatorname*{argmax}_a \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + V^*(s')] \tag{2.12}$$

Fig. 2.5: Value Iteration (from [Sutton, 1998] with minor changes)

The number $N_{\mathcal{P}}$ reaches $MN(N - N_{\mathrm{f}})$ if the transition probability are not zero at any combination of $a \in \mathcal{A}, s \in \mathcal{S}$, and $s' \in \mathcal{S} - \mathcal{S}_{\mathrm{f}}$. N_{f} denotes the number of final states. In this case, time complexity of the value iteration algorithm is $\mathrm{O}(\iota MN^2)$. If there are some millions of discrete states, the value iteration algorithm will not complete within feasible time.

In many cases, however, there are many state transitions that seldom occur. Moreover, if we ignore all state transitions whose probabilities are less than a threshold, the maximum number of state transitions is fixed toward the threshold. For example, when the threshold is 0.01, the number of possible posterior states are limited to 100 for each set of a previous state and an action. When this maximum number of state transitions is written as N', the order of time complexity can be reduced to $\mathrm{O}(\iota MNN')$. We should remember that the ill-effect of the cutoff is expected.

The number of sweeps, ι, varies according to the threshold Δ. Though we can reduce the number in compensation for efficiency loss of the map, there is a necessity minimum number of sweeps. We can forecast it roughly based on expected lengths of state-action sequences in the task. Since the state-value function is computed based on the value of final states in value

iteration, the values of non-final states are sequentially fixed in the reverse direction of the state-action sequences.

Space complexity of the value iteration algorithm is $\mathrm{O}(MNN')$, which is required for recording all of the state transitions. Though they do not required for recording in value iteration, it is a waste of computation time that they are computed in every sweep. As shown in Sec. 2.3, the order can be reduced if we utilize some sort of universality of state transition in state space.

2.3 Example with Puddle World Task

The puddle world task is used for studies about learning algorithms [Sutton, 1996]. Figure 2.6 illustrates the puddle world. An agent, whose state is defined as a point on the xy-coordinate system, aims to go to the goal area. The puddle in the figure prevents the agent reaching the goal directly. If the agent enters the puddle, penalty is given. The agent should minimize the sum of the penalty and walking steps from any point to the goal. Though this task is simple, there are some characteristic parts in this world such as a narrow path, a broad area, the area where the agent should detour, edge of the world, and the puddle area. Therefore, a method examined in this world must work well in the various areas assuredly.

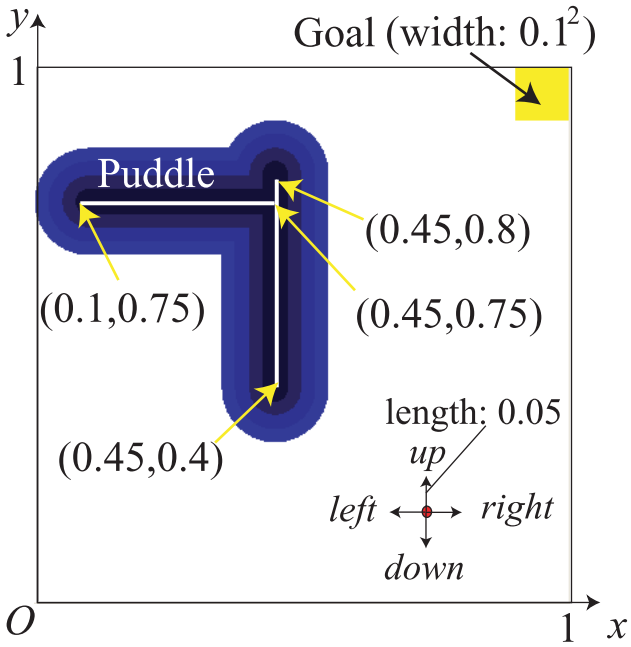


Fig. 2.6: Puddle World

2.3.1 Definition of The Task

As shown in Fig. 2.6, the shape of the world is a square

$$\mathcal{X} = \{(x, y) | x \in (0, 1), y \in (0, 1)\},$$

which can be regarded as the state space. In this case, x and y are the state variables of the agent. The agent does not have its orientation.

The center of the puddle is composed of the two line segments:

$$L_1 = \{(x, y)|x = 0.45, y \in [0.4, 0.8]\},$$
$$\text{and } L_2 = \{(x, y)|x \in [0.1, 0.45], y = 0.75\}$$

The puddle spreads 0.1 in width from the center lines. The goal is the square area at the upper right of the figure. In the state space, it means the final state:

$$\mathcal{X}_{\mathrm{f}} = \{(x, y)|x \in (0.9, 1), y \in (0.9, 1)\}.$$

We define the control vector as $\boldsymbol{u} = (\delta_x, \delta_y)^T$. δ_x and δ_y denote the target displacement of the agent along x-axis and that along y-axis respectively. Random Gaussian noise, whose standard deviation is 0.01, is added to the displacement in the direction of both axes. Therefore, a state transition is stated as

$$p^{\boldsymbol{u}}_{\boldsymbol{x}\boldsymbol{x}'} = \frac{1}{\{(2\pi)^2|\Sigma^{-1}|\}^{\frac{1}{2}}} \exp\left\{\frac{(\boldsymbol{x}' - \boldsymbol{x} - \boldsymbol{u})^T\Sigma^{-1}(\boldsymbol{x}' - \boldsymbol{x} - \boldsymbol{u})}{-2}\right\},$$

where $\Sigma = 0.01^2 \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$. Therefore,

$$p^{\boldsymbol{u}}_{\boldsymbol{x}\boldsymbol{x}'} = \frac{5000}{\pi} \exp\left(-5000|\boldsymbol{x}' - \boldsymbol{x} - \boldsymbol{u}|^2\right). \tag{2.13}$$

Since the world is surrounded by a wall, the agent cannot go beyond the wall. When the posterior state $\boldsymbol{x}'$ is not in the state space $\mathcal{X}$, $\boldsymbol{x}'$ is replaced at the intersecting point of the wall and the line segment between $\boldsymbol{x}$ and $\boldsymbol{x}'$ as shown in Fig. 2.7.

Reward -1[step] is given for each step. If it enters there, $-400\{0.1-\ell(\boldsymbol{x})\}$ is given. $\ell(\boldsymbol{x})$ is the distance from $\boldsymbol{x}$ to the nearest edge of the puddle. The total reward is defined as

$$r^{\boldsymbol{u}}_{\boldsymbol{x}\boldsymbol{x}'} = r_{\boldsymbol{x}} = -1 - 400\max\{0.1 - \ell(\boldsymbol{x}), 0\}. \tag{2.14}$$

2.3.2 Discretization of State-Space

At first, we divide each axis into some intervals. We define $[x]$ as an interval on x-axis. $[y]$ is also defined as an interval on y-axis. Then every interval is given a unique number by the following equation:

$$\begin{pmatrix} i_x \\ i_y \end{pmatrix} = \begin{pmatrix} \lfloor x\sqrt{N} \rfloor \\ \lfloor y\sqrt{N} \rfloor \end{pmatrix}, \tag{2.15}$$

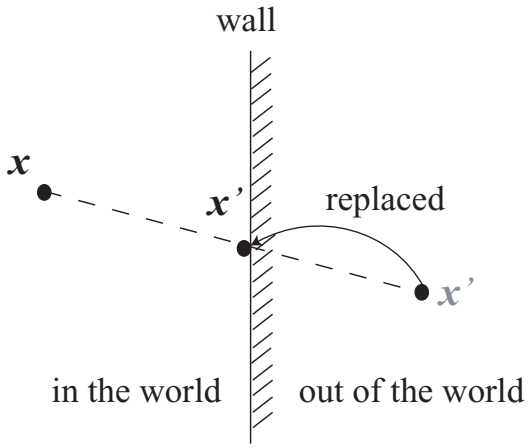


Fig. 2.7: Wall Consideration

where $\sqrt{N}$ denotes the number of intervals on each axis. It implies that the number on x-axis and that on y-axis are identical with each other in this implementation. We also confine $1/\sqrt{N}$ to a natural number. $\lfloor \cdot \rfloor$ denotes the floor function. Each discrete state is represented by the combination of intervals: $([x]_{i_x}, [y]_{i_y})$ $(i_x = 0, 1, 2, \ldots, \sqrt{N} - 1; i_y = 0, 1, 2, \ldots, \sqrt{N} - 1)$.

The transition ${}^{\mathcal{I}_S}\mathcal{T}_{\mathcal{X}}$ from $\boldsymbol{x} = (x, y)$ to a state index can be defined as

$$\begin{aligned} i &= {}^{\mathcal{I}_S}\mathcal{T}_{\mathcal{X}}(x, y) \\ &= i_y\sqrt{N} + i_x = \lfloor y\sqrt{N} \rfloor \sqrt{N} + \lfloor x\sqrt{N} \rfloor. \end{aligned} \tag{2.16}$$

With this definition, the puddle world is divided into N square discrete states. Figure 2.8 shows the numbering with Eq. (2.16) when $N = 100$.

We have to solve $\mathcal{P}^a_{ss'}$ and $\mathcal{R}^a_{ss'}$ under the above definition of discretization. If we do not consider collisions with the agent and the wall, $\mathcal{P}^a_{ss'}$ is calculated based on Eq. (2.13) by

$$\mathcal{P}^a_{ss'} = \frac{\int_{\boldsymbol{x} \in s} \int_{\boldsymbol{x}' \in s'} \frac{5000}{\pi} \exp\left(-5000 |\boldsymbol{x}' - \boldsymbol{x} - \boldsymbol{u}|^2\right) d\boldsymbol{x}' d\boldsymbol{x}}{\int_{\boldsymbol{x} \in s} d\boldsymbol{x}} \quad (\forall s \in \mathcal{S}, \forall s' \subset \Re^2). \tag{2.17}$$

s' can be an arbitrary area in xy-space. The method for coding this equation is stated in Appendix A.1. Transition probabilities that with the consideration of collision are computed by the algorithm in Sec. A.2.

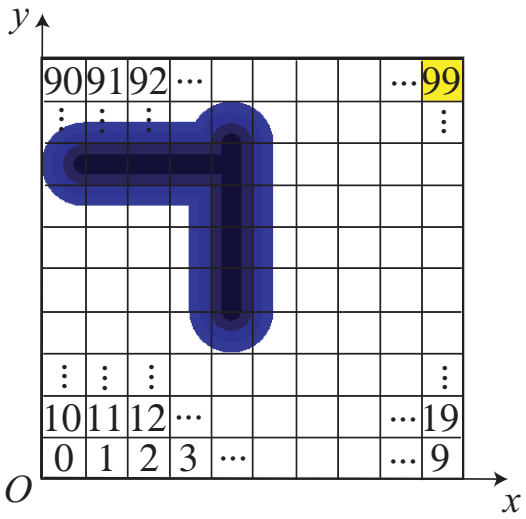


Fig. 2.8: State numbering

$\mathcal{R}^a_{ss'}$[step] is computed from Eq. (2.14) as

$$\begin{aligned}\mathcal{R}^a_{ss'} &= \frac{\int_{\boldsymbol{x}\in s} r_{\boldsymbol{x}} d\boldsymbol{x}}{\int_{\boldsymbol{x}\in s} d\boldsymbol{x}} \\ &= \frac{\int_{\boldsymbol{x}\in s} \left(-1 - 400\max\{0.1 - \ell(\boldsymbol{x}), 0\}\right) d\boldsymbol{x}}{\int_{\boldsymbol{x}\in s} d\boldsymbol{x}} \\ &= -1 - 400N \int_{\boldsymbol{x}\in s} \max\{0.1 - \ell(\boldsymbol{x}), 0\} d\boldsymbol{x}. \end{aligned} \tag{2.18}$$

Since integration of $\max\{\cdot\}$ is difficult, we solve it by using Monte Carlo method. Points in s are chosen with a reticular pattern and $\mathcal{R}^a_{ss'}$ is given as the average of penalties at the points.

In a standard puddle world problem, their types are limited to the following four vectors: $\boldsymbol{u} = 0.05(1,0)^T, 0.05(-1,0)^T, 0.05(0,1)^T$, and $0.05(0,-1)^T$. We give the symbols $a_{\text{right}}, a_{\text{left}}, a_{\text{up}}$, and a_{down} to them respectively.

2.3.3 Computing Result

State-action maps with $N_s = 10^2, 20^2, 40^2, 100^2, 200^2$, and 400^2 are created by the value iteration algorithm with the above condition. These maps and their state-value functions are shown in Fig. 2.10 and Fig. 2.9 respectively.

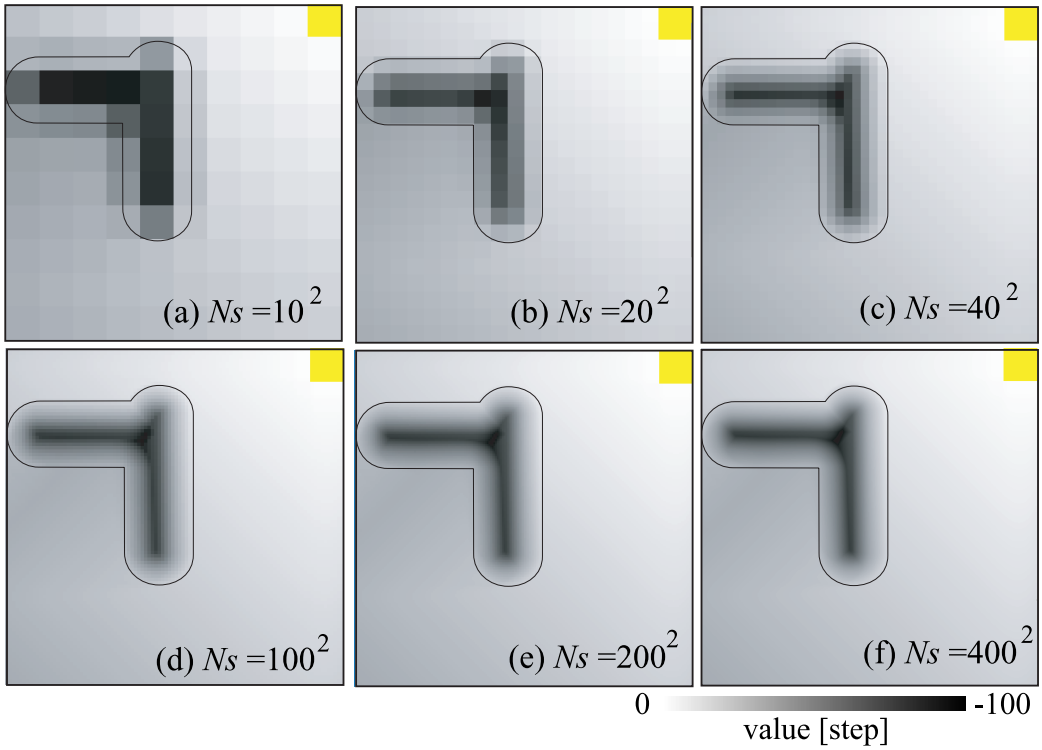


Fig. 2.9: State-Value Functions

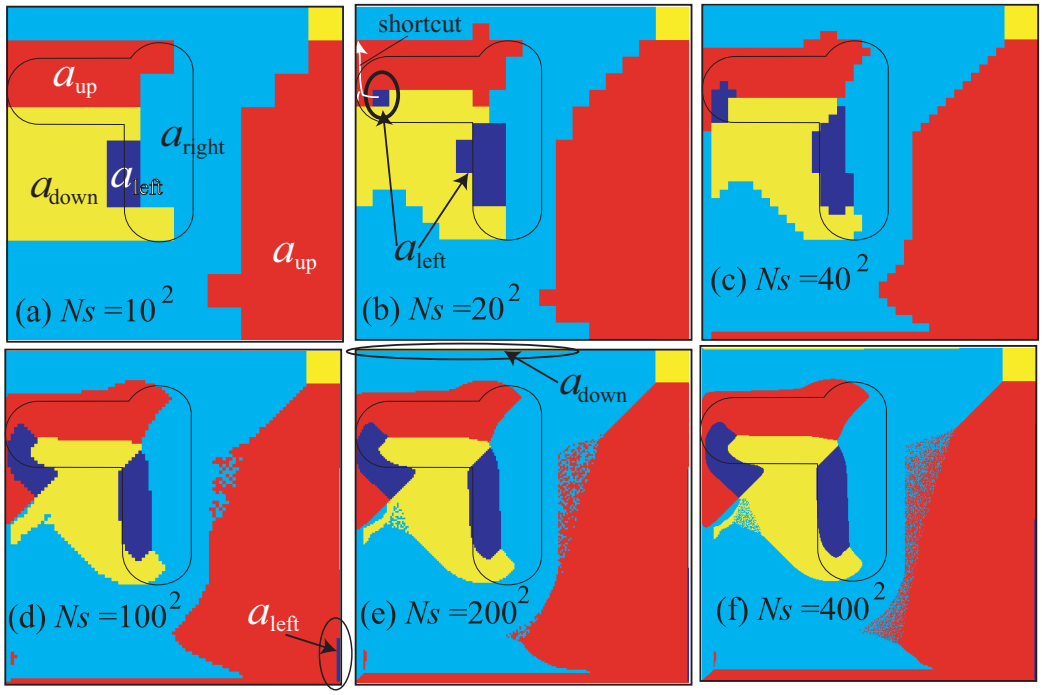


Fig. 2.10: State-Action Maps

We also show behavior of the agent from three initial states in Fig. 2.11. When $N_s = 10^2$, a policy for avoiding the puddle is obtained. We can find a shortcut, a narrow path between the wall at $x = 0$ and the edge of the puddle, in the maps with $N_s \geq 20^2$. An example of the shortcut is shown as the path of the agent from initial state "A" in Fig. 2.11. a_{left} for getting away from the wall at $x = 1$ appears when $N_s \geq 100^2$, and a_{down} for leaving the wall at $y = 1$ joins when $N_s \geq 200^2$.

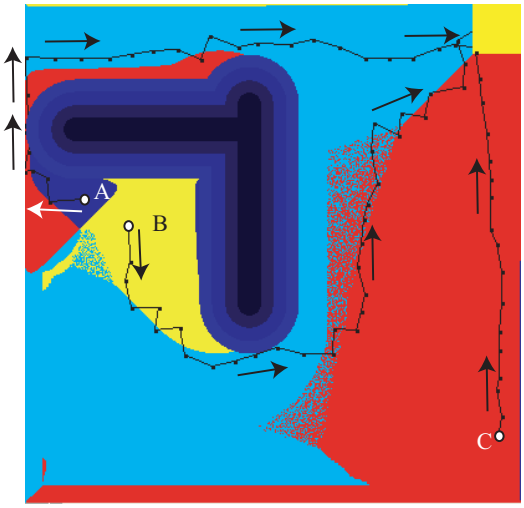


Fig. 2.11: Behavior of the agent with a state-action map ($N = 400^2$)

```
01010101010101111111010101010101111111110101010101
01011111111010101010010111111111010100001010111111
10101000010101111111101010100101011111111111111101
01010111111111111111010101011101010101010101010100
```

Fig. 2.12: Binary Sequence of A State-Action Map (a_{left}: 00, a_{right}: 01, a_{down}: 10, a_{up}: 11)

Figure 2.12 illustrates the raw binary sequence of the state-action map with $N = 10^2$. A state-action map is stored in memory with such a format.

2.3.4 Relation between Size and Efficiency

We measure the relation between the size of policy and its efficiency with the maps in Fig. 2.9. This result is used for evaluating compression methods.

Efficiency J of each state-action map is measured through motion of the agent in a computer. Initially, the agent is put at a point. The reward on each step is counted while the agent is going to the goal. The value of the initial point is computed as the sum of the values. This sort of trials are held with 10^6 initial points: $(x, y) = ((n - 0.5)10^{-3}, (m - 0.5)10^{-3})$ $(n, m = 1, 2, \ldots, 1000)$. J is obtained as the mean value of the 10^6 results. Memory usage L is evaluated by Eq. (2.2) with $M = 4$.

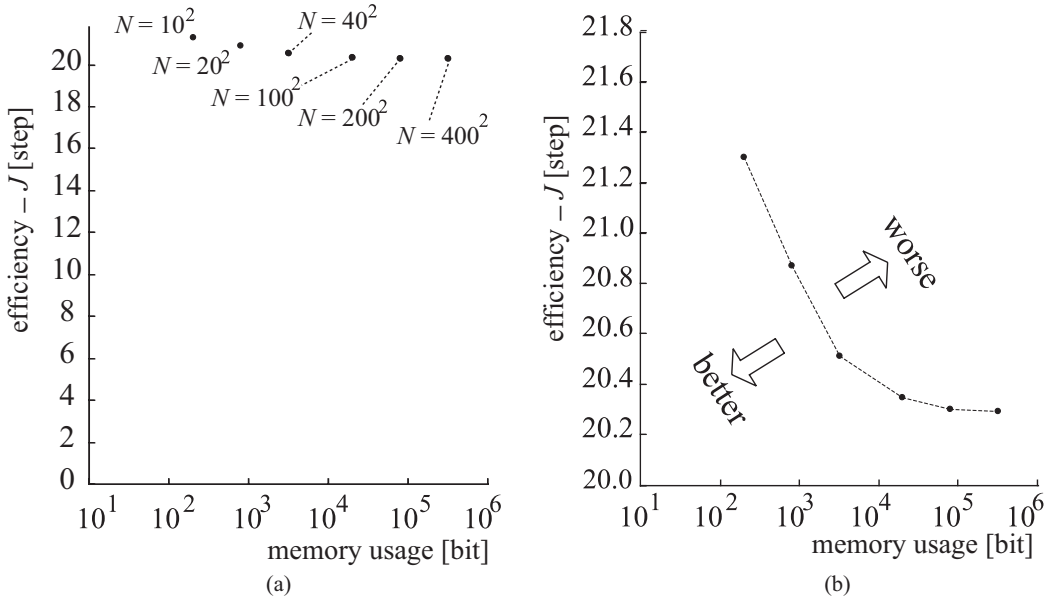


Fig. 2.13: Relation between Size and Efficiency of The State-Action Maps

The result is shown in Fig. 2.13(a). The unit of values on the vertical axis is $-J$[step]. The smaller $-J$ is, the better the efficiency is. Since $-J$ does not seem to be less than 20 however large N is, we show another graph in (b), where the intersecting point of the axes is changed. Though we have measured J five times for each state-action map, the variations in J have been too small to fill in (b). As shown in Fig. 2.13(b), the more amount of memory a state-action map consumes, the more its efficiency enhances. However, the enhancement is asymptotic.

2.4 Comparison with Value Functions

Though a state-action map seems to use memory wastefully, is it true? If the evaluation indexes: efficiency J and size L, are poorer than those of other methods, it is true.

As mentioned in Sec. 1.3.2, many function approximation techniques are proposed and utilized for representing the state-value function. They can be used in place of a state-action map if we add some algorithms for decision making with them. The time for decision making by a state-value function will be longer than that with a state-action map. However, use of a state-value function should be taken into account if its efficiency-per-size is better than a state-action map.

We compare the state-action maps obtained in this chapter with the following value functions:

- the state-value functions in Fig. 2.9,
- state-value functions obtained by tile coding [Albus, 1975a; Albus, 1975b; Sutton, 1996], and
- interpolated state-value functions computed from the functions in Fig. 2.9.

2.4.1 Decision Making from State-Value Function

At first, we fix the method of how to choose actions from a state-value function. An algorithm that is based on a Monte Carlo method is used for that purpose.

From state $\boldsymbol{x}$ of the agent, the algorithm generates ℓ samples of posterior states according to the probability density function $p^a_{\boldsymbol{x}\boldsymbol{x}'}$ for each action a. The samples are written as $\boldsymbol{x}'^{(0)}_a, \boldsymbol{x}'^{(1)}_a, \ldots, \boldsymbol{x}'^{(\ell-1)}_a$ here. From the samples, the following value

$$\hat{Q}(\boldsymbol{x}, a) = \frac{1}{\ell} \sum_{i=0}^{\ell-1} \left[r^a_{\boldsymbol{x}\boldsymbol{x}'^{(i)}_a} + V(\boldsymbol{x}'^{(i)}_a) \right] \tag{2.19}$$

is calculated. An action is then chosen based on the following policy

$$\pi_{\hat{Q}}(\boldsymbol{x}) = \operatorname*{argmax}_{a \in \mathcal{A}} \hat{Q}(\boldsymbol{x}, a). \tag{2.20}$$

The larger ℓ is, the more the value of $\hat{Q}(\boldsymbol{x}, a)$ is close to the actual action value function:

$$Q(\boldsymbol{x}, a) = \int_{\mathcal{X}} p^a_{\boldsymbol{x}\boldsymbol{x}'} \left[r^a_{\boldsymbol{x}\boldsymbol{x}'} + V(\boldsymbol{x}') \right] d\boldsymbol{x}'. \tag{2.21}$$

Though calculation of this equation is more precise for decision making than the Monte Carlo method, the difficulty of this calculation toward an arbitrary format of a state-value function may be imaginable.

2.4.2 Implementation of Tile Coding

As shown in [Sutton, 1996], we cover the puddle world with five discrete state spaces. One of them, named $\mathcal{S}_0$, divides the x-axis and y-axis into $\sqrt{N_{\text{tile}}}$ intervals respectively. Each cell in $\mathcal{S}_0$ is equally divided into 25 pseudo cells by the other spaces: $\mathcal{S}_2$, $\mathcal{S}_3$, $\mathcal{S}_4$, and $\mathcal{S}_5$. Figure 2.14 shows the case with $N_{\text{tile}} = 2^2$. $\mathcal{S}_2$, $\mathcal{S}_3$, $\mathcal{S}_4$, and $\mathcal{S}_5$ have an additional interval on each axis; otherwise cells of $\mathcal{S}_0$ adjacent to a wall cannot be divided into 25 parts.

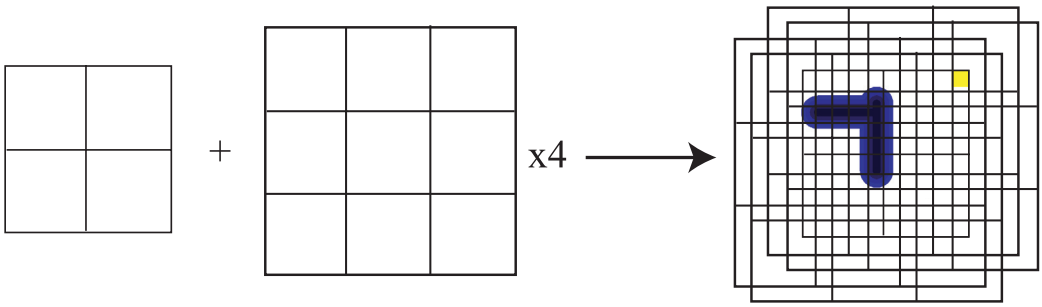


Fig. 2.14: Tile coding (an example with $N_s = 2^2$)

The value iteration algorithm is separately applied to the five kinds of space. Since one of them is identical with the normal state-value function, the value iteration algorithms are executed on the other four kinds of space. When the five state-value functions are superposed, we can obtain one of the state-value functions shown in the lower part of Fig. 2.15. As shown in the figure, each state-value function is smoothed by the superposition.

Since actions cannot be recorded in pseudo cells, the agent should decide its action with state-value functions on the five spaces. The agent needs

$$L = \left\{ N_{\text{tile}} + 4(\sqrt{N_{\text{tile}}} + 1)^2 \right\} z \text{ [bit]} \tag{2.22}$$

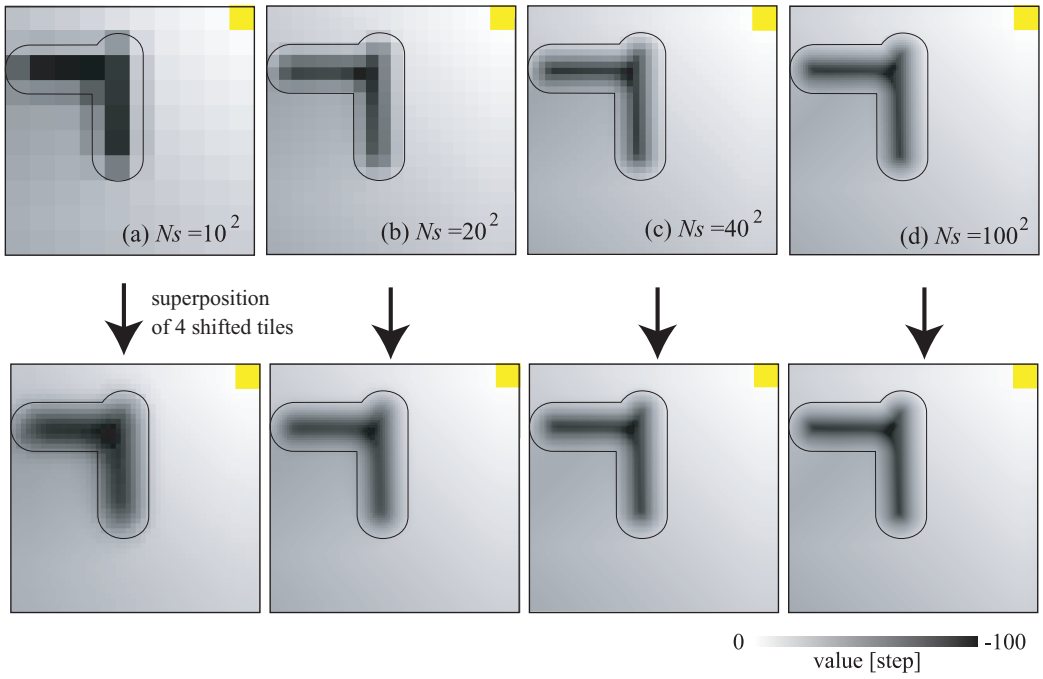


Fig. 2.15: State-Value Functions Obtained by Tile Coding

to store the five state-value functions. z is the number of bits for recording the value on a state. z is not the number of bits at computation of a state-value function. We use 32[bit] for computing it and reduce the number to z[bit] at recording. Each value is rounded off to a number from 0 to $2^z - 1$ evenly.

2.4.3 Implementation of Interpolation

Interpolation techniques have been used for learning as shown in [Takahashi, 2001]. In Takahashi's case, this technique has been applied to Q-learning.

We utilize this technique to enhance pseudo resolution of the state-value functions shown in Fig. 2.9. In Fig. 2.16, a part of a state-value function is drawn. The value at $\boldsymbol{x}$ is calculated as the weighed mean of values of states around $\boldsymbol{x}$. The weight is in proportion to the square measure of the overlapping area between each discrete state and a square whose center is $\boldsymbol{x}$. The size of the square is identical with a discrete state. When the value and the weight of each discrete state are represented by v_i and w_i $(i = 0, 1, 2, 3)$

respectively as shown in the figure, the value at $\boldsymbol{x}$ is calculated as

$$V(\boldsymbol{x}) = \frac{1}{\sum_{i=0}^{3} w_i} \sum_{i=0}^{3} v_i w_i. \tag{2.23}$$

Differently from the functions obtained by tile coding, the interpolated state-value functions consume no more memory than the size of the original state-value functions. Though we should count the amount of memory for the algorithm of the interpolation, it is a negligible amount when the number of discrete states are huge. Since the puddle world task is an example, we do not take the amount of memory into consideration.

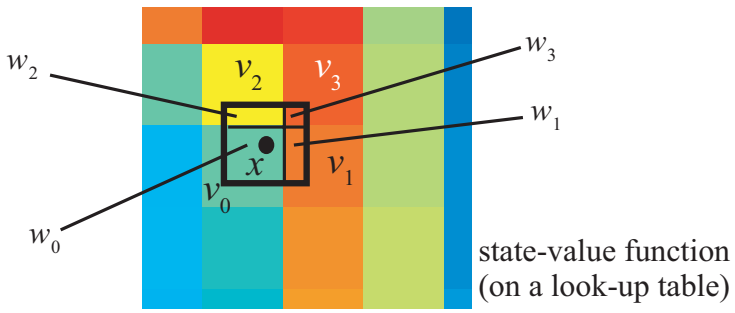


Fig. 2.16: Interpolation

In Fig. 2.17, state-value functions and their interpolation are illustrated. When $N = 10^2, 20^2, 40^2$, we can recognize that the mosaic patterns of the original functions are smoothed by the interpolation. The state-value function with $N = 100^2$ and more finer state-value functions can also be smoothed though we cannot judge it on this resolution.

2.4.4 Evaluation Result

The three kinds of state-value functions and the state-action maps are evaluated with the same simulation in Sec. 2.3.4. The number of samples on the Monte Carlo method and the number of bits for representing values are chosen from $(\ell, z) = (10, 8), (10, 16), (100, 8), (100, 16)$. The number of discrete states N are set to $N = 20^2, 40^2, 100^2, 200^2, 400^2$ in the cases of normal state-value functions and interpolated state-value functions. The parameter N_{tile} of tile coding is chosen from $N_{\text{tile}} = 20^2, 40^2, 100^2, 200^2$.

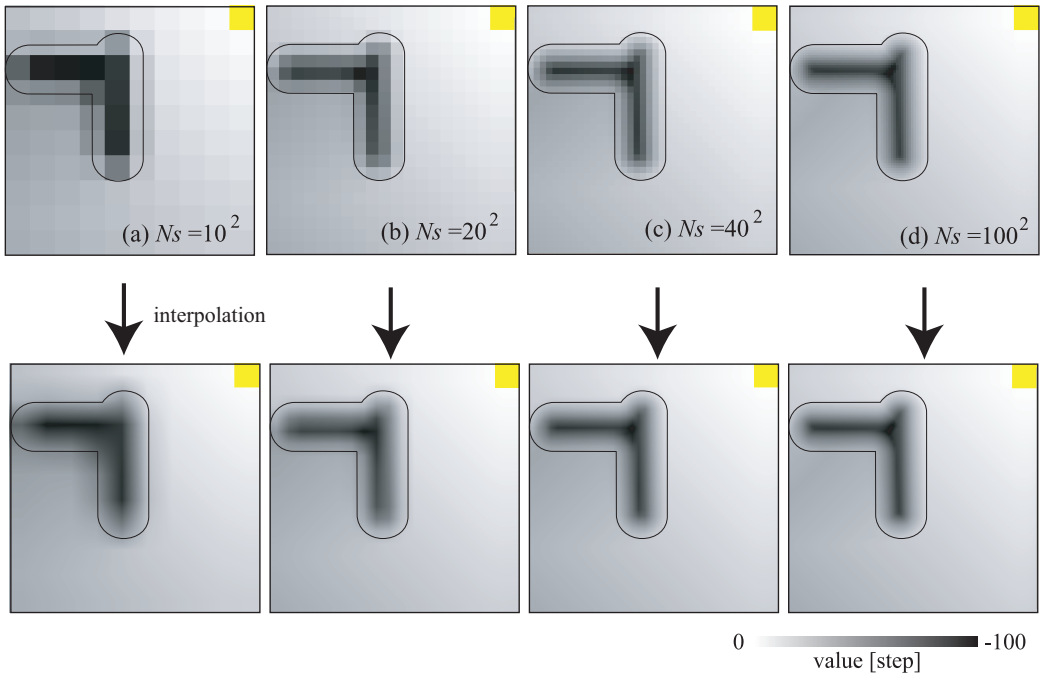


Fig. 2.17: Interpolated State-Value Function

The results are shown in Fig. 2.18. In all of the figures, the results of tile coding are not superior not only to those of state-action maps, but also to those of the normal state-value functions. From the figures, moreover, we can understand that value iteration by tile coding with $N_{\text{tile}} = 10^2$ cannot obtain better state-value functions than value iteration by simple discretization with $N = 20^2$. The same goes for the pair of $N_{\text{tile}} = 20^2$ and $N = 40^2$. We should say that tile coding is not always suitable for value iteration. Tile coding is suitable for reinforcement learning since it can reduce the number of trials for learning.

In the case of interpolation, some results in Fig. (a) are better than those of normal state-value functions. When the number of samples on the Monte Carlo method and the number of bits for representing values, the interpolated state-value functions seems to make the Monte Carlo method estimate action values more accurately than the normal functions. In other cases, however, the effect of interpolation is not significant. That is because the Monte Carlo method also has a functional capability of interpolation. Since the Monte Carlo method cannot be executed without the knowledge of dynamics, this result also indicates the difference between DP and reinforcement learning.

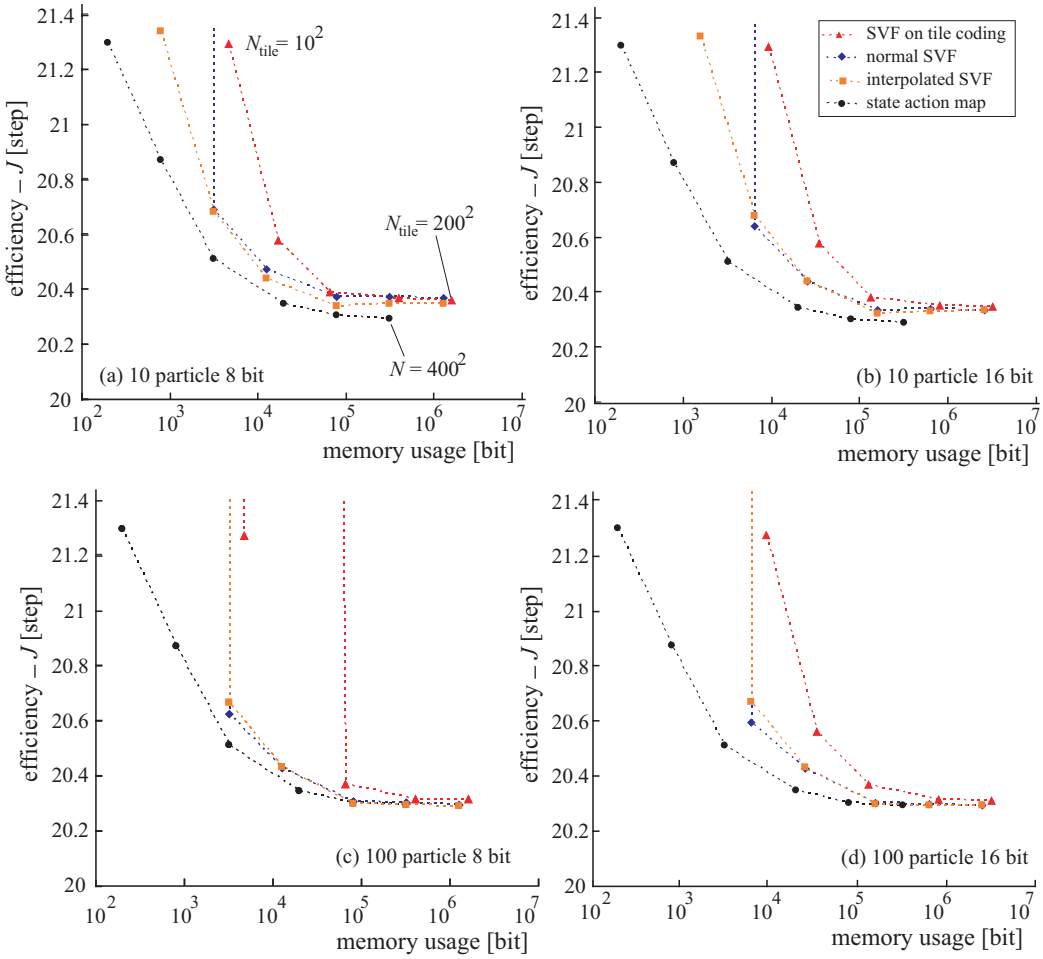


Fig. 2.18: Performance of State-Value Functions and State-Action Maps

2.5 Discussion

In this chapter, we have defined the state-action map and have explained how to create it by dynamic programming (DP) with an example of the puddle world task. Moreover, state-action maps for the puddle world task have been compared with the computing results of tile-coding and interpolation quantitatively.

Assurance of Dynamic Programming for State-Action Maps

In the simulation evaluation of the state-action maps with the puddle world task, we can give a good example of the relation between efficiency J and memory consumption L. The more memory is used for a state-action map, the more its efficiency enhances as shown in Fig. 2.13.

Because this simple relation does not always hold true in some other methods, it indicates the assurance of the DP method that is implemented in this chapter. In the case of learning methods, for example, the discretization of state space should be chosen according to distances of state transitions in the space. If the discretization is too coarse, an agent sometimes sticks around the same discrete state. On the other hands, efficiency of learning is dramatically lost if the discretization is too fine. The DP method implemented in this chapter can avoid these problems because it regards state transitions as flow of probabilities.

The Unexpected Ability of Function Approximation

We have compared the state-value functions on look-up tables with the functions obtained by the tile coding and by the interpolation. From the simulation, we notice that the tile coding is not effective for memory economization though there is a possibility that tile coding can reduce the time for learning or planning. No state-value function with tile coding is superior to any state-action map on a simple look-up table in the simulation of the puddle world task as the results in Fig. 2.18. Interpolation is not much effective too. As a matter of course, we should try those methods when memory is not enough to represent all discrete states on a simple look-up table. However, the value iteration algorithm sometimes can solve a problem with very coarse discretization as the state-action map with $N = 10^2$ on the puddle world task. We should not use a complicated function approximation method without any reason.

Chapter 3

State-Action Map Compression

In this chapter, we propose a vector quantization (VQ) method for compressing state-action maps. To write this section, we are greatly helped by Gersho and Gray's text [Gersho, 1992].

This chapter is composed of seven sections. In Sec. 3.7, necessary characters of compression methods and compressed state-action maps are mentioned. The major part of the proposed method in this thesis is presented in Sec. 3.2–3.4. Brief overview of VQ is done in Sec. 3.2. We will understand that VQ can be one of the most suitable compression methods for state-action maps. The format of compressed state-action maps by VQ is then explained in Sec. 3.3. After that, in Sec. 3.4, we present the most important mathematical formulae that make VQ compress state-action maps efficiently. The proposed method is applied to the state-action maps for the puddle world task in Sec. 3.5. In Sec. 3.6, the proposed method is compared to one of the most competitive methods. The method creates policies on a binary-tree structure. We conclude this chapter in 3.7 with the discussion about the result of comparison.

3.1 Suitable Manner for Compressing State-Action Map

To compress a state-action map, we should choose suitable methods for compression. A method determines the format of compressed state-action maps.

First of all, the format of maps should have good quality as a reflexive policy. On a relevant note, there is a concept of instantaneous decordable codes in the source coding theorem. When an instantaneous decordable code is read from its head, it should fulfill the following conditions:

- the end of the sequence of bits that denotes a word (a unit of significant data) can be recognized without waiting for the next bit, and
- the word can be decoded without waiting for it.

Well coded data by Huffman coding belongs to this category. Reflexive policies, on the other hand, must fulfill more severe conditions. It must be random access data because sequences of state transitions are not fixed. A compressed state-action map must provide an action index for a state within a fixed time cycle after the state is detected.

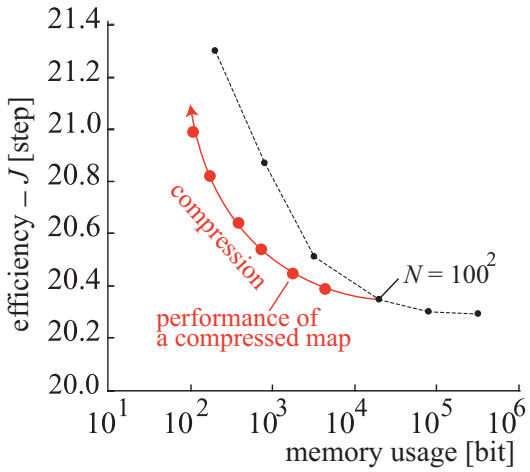


Fig. 3.1: An Ideal Case of Compression

Instead, we should allow efficiency loss of a state-action map by compression. It means that lossy compression methods are preferred rather

than lossless compression methods. Generally speaking, Lossy compression methods can actualize higher compression ratio than lossless compression methods. Moreover, lossy compression methods can control the compression ratio in accordance with required efficiency, or control the efficiency in accordance with a required compression ratio. Figure 3.1 illustrates an example. The data in this figure is identical with the data in Fig. 2.13(b). We assume that some compressed state-action maps are obtained from the $N = 100^2$ map. If the compression proceeds with the small efficiency loss as shown in Fig. 3.1, the adjustment of size or efficiency is easy. Not only high compression ratio, but also this kind of tractability will be required for practical use.

Figure 3.1 implies another important issue. If a pair of efficiency and size obtained from a compressed state-action map is worse than a coarse state-action map, the compressed data is meaningless. We can regard a coarse discretization as a way of compression. The proposed method must be advantageous against this simple compression method.

3.2 Vector Quantization

From the requisition of Sec. 3.1, we utilize vector quantization (VQ) [Gersho, 1992] for our purpose. Here we explain the fundamental idea of VQ. We can understand it is quite powerful and universal method for compression of signals, while we will also understand that its exact copy cannot be applied to compression of state-action maps.

3.2.1 Vector Quantization for Compression of Finite Amount of Data

Here we explain basic definitions of VQ in the case where the number of vectors is finite. Let us suppose that k-dimensional Euclidean space, $\Re^k$, is dotted with a set of vectors:

$$\mathcal{V} = \{\boldsymbol{v}_i | i = 0, 1, 2, \ldots, N_\nu - 1\} \tag{3.1}$$

as shown in Fig. 3.2(a). Here we define a mapping:

$$\Omega : \mathcal{V} \to \mathcal{C} \tag{3.2}$$

where $\mathcal{C} = \{\boldsymbol{c}_i \in \Re^k | i = 0, 1, 2, \ldots, N_c - 1\}$. $\mathcal{C}$ and $\boldsymbol{c} \in \mathcal{C}$ are called a codebook and a representative vector respectively. Ω, which is called a vector quantizer, changes any vector in $\mathcal{V}$ into a representative vector. As shown in Fig. 3.2(b), when a vector quantizer is given, the vectors in $\mathcal{V}$ are classified into N_c clusters:

$$\mathcal{K}_j = \{\boldsymbol{v} | \Omega(\boldsymbol{v}) = \boldsymbol{c}_j\} \ (j = 0, 1, 2, \ldots, N_c - 1). \tag{3.3}$$

Therefore, an algorithm for obtaining a suitable vector quantizer is called a clustering algorithm. A vector quantizer can be also regarded as a mapping from an index of a vector to an index of a representative vector. We represent this mapping for indexes as

$$\omega : \mathcal{I}_\mathcal{V} \to \mathcal{I}_\mathcal{C}, \tag{3.4}$$

where $\mathcal{I}_\mathcal{V} \equiv \{0, 1, 2, \ldots, N_\nu - 1\}$ and $\mathcal{I}_\mathcal{C} \equiv \{0, 1, 2, \ldots, N_c - 1\}$. Indexes in $\mathcal{I}_\mathcal{V}$ and in $\mathcal{I}_\mathcal{C}$ are called vector indexes and representative vector indexes respectively in this thesis.

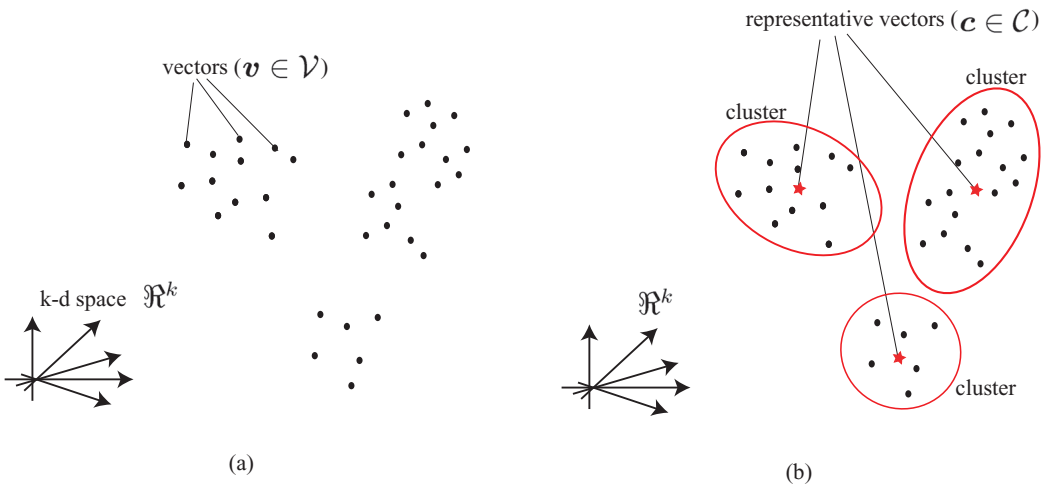


Fig. 3.2: vector quantization

3.2.2 Distortion Measure

Roughly speaking, a vector quantizer is regarded as a good one if it can convert vectors in $\mathcal{V}$ into representative vectors with small degradation. If we want to evaluate a vector quantizer, the rate of degradation should be measured. In [Gersho, 1992], "a distortion measure" is used as the term that denotes the rate of degradation. In almost all methods of VQ, a distortion measure is defined and used for obtaining an appropriate vector quantizer. The distortion measure between a vector and a representative vector is defined as

$$D : \mathcal{V} \times \mathcal{C} \rightarrow [0, \infty). \tag{3.5}$$

In the case where vectors belong to Euclidean space $\Re^k$, a distortion measure can be defined as any distance between $\boldsymbol{v} \in \mathcal{V}$ and $\boldsymbol{c} \in \mathcal{C}$ in $\Re^k$. According to [Gersho, 1992], the following distortion measures of Ω toward $\mathcal{V}$ are frequently used

$$\mathcal{D}(\mathcal{V}, \Omega) = \sum_{\boldsymbol{v} \in \mathcal{V}} D\left(\boldsymbol{v}, \Omega(\boldsymbol{v})\right), \tag{3.6}$$

$$\text{or } \mathcal{D}(\mathcal{V}, \Omega) = \max_{\boldsymbol{v} \in \mathcal{V}} D\left(\boldsymbol{v}, \Omega(\boldsymbol{v})\right) \tag{3.7}$$

for defining the optimal vector quantizer:

$$\Omega^* = \operatorname*{argmin}_{\Omega} \mathcal{D}(\mathcal{V}, \Omega). \tag{3.8}$$

When Eq. (3.6) is adopted, Ω^* is a vector quantizer that can minimize the distortion d averagely. The use of Eq. (3.7) means the request that a worst-case distortion of vectors should be minimized.

3.2.3 Blocking for VQ of A Sequence of Numbers

Though VQ is the method for reducing the redundancy of a set of vectors, it is usually used for compressing such a sequence of numbers as digital images [Tsai, 2000; Fekri, 2000] or digital sounds [Huang, 2002]. We show the flow of algorithms for compressing a sequence of numbers in Fig. 3.3. Besides the clustering process, the blocking process is required for the compression as shown in this figure.

When a sequence of numbers is regarded as an ordered set

$$\mathcal{N} = \{n_i | i = 0, 1, 2, \ldots, N-1\}, \tag{3.9}$$

these numbers are classified into some sets. In other words, $\mathcal{N}$ is divided into some blocks of equal length. We therefore call this procedure *blocking*. The numbers in each block are arranged in an ordered set, which is regarded as a vector.

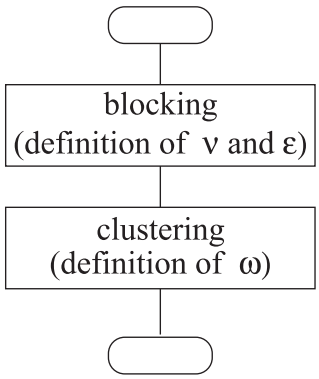


Fig. 3.3: The flow of VQ for sequence of numbers

In a rigorous manner, the blocking procedure is to give every $n_i \in \mathcal{N}$ the position of $\varepsilon(i)$th element of $\nu(i)$th vector by using a pair of transforms:

$$\nu : \mathcal{I} \to \mathcal{I}_{\mathcal{V}} = \{0, 1, 2, \ldots, N_\nu - 1\} \tag{3.10}$$

$$\varepsilon : \mathcal{I} \to \mathcal{I}_{\mathcal{E}} = \{0, 1, 2, \ldots, N_\varepsilon - 1\}, \tag{3.11}$$

where $\mathcal{I} = \{0, 1, 2, \ldots, N-1\}$. $\mathcal{I}, \mathcal{I}_\mathcal{V}$ and $\mathcal{I}_\mathcal{E}$ are called the set of indexes, the set of vector indexes, and the set of element indexes respectively. N_ν is the number of N_ε-dimensional vectors. Hence, $N_\nu N_\varepsilon = N$. Figure 3.4 shows the relation between these kinds of index with a sequence of data that is arranged by ν and ε. Of course, two different indexes in $\mathcal{I}$ should not transformed to the identical pair of a vector index and an element index. With these transforms, $\mathcal{N}$ is divided into the set of N_ν vectors. The set of vectors are defined as $\mathcal{V} = \{\boldsymbol{v}_0, \boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_{N_\nu - 1}\}$. A vector has the following form:

$$\begin{aligned} \boldsymbol{v}_{i_\nu} = (&v_{(i_\nu,0)}, v_{(i_\nu,1)}, \ldots, v_{(i_\nu,i_\varepsilon)}, \ldots, v_{(i_\nu,N_\varepsilon-1)}) \\ &(v_{(i_\nu,i_\varepsilon)} \in \mathcal{N}, i_\varepsilon = 0, 1, 2, \ldots, N_\varepsilon - 1), \end{aligned} \tag{3.12}$$

where

$$\begin{pmatrix} i_\nu \\ i_\varepsilon \end{pmatrix} = \begin{pmatrix} \nu(i) \\ \varepsilon(i) \end{pmatrix}. \tag{3.13}$$

When the set of vectors is compressed to representative vectors based on the transform in Eq. (3.4), the data for $i \in \mathcal{I}$ can be read from i_εth element of representative vector $\boldsymbol{c}_{i_{\nu\omega}}$. $i_{\nu\omega}$ and i_ε are obtained by

$$\begin{pmatrix} i_{\nu\omega} \\ i_\varepsilon \end{pmatrix} = \begin{pmatrix} \omega\left(\nu(i)\right) \\ \varepsilon(i) \end{pmatrix}. \tag{3.14}$$

Figure 3.5 illustrates the relation between original data and its compressed data. In this figure, a look-up table that represents Eq. (3.4) is added to Fig. 3.4. We name this look-up table **a quantization table**. The blocked data is then changed to the set of representative vectors. The look-up table in which $\boldsymbol{c}_i$ $(i = 1, 2, \ldots, N_c - 1)$ are recorded is called **a codebook**. The compressed data is composed of the pair of a quantization table and a codebook.

We sometimes represent an action index $c_{(i_{\nu\omega},i_\varepsilon)}$ on the codebook as $c(i_{\nu\omega}, i_\varepsilon)$. In this case, the codebook can be regarded as the mapping $c : \mathcal{I}_\mathcal{C} \times \mathcal{I}_\mathcal{E} \to \mathcal{I}_\mathcal{A}$.

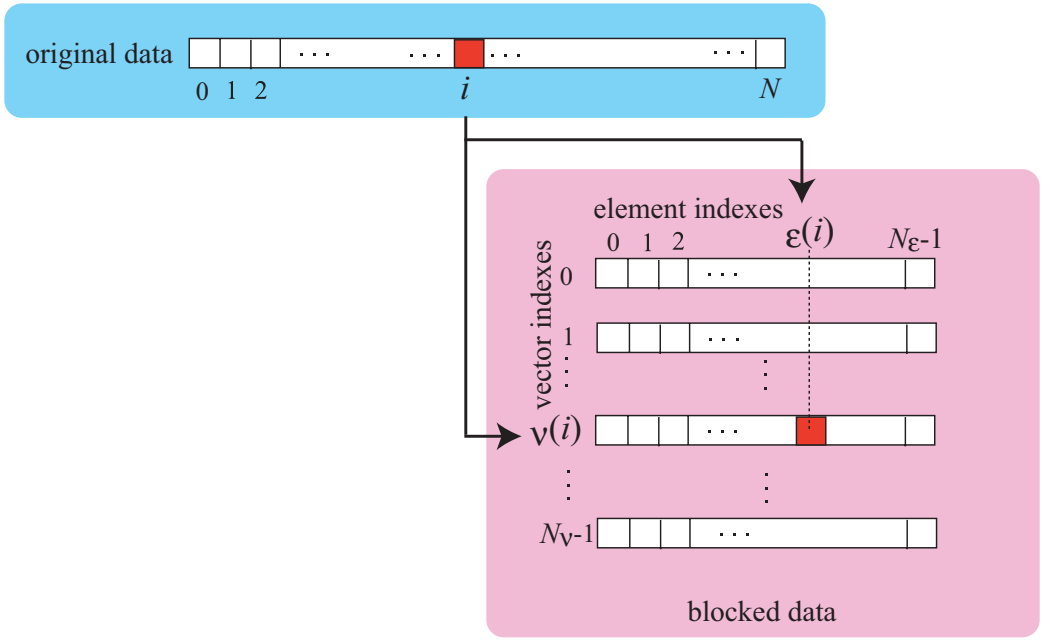


Fig. 3.4: Blocking

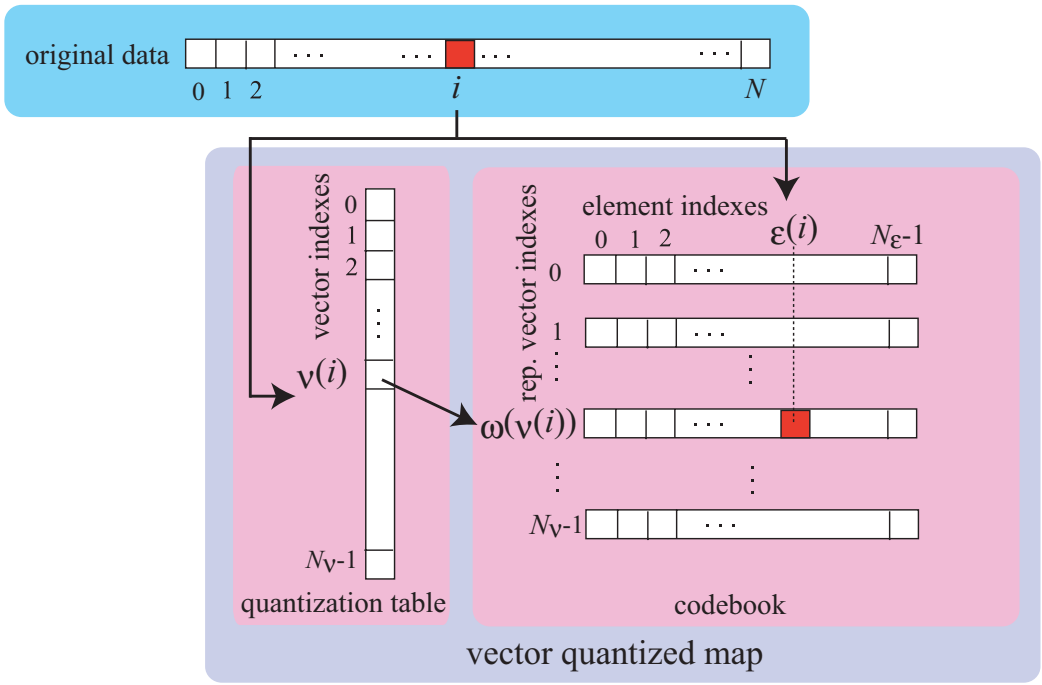


Fig. 3.5: Structure of Vector Quantized Data

3.3 Vector Quantized State-Action Map and Its Character

Since a state-action map is a sequence of binary numbers, it can be compressed by the method in Sec. 3.2.3. The format of the compressed map is identical with the compressed data in Fig. 3.5 except that the data is action indexes. However, if a state-action map is compressed in a similar fashion of compression of images, we cannot actualize high compression ratio.

In this section, a format of a compressed state-action map by VQ is firstly given. Secondly, we formulate the evaluation indexes: size L and efficiency J for the compressed map. There we will notice that it is very difficult to create a compressed state-action map without losing the efficiency J. Thirdly, we propose a definition of distortion measure that is specialized for state-action maps. Then we give some concrete creating processes of VQ maps.

3.3.1 Vector Quantized State-Action Map

If we replace the set of indexes $\mathcal{I}$ by the set of state indexes $\mathcal{I}_{\mathcal{S}}$, we can apply VQ to compression of any state-action map in form. In this case, a vector has action indexes as its elements. When ν and ε are given, each vector is defined as

$$\begin{aligned}\boldsymbol{v}_{i_\nu} &= (v_{(i_\nu,0)}, v_{(i_\nu,1)}, \ldots, v_{(i_\nu,N_\varepsilon-1)}) \\ &= \big(\pi_{\mathrm{MAP}}(s_{\mu(i_\nu,0)}), \pi_{\mathrm{MAP}}(s_{\mu(i_\nu,1)}), \pi_{\mathrm{MAP}}(s_{\mu(i_\nu,2)}), \ldots, \pi_{\mathrm{MAP}}(s_{\mu(i_\nu,N_\varepsilon-1)})\big),\end{aligned} \tag{3.15}$$

where $\mu(i,j)$ denotes the state index that is divided into $i \in \mathcal{I}_{\mathcal{V}}$ and $j \in \mathcal{I}_{\mathcal{E}}$ by Eq. (3.10) and (3.11). The function μ is defined as

$$\mu : \mathcal{I}_{\mathcal{V}} \times \mathcal{I}_{\mathcal{E}} \to \mathcal{I}_{\mathcal{S}}. \tag{3.16}$$

When a transform ω in Eq. (3.4) is given, the set of vectors $\mathcal{V} = \{\boldsymbol{v}_{i_\nu} | i_\nu = 0, 1, 2, \ldots, N_\nu - 1\}$ is changed into a set of representative vector $\mathcal{C} = \{\boldsymbol{c}_{i_{\nu\omega}} | i_{\nu\omega} = 0, 1, 2, \ldots, N_c - 1\}$. When each representative vector is written as

$$\boldsymbol{c}_{i_{\nu\omega}} = (c_{(i_{\nu\omega},0)}, c_{(i_{\nu\omega},1)}, c_{(i_{\nu\omega},2)}, \ldots, c_{(i_{\nu\omega},N_\epsilon-1)}) \quad (c_{(i_{\nu\omega},i_\varepsilon)} \in \mathcal{A}, \text{or } c_{(i_{\nu\omega},i_\varepsilon)} \in \mathcal{U}), \tag{3.17}$$

the action index for the state index i can be read by Eq. (3.14). When ω and $\mathcal{C}$ are recorded as the quantization table and the codebook, we call

these look-up tables **a vector quantized state-action map (a VQ state-action map, or a VQ map)**.

3.3.2 Size, Accessibility, and Efficiency Loss of VQ Map

When we build a VQ map, its consumption of memory, L[bit], is calculated by

$$\begin{aligned} L_{\mathrm{VQ}} &= N_\nu \lceil \log_2 N_c \rceil + N_\varepsilon N_c \lceil \log_2 M \rceil \\ &= \frac{N}{N_\varepsilon} \lceil \log_2 N_c \rceil + N_\varepsilon N_c \lceil \log_2 M \rceil, \end{aligned} \tag{3.18}$$

where $\lceil \cdot \rceil$ is the ceiling function. The first term of the right side of this equation is required bits for the quantization table and the second one is required bits for recording the codebook. Transforms ν and ε should be implemented as simple program code and we do not count bits for them.

The process in Fig. 3.6 are required for accessing the VQ map. In this case, Line 2 in Fig. 2.2, which illustrates the process for accessing a state-action map, is decomposed into the four lines from Line 6 to 9. The number of operations from Line 6 to 9 for a VQ map is several times as large as Line 2 for a state-action map. However, they are composed of operations for integer values. Their load is much smaller than that of any search algorithm and that for accessing any kind of function approximation data. Therefore, computing time does not need to be discussed unless in extreme circumstances.

```
1:  i ⟵ ᴵˢ𝒯_𝒳(x)
2:  GO_TO ACCESS_VQ_MAP
3:  u ⟵ ᵘ𝒯_{I_𝒜}(j)
4:
5:  ACCESS_VQ_MAP:
6:          i_ν  ⟵ ν(i)
7:          i_νω ⟵ ω(i_ν)
8:          i_ε  ⟵ ε(i)
9:          j    ⟵ c(i_νω, i_ε)
```

Fig. 3.6: Procedure for accessing a VQ state-action map

By the compression, π_{MAP} becomes another map π'_{MAP}, the loss of efficiency can be measured by Eq. (2.3) as

$$J^{\pi_{\mathrm{MAP}}} - J^{\pi'_{\mathrm{MAP}}} = \int_{\mathcal{X}} p(\boldsymbol{x}_0) \left\{ V^{\pi_{\mathrm{MAP}}}(\boldsymbol{x}_0) - V^{\pi'_{\mathrm{MAP}}}(\boldsymbol{x}_0) \right\} d\boldsymbol{x}_0. \tag{3.19}$$

When we use the policy iteration algorithm shown in Fig. 2.4, this value can be computed by

$$J^{\pi_{\mathrm{MAP}}} - J^{\pi'_{\mathrm{MAP}}} = P(\boldsymbol{x}_0 \in s) \left\{ V^{\pi}(s) - V^{\pi'}(s) \right\}, \tag{3.20}$$

where $P(\boldsymbol{x}_0 \in s) = \int_s p(\boldsymbol{x}_0) d\boldsymbol{x}$.

3.4 State-Value Distortion

The Specialty of State-Action Map on Compression

We utilize some existing clustering algorithms for DP. Since the problem of how to choose a suitable ω is argued in the studies of VQ, we follow their footsteps. However, the existing algorithms are not workable if a distortion measure for action indexes in a state-action map is not defined.

In Eq. (3.5), a distortion measure is defined as $D : \mathcal{V} \times \mathcal{C} \to [0, \infty)$. For instance, the inner product between a vector and a representative vector may be the simplest distortion measure. In this case, D is represented by

$$D(\boldsymbol{v}, \boldsymbol{c}) = \boldsymbol{v} \cdot \boldsymbol{c}. \tag{3.21}$$

The inner product must be defined toward a pair of sequences of actions if it is applied to vectors in a state-action map. When each action $a \in \mathcal{A}$ can be represented by only one parameter (e.g. a value of torque, a value of displacement), the definition is possible. Even if actions has more than one parameters, there are some methods for calculations.

However, we should not forget that the distortion between $\boldsymbol{v}$ and $\boldsymbol{c}$, is changed at different parts of a state-action map. Even if an action $a^* = \pi^*(s)$ can replaced by another action a' without distortion at the state s, it never means that the change from a^* to a' is distortion-free at any state s' that fulfills $\pi^*(s') = a^*$.

State-Value Distortion

Therefore, we propose a novel distortion measure. With this definition, clustering algorithms can be applied to any state-action map. When an action of a state $s \in \mathcal{S}$ on an uncompressed state-action map, is changed to another, $a \in \mathcal{A}$, the distortion by this change is measured by

$$d^{\pi}(s, a) = V^{\pi}(s) - \sum_{s' \in \mathcal{S}} \mathcal{P}^{a}_{ss'} \left[\mathcal{R}^{a}_{ss'} + V^{\pi}(s') \right], \tag{3.22}$$

where π is the policy that is given by the uncompressed state-action map. For state indexes and action indexes, the following representation:

$$d^{\pi_{\mathrm{MAP}}}(i, j) = d^{\pi}(s_i, a_j) \quad (i \in \mathcal{I}_{\mathcal{S}}, j \in \mathcal{I}_{\mathcal{A}}). \tag{3.23}$$

is suitable. We call each of them **a state-value distortion**.

In the clustering process, all kinds of distortion caused by changes of action are measured by the sum of the above distortion measure. The distortion that is caused by a change of vector is measured by

$$D^{\pi}_{\text{vector}}(\boldsymbol{v}, \boldsymbol{c}) = \sum_{i=0}^{N_{\varepsilon}-1} d^{\pi_{\text{MAP}}}(v_i, c_i), \tag{3.24}$$

where $\boldsymbol{v} = (v_0, v_1, \ldots, v_{N_{\varepsilon}-1}) \in \mathcal{V}, \boldsymbol{c} = (c_0, c_1, \ldots, c_{N_{\varepsilon}-1}) \in \mathcal{C}$. The distortion measure of a cluster $\mathcal{K} \subset \mathcal{V}$ is

$$D^{\pi}_{\text{cluster}}(\mathcal{K}) = \min_{\boldsymbol{c} \in \mathcal{C}} \sum_{\boldsymbol{v} \in \mathcal{K}} D^{\pi}_{\text{vector}}(\boldsymbol{v}, \boldsymbol{c}). \tag{3.25}$$

This equation also means that representative vector $\boldsymbol{c}$ of cluster $\mathcal{K}$ is represented by the following equation:

$$\boldsymbol{c} = \operatorname*{argmin}_{\boldsymbol{c}} \sum_{\boldsymbol{v} \in \mathcal{K}} D^{\pi}_{\text{vector}}(\boldsymbol{v}, \boldsymbol{c}). \tag{3.26}$$

The distortion of a VQ map π'_{MAP} that gives a policy π' is measured by

$$\begin{aligned} \mathcal{D}^{\pi}(\pi') &= \sum_{s \in \mathcal{S}} d^{\pi}(s, \pi'(s)) \\ &= \sum_{s \in \mathcal{S}} \left\{ V^{\pi}(s) - \sum_{s' \in \mathcal{S}} \mathcal{P}^{a}_{ss'} [\mathcal{R}^{a}_{ss'} + V^{\pi}(s')] \right\}, \end{aligned} \tag{3.27}$$

or

$$\mathcal{D}^{\pi_{\text{MAP}}}(\pi'_{\text{MAP}}) = \sum_{i=0}^{N-1} d^{\pi_{\text{MAP}}}(i, \pi'_{\text{MAP}}(i)). \tag{3.28}$$

When the vectors in the map π_{MAP} is classified into clusters $\mathcal{K}_i$ ($i = 0, 1, 2, \ldots, N_c - 1$), this equation is equal to

$$\mathcal{D}^{\pi_{\text{MAP}}}(\pi'_{\text{MAP}}) = \sum_{i=0}^{N_c-1} D^{\pi}_{\text{cluster}}(\mathcal{K}_i). \tag{3.29}$$

This distortion measure represents the loss of efficiency. Though this equation is different from Eq. (3.19), we use this equation for clustering due to the problem of calculation amount.

3.5 Example of Implementation

Here we implement a vector quantization algorithm for the state-action map so as to give an example of implementation.

3.5.1 Blocking

At first, we fix the manner of blocking. Here we cut a state-action map into congruent rectangles, which are regarded as vectors. Such a manner of blocking can be represented by the following transforms:

$$\begin{pmatrix} i_\nu \\ i_\varepsilon \end{pmatrix} = \begin{pmatrix} \nu(i) \\ \varepsilon(i) \end{pmatrix} = \begin{pmatrix} \lfloor i_x/w_x \rfloor + \sqrt{N}/w_x \lfloor i_y/w_y \rfloor \\ \lfloor i_x \% w_x \rfloor + w_x \lfloor i_y \% w_y \rfloor \end{pmatrix}, \tag{3.30}$$

where w_x and w_y are the numbers of intervals of each block along x-axis and y-axis respectively. Figure 3.7 illustrates an example of blocking with these transforms.

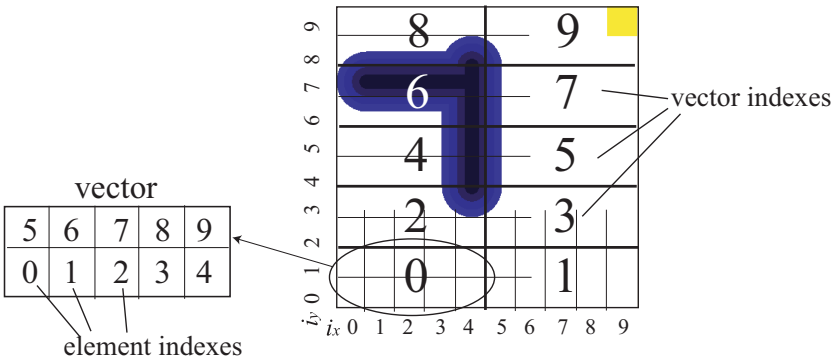


Fig. 3.7: Example of blocking ($N = 10^2$, $w_x = 5$, and $w_y = 2$)

The parameters w_x and w_y affect the success of compression. This issue is discussed in Chapter 4. Here we fix the blocking way as $(w_x, w_y) = (\sqrt{N}, 1)$. Therefore, Eq. (3.30) is simplified as

$$(i_\nu, i_\varepsilon) = (\nu(i), \varepsilon(i)) = (i_y, i_x)\,. \tag{3.31}$$

Then, the relation between the state index i and (i_ν, i_ε) can be described as

$$\mu(i_\nu, i_\varepsilon) = \mu(i_y, i_x) = i_x + i_y\sqrt{N}. \tag{3.32}$$

Each vector is defined as

$$\begin{aligned}\boldsymbol{v}_{i_\nu} &= \big(\pi_{\mathrm{MAP}}(s_{\mu(i_\nu,0)}), \pi_{\mathrm{MAP}}(s_{\mu(i_\nu,1)}), \pi_{\mathrm{MAP}}(s_{\mu(i_\nu,2)}), \ldots, \pi_{\mathrm{MAP}}(s_{\mu(i_\nu,N_\varepsilon-1)})\big) \\ &= \big(\pi_{\mathrm{MAP}}(s_{i_\nu\sqrt{N}}), \pi_{\mathrm{MAP}}(s_{i_\nu\sqrt{N}+1}), \pi_{\mathrm{MAP}}(s_{i_\nu\sqrt{N}+2}), \ldots, \pi_{\mathrm{MAP}}(s_{(i_\nu+1)\sqrt{N}-1})\big)\end{aligned} \tag{3.33}$$

from Eq. (3.15).

3.5.2 Clustering Algorithms

The vectors $\boldsymbol{v}_{i_\nu}$ $(i_\nu = 0, 1, 2, \ldots, \sqrt{N}-1)$ are classified into N_c clusters and the representative vector is computed in each cluster. Here, the pairwise nearest neighbor (PNN) algorithm [Equitz, 1989] is used for clustering. Moreover, the Lloyd iteration [Gersho, 1992] is used for enhancing the clustering result after the PNN algorithm.

Pairwise Nearest Neighbor Algorithm

A pseudo code of the PNN algorithm is shown in Fig. 3.8. The first step of this algorithm is to create N_ν clusters: $\mathcal{K}_i = \{\boldsymbol{v}_i\}$ $(i = 0, 1, 2, \ldots, N_\nu - 1)$. In a loop of the PNN algorithm, two of these clusters are combined to a cluster. Therefore, clusters are disappeared one by one at every loop. The loop is repeated until the number of clusters are larger than a target number: N_c. The pair of clusters that are combined is chosen based on the following value:

$$\delta \longleftarrow D^\pi_{\mathrm{cluster}}(\mathcal{K}_i \cup \mathcal{K}_j) - D^\pi_{\mathrm{cluster}}(\mathcal{K}_i) - D^\pi_{\mathrm{cluster}}(\mathcal{K}_j), \tag{3.34}$$

where $D^\pi_{\mathrm{cluster}}(\mathcal{K}_i \cup \mathcal{K}_j)$ denotes the distortion of the combined cluster $\mathcal{K}_i \cup \mathcal{K}_j$ on the assumption that $\mathcal{K}_i$ and $\mathcal{K}_j$ are combined. The pair of $\mathcal{K}_i$ and $\mathcal{K}_j$ that minimizes δ is combined in a loop.

The time complexity of this algorithm is $\mathrm{O}(MNN_\nu^2) = \mathrm{O}(MN^3/N_\varepsilon^2)$. Therefore, the smaller the number of elements in a vector is, the larger the computation time expands.

```
1:  K_i ⟵ {v_i} and c_i ⟵ v_i   (i = 0, 1, 2, ..., N_ν − 1)
2:  n ⟵ N_ν
3:  while n > N_c
4:      δ_min ⟵ ∞
5:      for every (i, j)  (i = 0, 1, 2, ..., n − 2; j = i + 1, i + 2, ..., n − 1)
6:          if K_i ≠ ∅ and K_j ≠ ∅  (∅ : null set)
7:              D^π_cluster(K_i ∪ K_j) ⟵ min_c Σ_{v∈K_i∪K_j} D^π_vector(v, c)
8:              c_tmp ⟵ argmin_c Σ_{v∈K_i∪K_j} D^π_vector(v, c)
9:              δ ⟵ D^π_cluster(K_i ∪ K_j) − D^π_cluster(K_i) − D^π_cluster(K_j)
10:             if δ < δ_min
11:                 δ_min ⟵ δ
12:                 i_min ⟵ i
13:                 j_min ⟵ j
14:                 c_min ⟵ c_tmp
15:     K_{i_min} ⟵ K_{i_min} ∪ K_{j_min}
16:     D^π_cluster(K_{i_min}) ⟵ D^π_cluster(K_{i_min} ∪ K_{j_min})
17:     c_{i_min} ⟵ c_min
18:     K_{j_min} ⟵ ∅
19:     n--
```

Fig. 3.8: Pairwise nearest neighbor algorithm [Equitz, 1989]

Lloyd Algorithm

After the PNN algorithm, we apply a Lloyd algorithm to an obtained VQ map. The pseudo code of a Lloyd algorithm is shown in Fig. 3.9. This algorithm repeats composition of clusters and computation of representative vectors alternately. In the composition of clusters, each vector is added in a cluster whose representative vector can give the smallest distortion toward the vector. The computation of representative vectors is simply based on Eq. (3.26). After some iteration, Δ in each process will stop reducing.

The time complexity of the Lloyd algorithm varies by implementation. If all of the distortions $d(s, a)$ are calculated and recorded on memory in advance, it becomes $\mathrm{O}(\iota N N_c)$. ι is the number of iteration.

```
composition of clusters:
/* c_j (j = 0, 1, 2, ..., N_c − 1) are already set.*/
1:  K_j ⟵ ∅ (j = 0, 1, 2, ..., N_c − 1)
2:  Δ ⟵ 0
3:    for each v_i (i = 0, 1, 2, ..., N_ν − 1)
4:      K_j ⟵ argmin_{K_j} D^π_vector(v_i, c_j)
5:      K_j ⟵ K_j ∪ {v_i}
6:      Δ ⟵ Δ + D^π_vector(v_i, c_j)
7:  return Δ
computation of representative vectors:
/* K_j (j = 0, 1, 2, ..., N_c − 1) are already set.*/
8:  Δ ⟵ 0
9:  for each K_j (j = 0, 1, 2, ..., N_c − 1)
10:   c_j ⟵ argmin_c Σ_{v∈K_j} D^π_vector(v, c) /*Eq.(3.26)*/
11:   Δ ⟵ Δ + Σ_{v∈K_j} D^π_vector(v, c_j)
12: return Δ
```

Fig. 3.9: Lloyd Algorithm [Gersho, 1992]

3.5.3 Obtained VQ Maps

Figure 3.10 illustrates exsamples of compression result. The $N = 100^2$ state-action map in (a) is compressed with two ways of blocking: $(w_x, w_y) = (10, 10), (100, 1)$. The number of representative vectors is 10. At first, the PNN algorithm in Fig. 3.8 is used for reducing the number of clusters from 100 to 10. The Lloyd algorithm in Fig. 3.9 is then applied to. Obtained VQ maps are illustrated in (b). (c) shows each policy that is obtained by each VQ map. For convenience of explanation, policies that are obtained from VQ maps as shown in (c) are called VQ maps in this thesis.

As shown in (b), a VQ map is composed of representative vectors and a table that indicates where each representative vector is adapted to. In a computer, representative vectors and the table are described as binary digit strings respectively and are called a codebook and a quantization table. The size of each VQ map is calculated as

$$\begin{aligned} L_{\mathrm{VQ}} &= N_\nu \lceil \log_2 N_c \rceil + N_\varepsilon N_c \lceil \log_2 M \rceil \\ &= 100 \lceil \log_2 10 \rceil + 100 \cdot 10 \lceil \log_2 4 \rceil \\ &= 400 + 2000 = 2400[\mathrm{bit}] \end{aligned} \tag{3.35}$$

from Eq. (3.18). Since the size of the uncompressed map with $N = 100^2$ is 20000[bit], the compression ratio is $1 : 0.12$.

From (c), changes of policy by lossy compression can be observed. Many a_{up} and a_{right} are replaced to each other in the part where no puddle exists. In this part, values of state-value distortion are small toward these changes. On the other hand, actions allocated around the puddle are not largely changed. By the use of state-value distortion, frequency of changes can be biased like this.

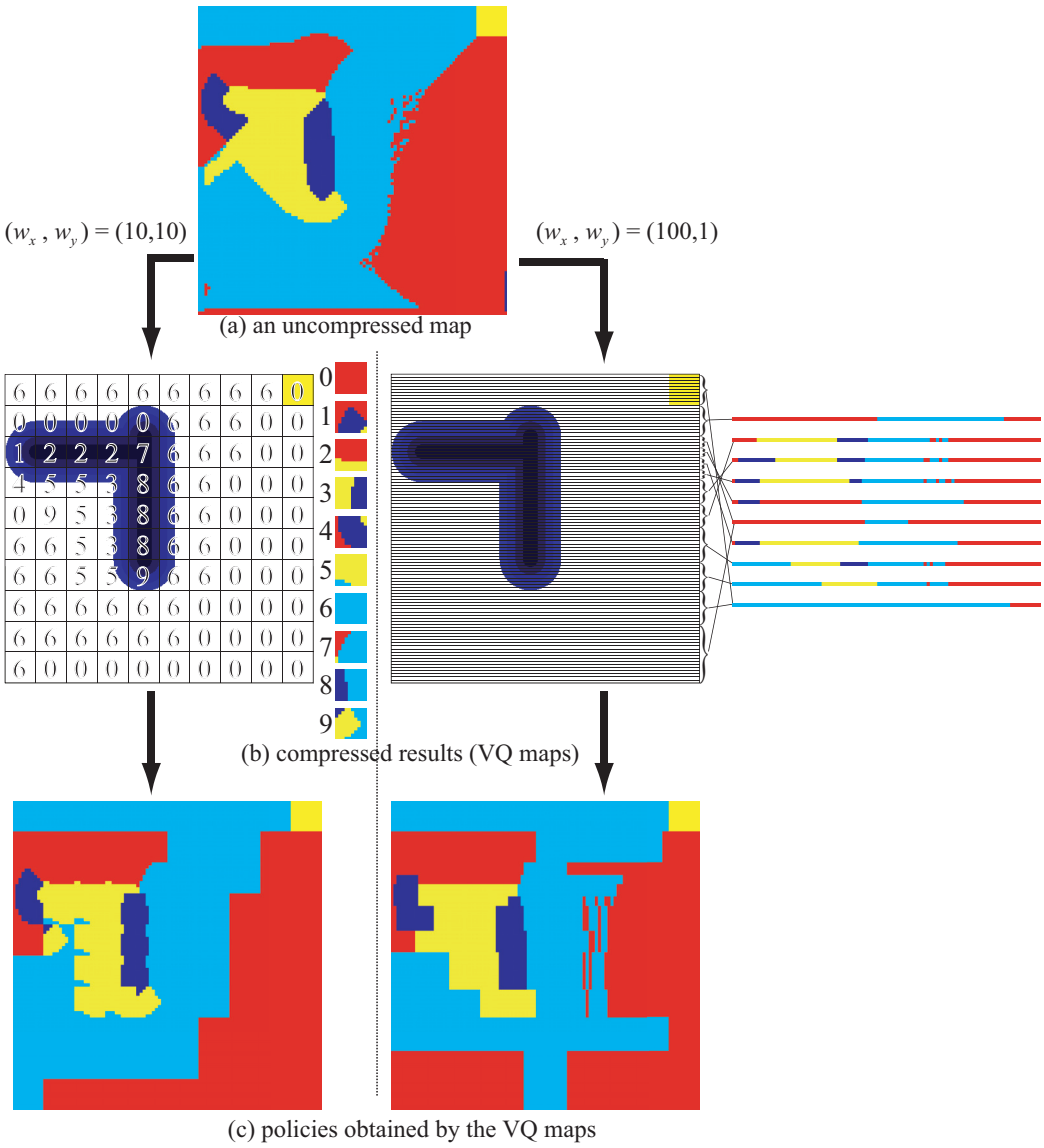


Fig. 3.10: Structure of VQ Maps

3.6 Evaluation with Puddle World Task

3.6.1 Comparison with Coarse Discretization

Here we compare the size and efficiency of pairs of VQ maps and uncompressed state-action maps that are created. If a VQ map is larger than a coarse uncompressed state-action map, the VQ map is useless.

In Fig. 3.11, size and efficiency of uncompressed maps and those of VQ maps are compared. In Fig. (b)-(f), there are data points of VQ maps under the dot-lines that connect data points of the uncompressed maps. We can consider the VQ maps that have such data points as superior to the uncompressed maps.

We show three pairs of an uncompressed state-action map and a VQ map that is smaller and more efficient than the uncompressed map in Fig. 3.12. The three maps in (a-1), (b-1), and (c-1) are uncompressed state-action maps and the other three maps in (a-2), (b-2), and (c-2) are VQ maps that are compared to the above maps respectively. Each VQ map is compressed from a state-action map that has larger N than each of the above map.

In Table 3.1, size and efficiency J of every map is illustrated. The compression ratio is calculated as the ratio of sizes between an uncompressed map and a VQ map that are in the same row. This ratio is called an **effective compression ratio**. The effective compression ratio is defined as the ratio of the size of a VQ map to that of a uncompressed state-action map that is equal or less efficiency than the VQ map. When an effective compression ratio of a VQ map is more than one toward any state-action map whose efficiency is more than that of the VQ map, the VQ map is meaningless. On the other hand, we call a compression ratio as a **self-compression ratio** when it denotes the ratio of the size of a VQ map to that of its original state-action map.

The effective compression ratio of each VQ map in the table is less than 0.7. Though the ratios are not high, we can assuredly find the VQ maps that are more effective than the state-action maps. The major difference between the uncompressed maps and the VQ maps is the difference of the parts of a_{up} and a_{right} (e.g. A-D marked in the figure). In these parts, there is no difference between the choice of a_{up} and a_{right}. Since the state-value distortion can consider this nature, the drastic change is possible. The change then reduces the redundancy of a state-action map.

From the pair of (b-1) and (b-2), moreover, we can recognize that a kind of variable bit allocation is done in the VQ map. In the upper edge of (b-2), which is marked as E, a_{down} is allocated. This action can make the agent avoid collisions with the upper edge of the world and seems to contribute to efficiency enhancement. In (b-1), on the other hand, a_{right} is allocated in the part due to the coarse discretization. As well as cutting the redundant part of a state-action map, the VQ algorithm can leave such a subtlety that is brought by the fine discretization.

Fig. 3.13 highlights the difference of a state-action map and a VQ map from the viewpoint of behavior of the agent. In each figure, the frequency of visit by the agent at the trials for evaluation is illustrated by gray scale. Every figure is divided into 100^2 cells regardless of the coarseness of the map. The visit frequency of each cell is then counted at the trials. The darker the cell is drawn, the more frequent the agent visits. The dark part can be likened to animal trails. (i) and (iii) are the images of trails obtained by the state-action maps with $N = 20^2$ and $N = 40^2$ respectively. (ii) is the image of trails by the VQ map that is shown in Fig. 3.12(a-2). The efficiency of the VQ map is better than the state-action maps with $N = 20^2$ as mentioned before.

When these three images are compared, we notice that the compression is something more than reduction of resolution, which is shown in lossy compression of images. When we compare (i) and (iii), there is no difference but the roughness of trails. However, (ii) illustrates mode change of behavior by the compression. By a partial change of the map at part B in Fig. 3.12(a-2), the path of the agent is changed largely.

Table 3.1: Comparison of Size and Efficiency of Maps in Fig. 3.12

uncomp. map			VQ map			effective
Fig.	size [bit]	$-J$[step]	Fig.	size [bit]	$-J$[step]	comp. ratio
(a-1)	800	20.87	(a-2)	520	20.79	1:0.65
(b-1)	20k	20.35	(b-2)	13.8k	20.32	1:0.69
(c-1)	80k	20.30	(c-2)	53.6k	20.29	1:0.67

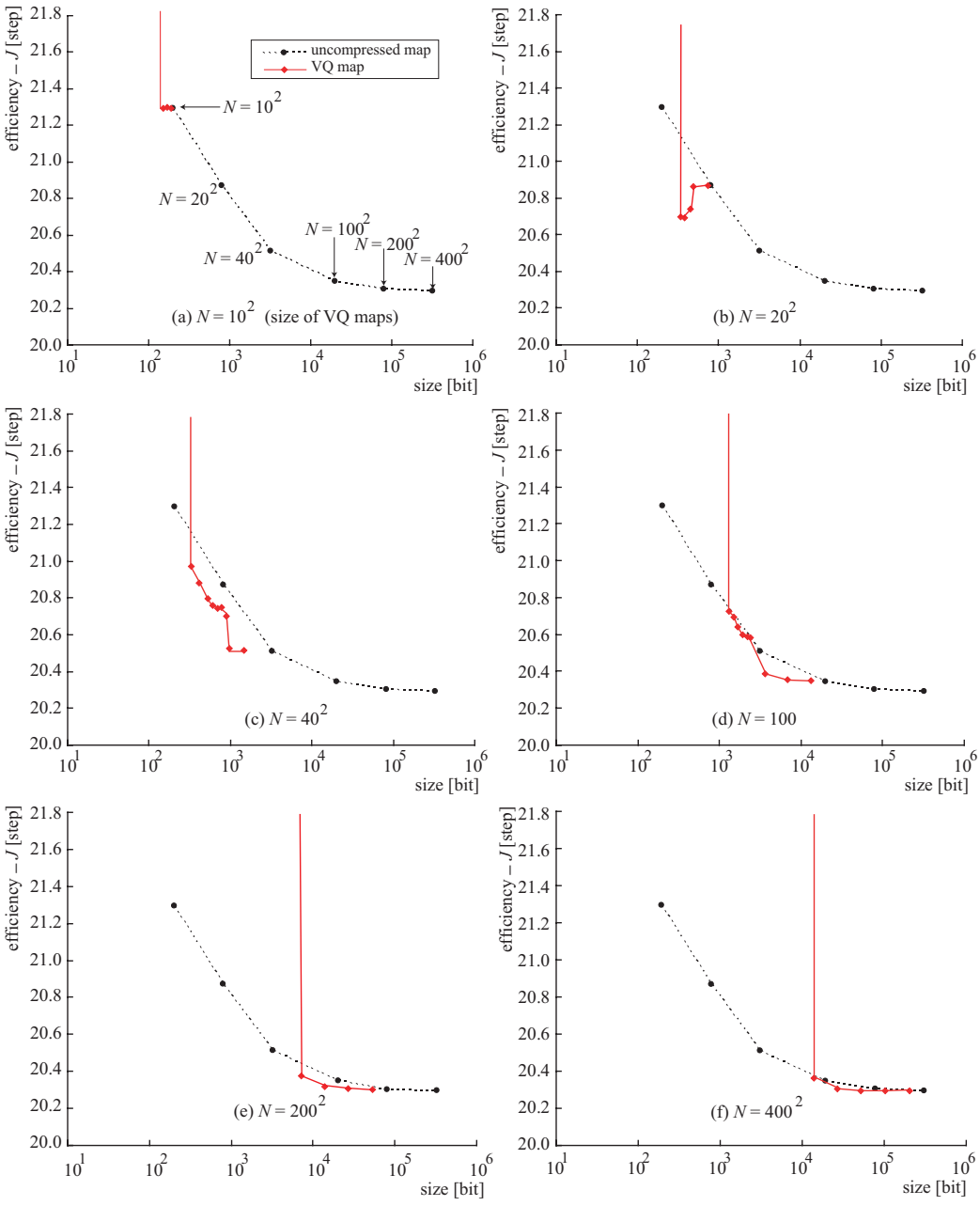
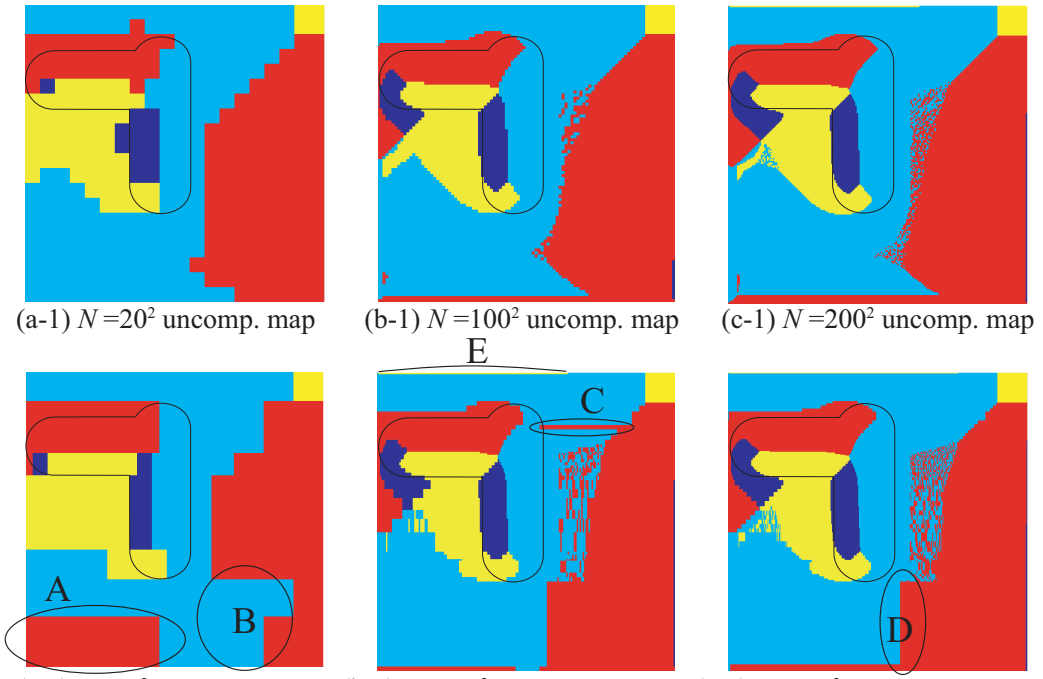


Fig. 3.11: Comparison between VQ maps and Uncompressed Maps



(a-1) N =20^2 uncomp. map (b-1) N =100^2 uncomp. map (c-1) N =200^2 uncomp. map

(a-2)N =40^2, Nc =5 VQ map (b-2)N =200^2, Nc =32 VQ map (c-2)N =400^2, Nc =64 VQ map

Fig. 3.12: Uncompressed Maps and Efficiency Equivalent VQ maps

(i) uncompressed map (N =20^2 uncomp. map)

(ii) VQ map N =40^2, Nc =5 VQ map

(iii) original map of (ii) (N =40^2 uncomp. map)

Fig. 3.13: Trails of The Agent ((i): (a-1) in Fig. 3.12, (ii): (a-2) in Fig. 3.12, (iii): state-action map with $N = 40^2$)

3.6.2 Comparison with Tree Structure

We finally compare the VQ method to another compression method that is based on a binary-tree representation, which is another possible structure for representing policies compactly. We name this method a binary-tree compression method of state-action maps.

Creation of Binary-Trees

As shown in Fig. 3.14, a binary-tree has nodes, branches, and leaves. This method starts from a node, which has two branches: e^-,e^+. They cover $0 \leq x < 1/2$ area and $1/2 \leq x < 1$ area respectively. Each branch has a leaf in which an action is written. Every state-value distortion $d(s, a)$ has been calculated from a state-value function on a fine look-up table.

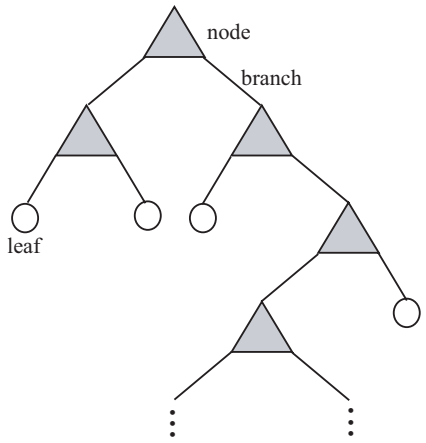


Fig. 3.14: Binary-Tree Structure

From that state, the tree grows with a procedure that changes a leaf into a node. The leaf that is changed is chosen from all leaves based on the state-value distortion. To decide the leaf that is changed, the method calculates every sum of $d(s, a)$ in the area of every branch that has a leaf. A branch that has the largest sum is attached a node instead of the leaf. When the area of a branch is not larger than the size of a discrete state on the state-value function, however, the branch cannot have a node. This restriction can prevent excess splitting of the state-space.

When a node is added to the tree, the area that is covered by the node must be split and allocated to both leaves. If the area can be split arbitrarily,

the node must use some bits to memory the way of splitting. We choose to fix a splitting rule so as to save the amount of memory. The rule is as follows:

- when the area is a square, it is split evenly along y-axis;
- otherwise it is split evenly perpendicular to long edges.

By this rule, the area that is covered by every leaf can be calculated only with the structure of the tree.

After that, unnecessary nodes are cut. At first, a node that has an identical action on its both branches can be erased. By an algorithm, every unnecessary node is replaced by a leaf. This procedure is iterated until unnecessary nodes exist. Secondly, a node can be erased when another node has identical contents. In this case, it can be erased when the branch that points the node is reconnected to the other node as shown in Fig. 3.15. This procedure is also iterated until unnecessary nodes exist.

We show two examples that are policies on the tree-structures created by the above method in Fig. 3.16. The $N = 400^2$ state-value function was used for computing the distortion measures. The policy in Fig. (a) is composed of 277 leaves ($N_{\text{node}} = 172$). Incidentally, there was 300 nodes before the cut of unnecessary nodes, 276 nodes before the first cut of unnecessary nodes. Though the number of leaves are smaller than that of states on the state-action map with $N = 20^2$ (Fig. 2.10(b)), its granularity is much finer. When the number of leaves is 1913, which is incidentally reduced from $2,000$ nodes, the behavior for avoiding the edge of the world is generated at A and B of the figure. At the first face, compression by binary-tree is successful.

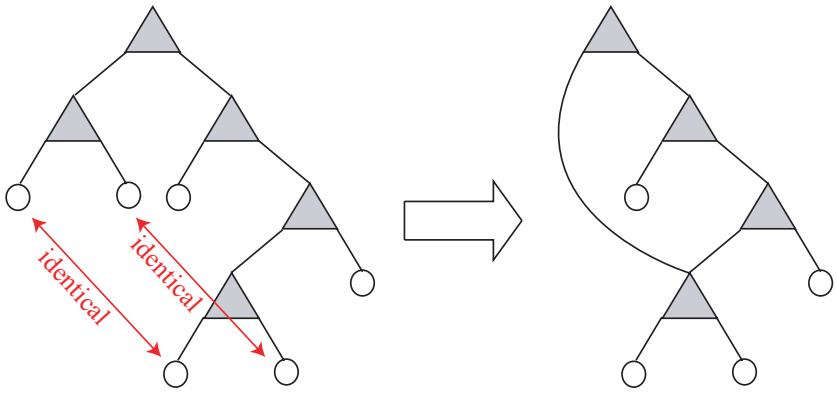


Fig. 3.15: Reconnection of a Branch

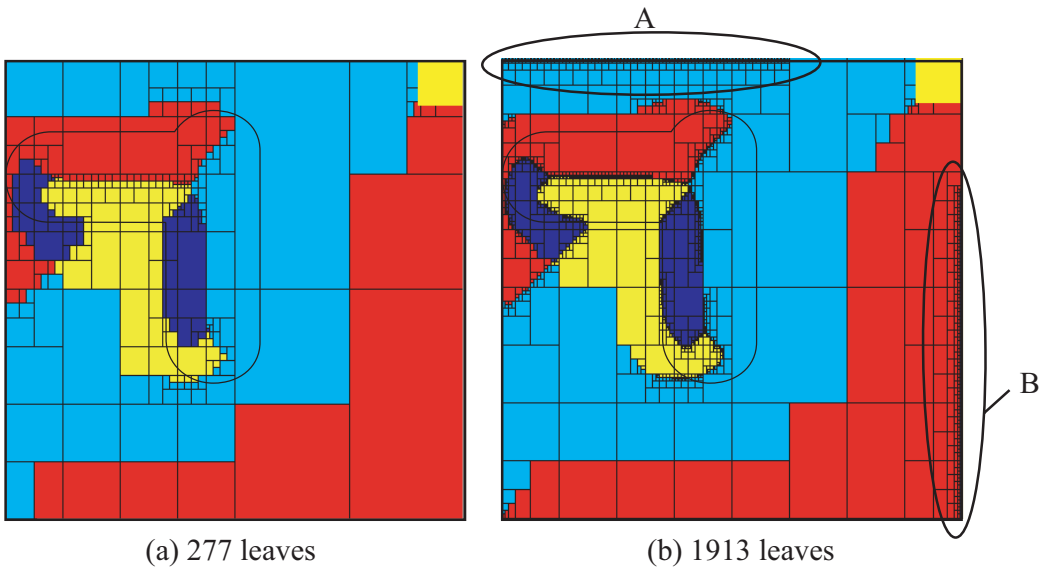


Fig. 3.16: Policies on Binary-Trees

Coding of Binary-Trees

Then we discuss memory use of binary-trees. In a computer, the branches are implemented as pointers (address variables). A node has two pointers and each of them points the address of another node or a leaf. Since an address variable consumes 32 bits or 64 bits on a 32-bit or 64-bit computer, address variables of a system should not be used for addressing.

Therefore, we code a tree with the following format.

- A node has two unsigned integer variables. The size of each variable is $\lceil \log_2(N_{\text{node}} + M) \rceil$[bit], where N_{node} and M denote the number of nodes and actions respectively.
- The first integer relates to the area whose position on x-axis or y-axis is far from the origin of xy-plane compared to the other area. The second integer relates to the other area.
- An array with $2N_{\text{node}}\lceil \log_2(N_{\text{node}} + M) \rceil$[bit] is prepared.
- The value i of each variable is defined as follows.
  - When $i < M$, the variable is regarded as a leaf and a_i is the action of the leaf.

– $i \geq M$, the variable is regarded as a pointer to another node. The variables of the other node are written from $2(i-M)\lceil \log_2(N_{\text{node}}+M)\rceil$th bit in the array.

By this format, the number of bits for pointers can be changed from a unique number of bits of a CPU to $\lceil \log_2(N_{\text{node}}+M)\rceil$[bit].

Comparison Result

We have measured the efficiency J of some binary-tree policies. N_{node} before the reduction of nodes is chosen form $n \cdot 10^m$ ($n = 1, 2, \ldots, 9; m = 0, 1, 2, \ldots$). Since the cut of unnecessary nodes is applied to each map, the actual value of N_{node} becomes smaller than the chosen one. We use all of the state-value functions with $N = 10^2, 20^2, 40^2, 100^2, 200^2$, and 400^2.

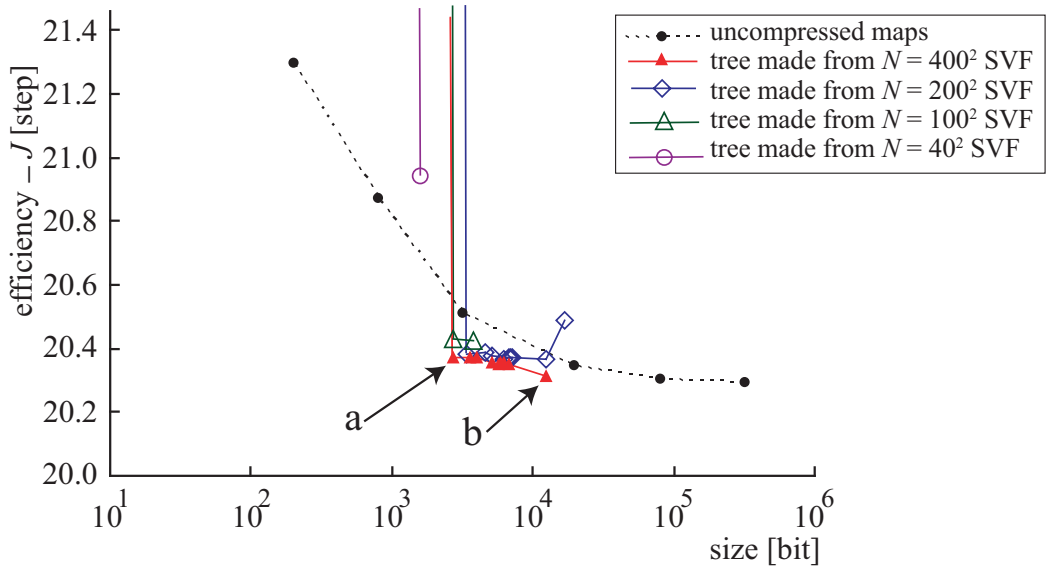


Fig. 3.17: Evaluation of Tree-Structure Policies

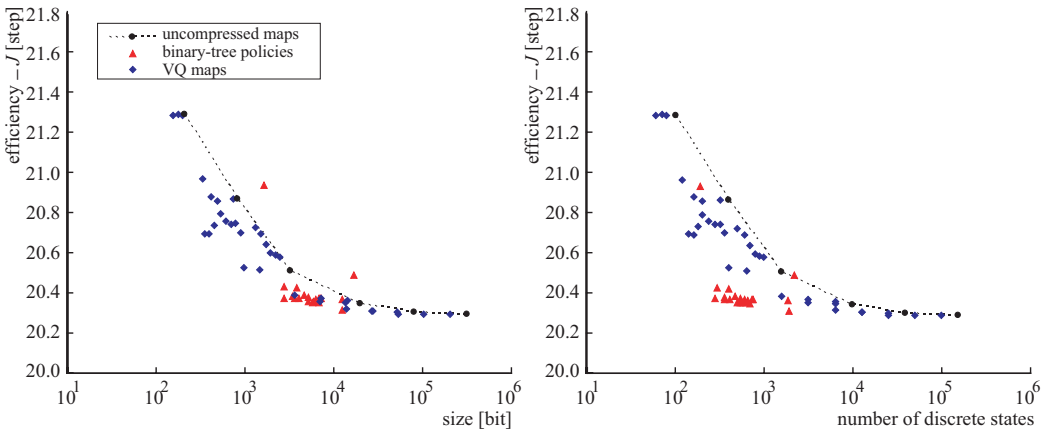
The evaluation results are shown in Fig. 3.17 with the results of the state-action maps. SVF denotes a state-value function. Points "a" and "b" in the figure relate to the policies in Fig. 3.16(a) and (b) respectively. When $N = 10^2, 20^2$, we cannot obtain unbroken policies.

The binary-tree policies "a" can be obtained in 30[s] with a 1.5GHz Pentium M CPU. On the other hand, the VQ maps with $N_c = 32$ from $N = 400^2$, for example, is obtained in 16[min]. Therefore, binary-tree policies are much superior to VQ maps in the viewpoint from time consuming. However, we should note that this difference of magnitude becomes small when time consuming to create the state-action map with $N = 400^2$ (21[min]) is added to them.

Figure 3.17 suggests the following things:

- almost all of the binary-tree policies are superior to the state-action maps, and
- the state-value function for calculating state-value distortions should be fine.

The former means that the binary-tree compression method can be a good competitor of the VQ method. In a part of the graph, some binary-tree policies are smaller than some VQ maps whose efficiencies are comparable level. However,the latter will be a disadvantage compared to the VQ method because the VQ method can compress state-action maps in keeping with their granularity. As shown in the figure, the size of a binary-tree does not relate to the size of state-value functions.



(a) size - efficiency graph (b) # of states - efficiency graph

Fig. 3.18: Comparison of VQ Maps and Binary-Tree Policies

In Fig. 3.18(a), all evaluations of the VQ maps in Fig. 3.11 and those of binary-tree policies are plotted. VQ maps have a range of different sizes

and efficiencies, while the sizes of binary-tree polices are more than 10^3[bit]. The reason can be understood from Fig. 3.18(b), where the horizontal axis of the graph is not the size but the number of discrete states. The number of discrete states means the number of leaves in the case of the binary-trees, and the number of elements in a codebook in the case of VQ maps. This figure suggests that the efficiency of some binary-tree policies are superior to the state-action maps if we consider not the size but the number of discrete states. In other words, the size on memory of a binary-tree policy becomes about ten times larger than the number of discrete states. In contrast, the relation between the state-action maps and the VQ maps changes little in appearance. It means that the VQ maps do not use large amounts of memory for representing the structure of vector quantization.

In Table 3.2, we choose some policies so as to illustrate the excess use of memory of binary-trees. The bits for actions are computed as 2[bit] are used for representing an action index in every structure. In the case of state-action maps, all memory is used for recording actions. Though some equations for relating the continuous state-space and the discrete state-space are required, they are also used for the VQ maps and binary-tree policies. In the case of VQ maps, quantization tables can be regarded as the data for composing the structure of VQ maps. For recording quantization tables, $20 - 1.5$[%] of memory is used in the VQ maps on this table. On the other hand, binary-tree policies are composed of pointers. Action indexes are recorded on some gaps of the pointers. Even though we have applied the node reduction procedures, the percentage of memory for actions are 30[%] at most. It may not be an effective usage of memory.

Table 3.2: Breakdown of Use of Memory

structure	parameters	total size [bit]	memory for actions [bit]	[%]
binary-tree	$N_{\text{nodes}} = 172$ (277 leaves)	2,752	554	20.1
	$N_{\text{nodes}} = 612$ (1,913 leaves)	12,240	3,826	31.3
VQ	$N = 400^2, N_\varepsilon = 400, N_c = 16$	14,400	12,800	88.9
	$N = 400^2, N_\varepsilon = 400, N_c = 256$	208,000	204,800	98.5
	$N = 10^2, N_\varepsilon = 10, N_c = 6$	150	120	80.0
	$N = 10^2, N_\varepsilon = 10, N_c = 8$	190	160	84.2
state-action map	N	$2N$	$2N$	100.0

3.7 Discussion

In this chapter, we have proposed the vector quantization method for state-action maps. Use of the state-value distortion is the significant feature of this method. That is because the compression can be formulated in the concept of optimal control problem. The VQ method is not a simple compression method for data, but an optimization method that reuses the state-value function for simultaneous pursuit of high ratio compression and small efficiency loss.

Evaluation Result

As shown in the comparison with the binary-tree structure, use of memory of VQ maps is very economical. In the case of VQ maps, more than half of memory is used for representing codebooks, while more than half of memory is used for representing the structure at the binary-tree compression. Incidentally, if the reconnection procedure of branches is not applied, the percentage of memory for representing tree-structure is roughly doubled. Another problem of the tree-structure is that we cannot create efficient binary-tree policies from coarse state-value functions. The method can create binary-tree policies that are similar to high-resolution state-action maps from a state-value function with $N = 400^2$ as shown in Fig. 3.16. However, the method can no effective policy from the state-value functions with $N \leq 40^2$. Value iteration with a coarse look-up table is rather acceptable when we want to obtain a small size policy. Moreover, the VQ method can reduce the amount of memory from the coarse state-action map.

Rare Efficiency Enhancement by Compression

In Fig. 3.11(b), we can see that the values of efficiency of some VQ maps are better than the uncompressed state-action map with. We should explain about this phenomenon. In Fig. 3.19, we show the state-action map with $N = 20^2$ and two VQ maps that are compressed from the state-action map. Though the VQ map in (b) is smaller than the uncompressed state-action map in (a), its efficiency is better than that of the map in (a). Though (c) is the smallest map in the three maps, the efficiency is the best. As shown in this figure, some actions that are chosen for avoidance of puddle are omitted in the process of compression. It seems that these omissions enhance the efficiency.

This result is just an accident. However, it shows that the value iteration algorithm does not always give the best policy for coarse discretization. The reason is that the value iteration algorithm excludes the information of where the agent likely exists in a discrete state. When a state transition probability $\mathcal{P}^a_{ss'}$ is calculated, the probability distribution of where the state $\boldsymbol{x}$ exists in s is assumed as a uniform distribution in s. This assumption is not different from the actual probability distribution when the discretization is fine. However, it is not always true under the coarse discretization.

For example, we refer to the discrete state whose action is changed from a_{up} in (a) to a_{right} in (b). When the agent visits this discrete state, it seems that the distribution of $\boldsymbol{x}$ in this state s is biased toward the upper part on paper. Before the agent visits s, it chooses a_{up} if it comes close to the puddle. If the agent comes close to the puddle again, it chooses a_{up} again. That behavior can be imagined from the maps both in (a) and (b). As a result, the agent is apart from the puddle with high probability before it visits s. However, the state-action map in (a) makes the agent choose a_{up} though there is a small probability that the agent enters the puddle. In the value iteration algorithm, we have no choice but to discretize the state space finer if we want to avoid the loss of steps.

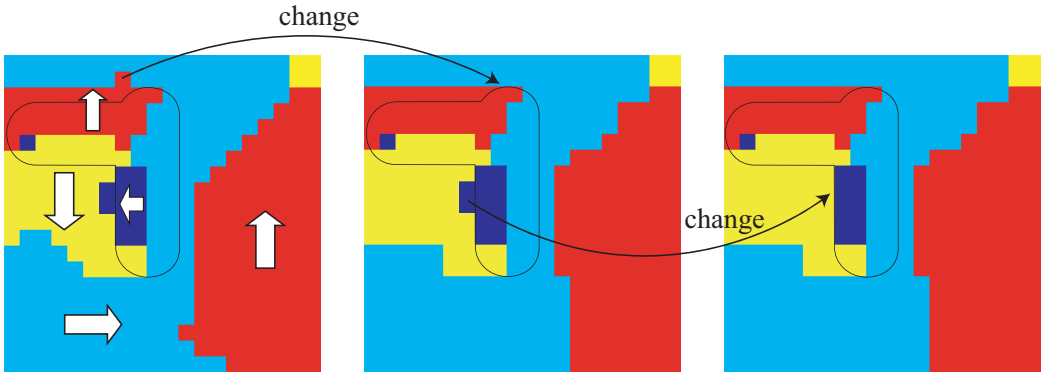


Fig. 3.19: VQ Maps ((b) and (c)) that are Better than Uncompressed Map (a).

In the process of compression from (a) to (b), a_{up} of the discrete state is changed to a_{right}. The enhancement of efficiency is a rare accident and it

cannot be expected as mentioned above. On the other hand, it is also quite true that this change occurs because the state-value distortion $d(s, a_{\text{right}})$ of this state s is small. The changes of actions from (b) to (c) occurs thanks to the appropriate values of state-value distortion too.

Chapter 4

Techniques for Quick and Efficient Compression

In this chapter, we propose some techniques for enhancing *efficiency per bit* of VQ maps. Though the VQ method proposed in Chapter 3 can be successfully applied to the puddle world task, there is room for reducing the sizes of the VQ maps and for enhancing their efficiency. If the number of discrete states is large, moreover, we have to consider the computing time for compression.

This chapter is composed of four sections. In Sec. 4.1, we point out a cause of performance reduction of VQ maps. We try applying value iteration to VQ maps so as to solve this problem. In Sec. 4.2, we try to find suitable ways of blocking before compression. Entropy functions are utilized for this attempt. In Sec. 4.3, the redundancy of VQ maps is reduced by one more vector quantization. Partitioning technique, which reduces calculation amount of some clustering methods is then introduced. Both of these methods change the format of VQ maps to tree structures. We conclude this section with discussion about the effectiveness of the techniques in Sec. 4.4.

4.1 Value Iteration for VQ Map

4.1.1 A Special Problem to Build VQ Maps

Equation (3.19) implies the difficulty of state-action map compression because a state-value function is globally changed by a change of just one action. This problem is related to the fact that any state-action map is a generator of state-action sequences. A VQ map should not generate any strange state-action sequence even if many actions in the map are changed to others. This request is much severe than that for data compression of image and sound.

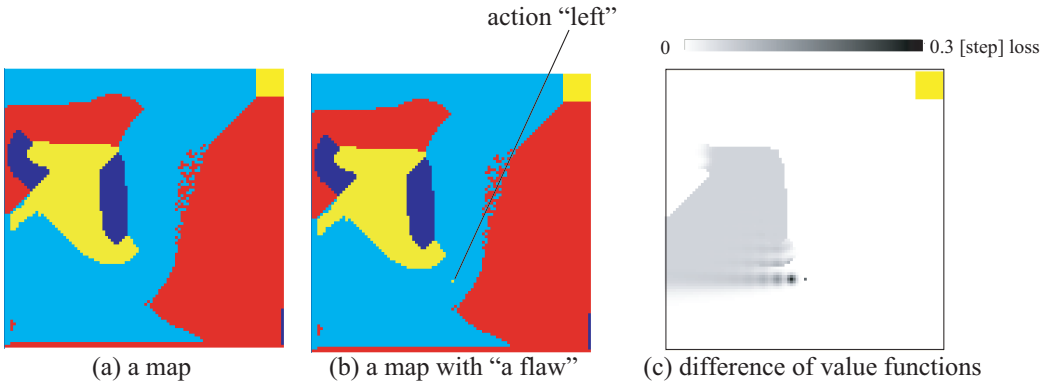


Fig. 4.1: Change of Value Function ($N = 100^2$)

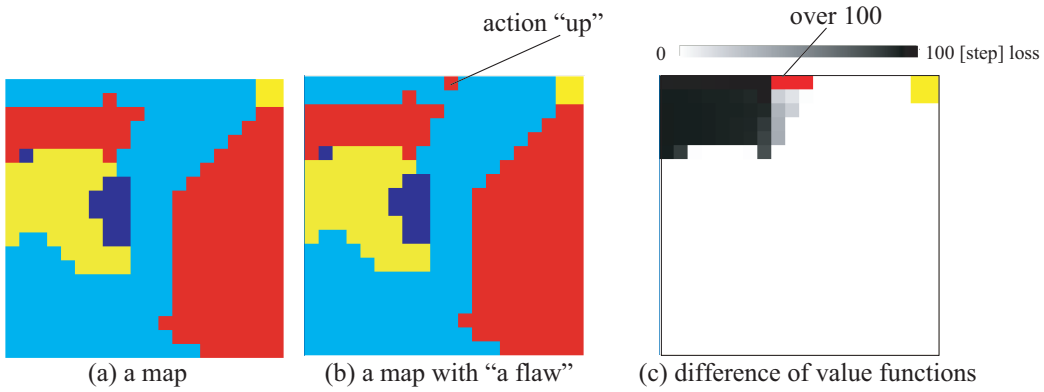


Fig. 4.2: Change of Value Function (a serious case, $N = 20^2$)

For example, we compare the state-value functions of the maps in

Fig. 4.1. The map in Fig. (a) is the same one with Fig. 2.9(d). In the map shown in Fig. (b), a right action on the map in Fig. (a) is changed to a left action. We compute the state-value functions of these maps by using the policy iteration algorithm. The time complexity of the policy iteration algorithm is $O(\iota NN')$. The difference of the state-value functions is illustrated in Fig. (c) by gray scale. As shown in Fig. (c), the loss of values occurs not only at the state where the action is changed, but also the others that are located upstream of the state. In the case shown in Fig. 4.2, the loss of values is more serious than that in Fig. 4.1. As with Fig. 4.1, the map in Fig. (a) is identical with (b) except that an action is changed. In this case, the changed action makes the robot collide with the wall, and the robot cannot escape from the state. What is worse is that the upper left part of this map becomes useless due to one change of action.

If we assume that a state-action map is compressed by the computer that is used for creating the map, computing resources for compressing are identical with those for value iteration. As mentioned in Sec. 2.2, the time complexity for value iteration is $O(\iota MNN')$. The complexity for compression should not be more than this order.

It means that we can use the policy iteration algorithm for evaluating ΔJ only several times. That is because the order of its time complexity is $O(\iota NN')$. Since a VQ map can be defined by ν, ε and ω, VQ maps exist by an increment of their combinations. A feasible combination must be found within the order of computation.

Figure 4.3 illustrates a case where the efficiency loss is much larger than the loss that is calculated with the state-value distortion. The maps shown in Fig. 4.3 are modified from the map in Fig. 4.2(a). The losses of values are illustrated under the maps respectively. We notice that the efficiency loss of the map in (c) is much larger than the sum of losses of the maps in (a) and (b).

If we try to consider this problem at a process of vector quantization, it causes the exponential increase of computing time. That is because the state-value function must be evaluated by the policy iteration algorithm at every time an action index is changed in the process of VQ.

4.1.2 Value Iteration after Vector Quantization

Therefore, we propose value iteration algorithms that optimize a VQ map after compression. In these algorithms, actions in representative vectors

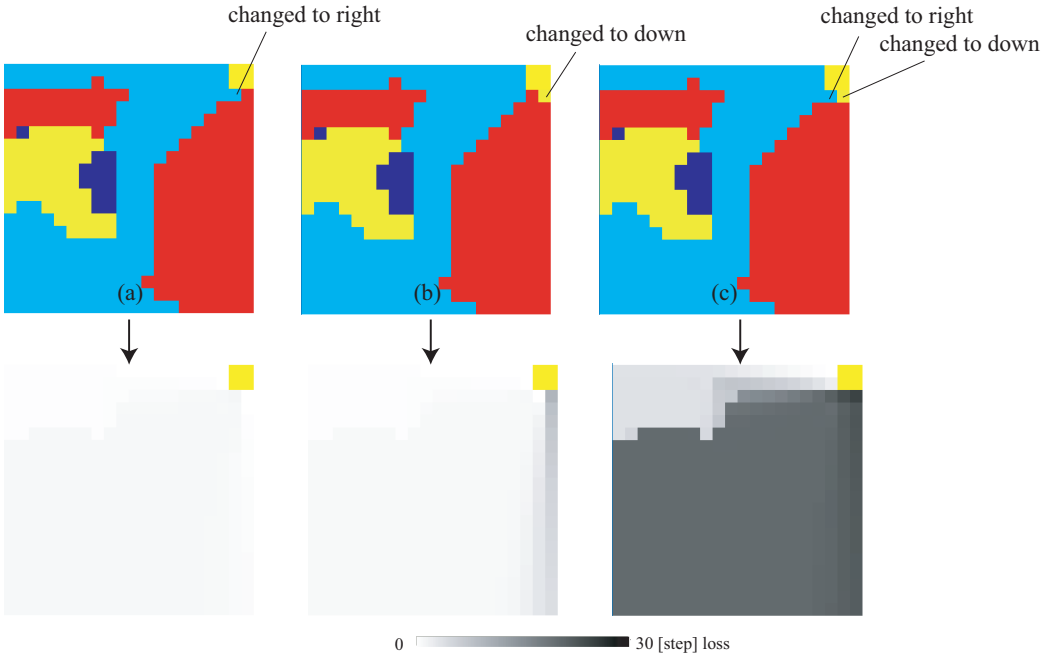


Fig. 4.3: Synergic Efficiency Loss by Change of Actions

are changed for optimization with no change of clustering result. In other words, they compute the optimal policy under the constraint of clustering.

Differently from the normal value iteration algorithm, the state-value function sometimes does not converge due to the constraint of clustering. However, we do not care about this issue at first. In this case, the code of the algorithm can be implemented as the pseudo code shown in Fig. 4.4. In Line 7, this algorithm chooses the action that maximizes of the sum of values at the states that are related to i_εth element of representative vector $\boldsymbol{c}_{i_\nu}$. In Line 11, the values of the states are recomputed since the action of the states are changed to a_{new}.

If there are N_ν clusters that contain one vector respectively, this algorithm is not different with the value iteration algorithm in Fig. 2.5. In this case, the state-value function may converge. In the other cases, there are no assurance of convergence.

When the increase of $\sum_{s\in\mathcal{S}} V(s)$ is prohibited at a sweep, the state-value function decreases monotonically. If the algorithm in Fig. 4.4 does not

```
1:  V ⟵ V*
2:  repeat
3:      Δ ⟵ 0
4:      for every (i_c, i_ε) ∈ (I_C, I_E)
5:          I_tmp1 ⟵ {i_ν | i_ν ∈ I_V, v_{i_ν} ∈ K_{i_c}}
6:          I_tmp2 ⟵ {i = μ(i_v, i_ε) | i_v ∈ I_tmp1, i_ε ∈ I_E}
7:          a_new ⟵ argmax_{a∈A} Σ_{i∈I_tmp2} Σ_{s'∈S} P^a_{s_i s'} [R^a_{s_i s'} + V(s')]
8:          c_(i_c,i_ε) ⟵ index of a_new
9:          for every i ∈ I_tmp2
10:             v ⟵ V(s_i)
11:             V(s_i) ⟵ Σ_{s'∈S} P^{a_new}_{s_i s'} [R^{a_new}_{s_i s'} + V(s')]
12:             Δ ⟵ max(|v − V(s_i)|, Δ)
13: until Δ < Θ (a small number)
```

Fig. 4.4: Value Iteration for VQ Map

converge, this idea should be introduced. The other algorithm, which reflects this idea, is shown in Fig. 4.5. In this algorithm, the action and the state-value function are changed only when the new value of every state is not less than the old value.

4.1.3 Evaluation

Though other algorithms are imaginable, we examine the algorithms in Fig. 4.4 and 4.5. The way of evaluation is identical with that in Sec. 2.3.4.

Figure 4.6 shows the simulation results with the simulation results of the uncompressed maps. The evaluations of *VQ maps before value iteration* in Fig. 4.6 are the data in Fig. 3.11. The data of *VQ maps after value iteration* is obtained by one of the value iteration algorithms in this section. The better one is drawn in these graphs.

From Fig. 4.6, we can verify that the value iteration algorithm enhances the limit of compression ratio in all granularity. Figure 4.7 illustrates two VQ maps for example. One of them in (a) is not applied the value iteration algorithm. The other, shown in (b), is the VQ map obtained by value iteration of the map in (a). As shown in (a), the states on the left side

$V \longleftarrow V^*$
policy evaluation (until convergence or a fixed times)
repeat
  $\Delta \longleftarrow 0$
  for every $(i_c, i_\varepsilon) \in (\mathcal{I}_\mathcal{C}, \mathcal{I}_\mathcal{E})$
    $\mathcal{I}_{\mathtt{tmp1}} \longleftarrow \{i_\nu | i_\nu \in \mathcal{I}_\mathcal{V}, \boldsymbol{v}_{i_\nu} \in \mathcal{K}_{i_c}\}$
    $\mathcal{I}_{\mathtt{tmp2}} \longleftarrow \{i = \mu(i_v, i_\varepsilon) | i_v \in \mathcal{I}_{\mathtt{tmp1}}, i_\varepsilon \in \mathcal{I}_\mathcal{E}\}$
    $a_{\mathtt{new}} \longleftarrow \underset{a \in \mathcal{A}}{\operatorname{argmax}} \sum_{i \in \mathcal{I}_{\mathtt{tmp2}}} \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{s_i s'} \left[\mathcal{R}^a_{s_i s'} + V(s')\right]$
    $V_{\mathtt{tmp}}(s_i) \longleftarrow \sum_{i \in \mathcal{I}_{\mathtt{tmp2}}} \sum_{s' \in \mathcal{S}} \mathcal{P}^{a_{\mathtt{new}}}_{s_i s'} \left[\mathcal{R}^{a_{\mathtt{new}}}_{s_i s'} + V(s')\right]$
    if $V_{\mathtt{tmp}}(s_i) \geq V(s_i)$ for every $i \in \mathcal{I}_{\mathtt{tmp2}}$
      $c_{(i_c, i_\varepsilon)} \longleftarrow c_{\mathtt{tmp}}$
      $\Delta \longleftarrow \max(|V_{\mathtt{tmp}}(s_i) - V(s_i)|, \Delta)$ for every $i \in \mathcal{I}_{\mathtt{tmp2}}$
      $V(s_i) \longleftarrow V_{\mathtt{tmp}}(s_i)$ for every $i \in \mathcal{I}_{\mathtt{tmp2}}$
until $\Delta < \Theta$ (a small number)

Fig. 4.5: Value Iteration for VQ Map 2

of the goal are allocated a_{up} though a_{right} should be allocated there. The states are connected with other states by VQ. That is because the distortion measure of a_{up} calculated by D^π_{vector} is better than that of a_{right}. After the value iteration, a_{right} is allocated to the states and the VQ map is corrected.

This algorithm is practically useful only when $N \leq 20^2$. As shown in Fig. 4.6(b), two VQ maps with $N = 20^2$ are smaller and more effective than the uncompressed map with $N = 10^2$. In Fig. (a), there are VQ maps whose sizes are smaller than the uncompressed map. $N = 10^2$ is the smallest number of discrete states if no discrete state that contains both final state and non-final state is allowed. VQ can get rid of the limitation that exists for convenience of DP. When $N \geq 40^2$ (Fig. (c)-(f)), however, the number of VQ maps that are more effective than the uncompressed maps with coarse discretization is not increased by the value iteration algorithms.

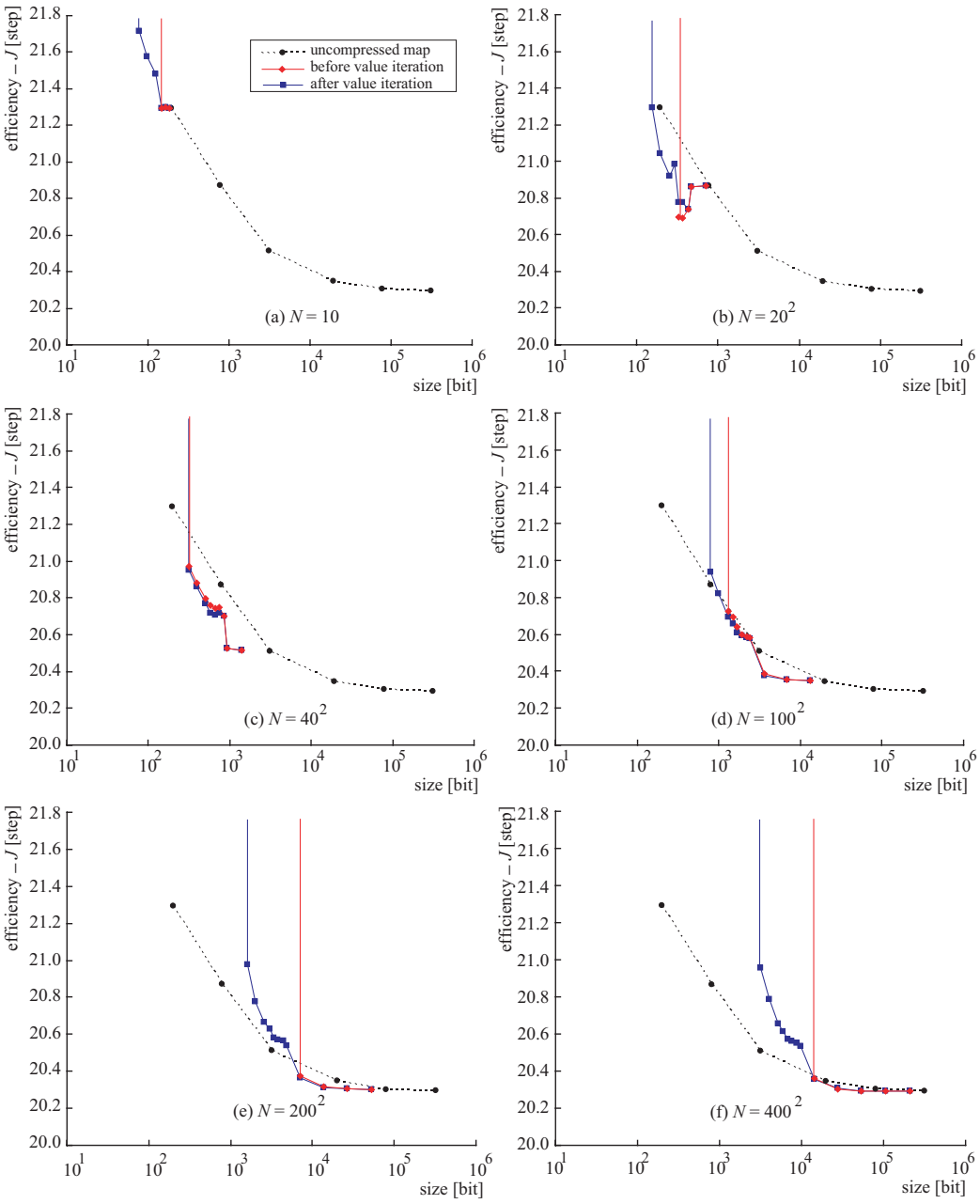


Fig. 4.6: Evaluation of the Value Iteration Algorithm

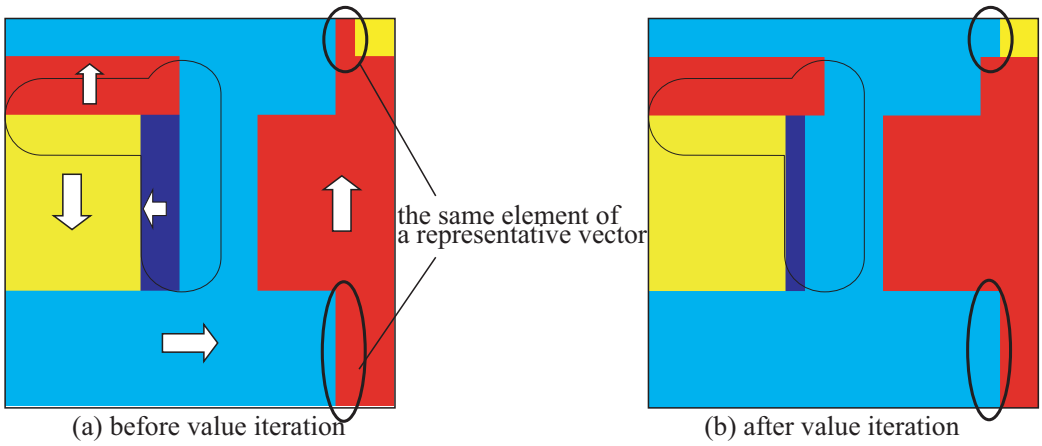


Fig. 4.7: VQ Maps Before/After Value Iteration ($N = 20^2, N_c = 3$)

4.2 Choice of Blocking

We daringly try finding some good blocking ways that make possible to compress state-action maps both with high compression ratio and efficiency. As mentioned before, the VQ method starts from the blocking process. None the less, it is very difficult to find a blocking way that enables to crate small but efficient VQ maps. In fact, we cannot know the efficiency of a VQ map until it is created and used practically. However, we can remove some unsuitable ways of blocking in advance. Moreover, it may be possible to find some ways of blocking with the potentiality of high efficiency and high compression ratio.

4.2.1 Limitation of Compression Ratio

First of all, we should note that limitation of compression ratio exists. We can calculate it from Eq. (2.2) and Eq. (3.18) as:

$$\begin{aligned} L_{\mathrm{VQ}}/L_{\mathrm{uncomp}} &= \frac{N/N_\varepsilon\lceil\log_2 N_c\rceil + N_\varepsilon N_c\lceil\log_2 M\rceil}{N\lceil\log_2 M\rceil} \\ &= \frac{\lceil\log_2 N_c\rceil}{N_\varepsilon\lceil\log_2 M\rceil} + \frac{N_\varepsilon N_c}{N}\ [\mathrm{bit}], \end{aligned} \tag{4.1}$$

where L_{uncomp} denotes the size of a state-action map before compression. The first and second terms are related to the ratios of quantization table and codebook respectively. This equation suggests that N_ε, the number of elements in a vector, has an influence on the compression ratio as well as the number of representative vectors N_c.

For instance, Fig. 4.8 shows the compression ratio according to each set of N_ε and N_c when the state-action map in Fig. 2.10 (c) is compressed. As shown in this figure, a suitable range of N_ε exist for each number of representative vectors. When N_ε is smaller than the range, the compression ratio is restricted. That is because the size of the quantization table becomes enlarged. On the other hands, a large N_ε makes the codebook enlarge.

4.2.2 Estimation of Better Blocking Way

We should also consider how to cut the map. In this process, we should consider the bias of the distribution of actions on a map. We show an example in Fig. 4.9. There are 6×6 tiles that are distinguished by their colors. We regard them as data that is compressed. The array of tiles is

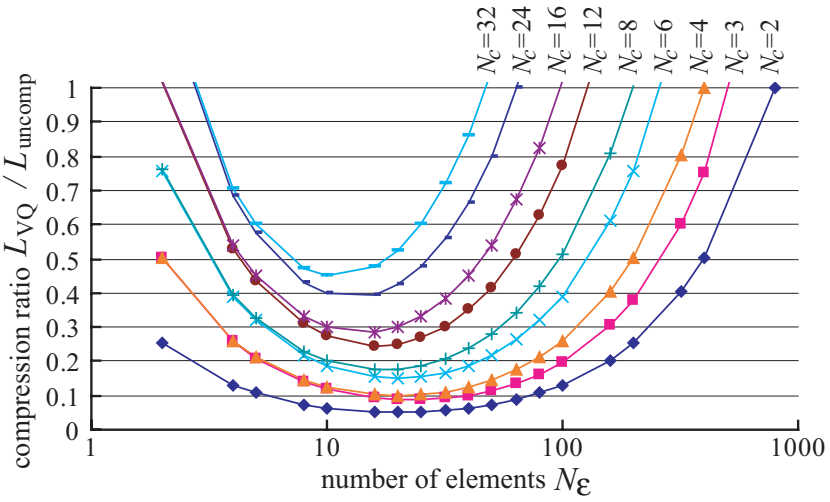


Fig. 4.8: Example of Relation between Number of Elements, Number of Clusters and Compression Ratio (a state-action map for the puddle world task with $N = 40^2$)

horizontally cut off in Fig. (a), while it is divided vertically. We can think that the case of Fig. (a) is suitable for clustering than the case of Fig. (b) because the vectors in Fig. (a) are all alike.

It is useful if the goodness of blocking, which is equivalent to the definition of the pair of ν and ε is quantified. Though it is possible when the state-value distortion is used. However, it takes considerable time for evaluating some pairs of ν and ε toward a large state-action map.

Evaluation with Information Entropy

Instead of the state-value distortion, here we try to apply the concept of information entropy to evaluation of each pair of ν and ε. This concept is related to lossless compression methods rather than lossy methods.

When a set of actions exists, its entropy is calculated as follows:

$$H(\text{the set}) = -\sum_{i=0}^{M-1} P(a_i) \log_2 P(a_i), \tag{4.2}$$

where $P(a_i)$ denotes the occurrence rate of a_i in the sequence. When $P(a_i) = 0$, $P(a_i) \log_2 P(a_i)$ is regarded as zero. If the actions appear evenly, this value becomes $H = \log_2 M$, which is the maximum value of the entropy.

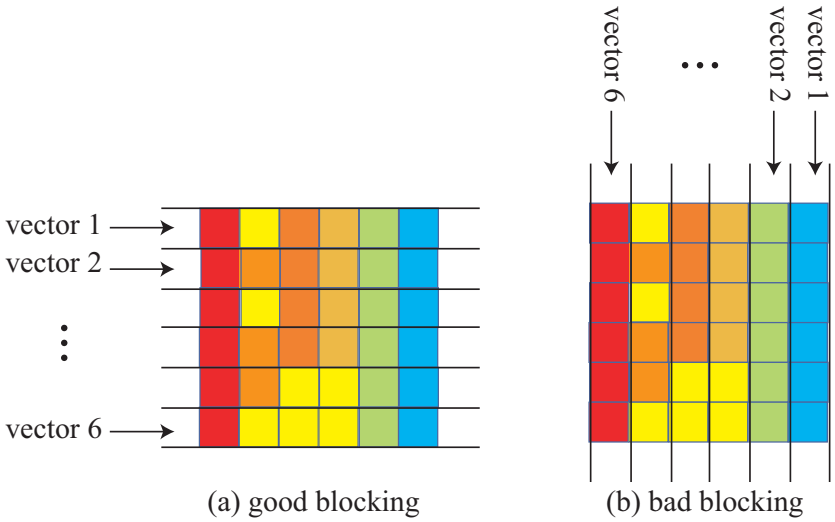


Fig. 4.9: Examples of Good/Bad Blocking

On the other hand, the value becomes zero if the set contains only one kind of action.

In the VQ method, states that are related to an element of a representative vector should not contain various actions as explained with Fig. 4.9. The entropy function can be used for quantifying the variety of actions of the states. The smaller the entropy is, the smaller the distortion will be expected when the set of actions (action indexes) is replaced by an representative action.

We define the above idea. When a pair of ν and ε is given, we consider the sets of actions $\mathcal{E}_{i_\varepsilon}$ $(i_\varepsilon = 0, 1, 2, \ldots, N_\varepsilon - 1)$. $\mathcal{E}_{i_\varepsilon}$ contains all actions of the states that are related to i_εth element of representative vectors. The smaller the entropy $H(\mathcal{E}_{i_\varepsilon})$ is, the more the actions in $\mathcal{E}_{i_\varepsilon}$ are biased. The average of the entropy:

$$\overline{H}(\nu, \varepsilon) = \frac{1}{N_\varepsilon} \sum_{i_\varepsilon=0}^{N_\varepsilon - 1} H(\mathcal{E}_{i_\varepsilon}) \tag{4.3}$$

would be an evaluation measure for a way of blocking. Since this equation evaluate each set separately, cross-interaction of each set is not considered.

By Eq. (4.3), we evaluate the way of blocking for the puddle world task. The pair of (ν, ε) has been defined in Eq. (3.30) with two parameters: w_x

and w_y, which are the numbers of intervals of each block along each axis respectively. Though we have used $(w_x, w_y) = (\sqrt{N}, 1)$ (horizontal blocking) as a simple way of blocking in Chapter 3, $(w_x, w_y) = (1, \sqrt{N})$ (vertical blocking) is also a simple way. As a matter of fact, the horizontal blocking has been chosen based on the evaluation of entropy shown in Table 4.1.

We evaluate the efficiency of VQ maps that are built with vertical blocking. The results are shown in Fig. 4.10. Though some of the VQ maps are superior to the VQ maps with horizontal blocking, compression with vertical blocking is unstable. Some VQ maps with large N_c are broken as shown in this figure. Though we cannot estimate efficiency of final products completely from the entropy function, it can be some amount of information.

Table 4.1: Entropy $\overline{H}$ (small: good)

N	horizontal blocking $(w_x, w_y) = (\sqrt{N}, 1)$	vertical blocking $(w_x, w_y) = (1, \sqrt{N})$
10^2	0.98	1.19
20^2	1.03	1.26
40^2	1.03	1.24
100^2	1.04	1.24
200^2	1.05	1.24
400^2	1.06	1.24

Evaluation of Blocking on Different Number of Elements

When pairs of (ν, ε) that have different N_ε are compared, the pair with smaller $N_\nu (= N/N_\varepsilon)$ has advantage to make $\overline{H}$ small. N_ν is the number of elements in each set $\mathcal{E}_{i_\varepsilon}$. That is because the bias of elements tends to become large due to statistical insufficiency when N_ν is small. This kind of bias never contributes to make compression easy.

Therefore, $\overline{H}$ should be normalized if we want to use entropy to compare some pairs of (ν, ε) with different N_ε. Though there are some methods for normalizing, we decide to divide $\overline{H}$ by $\log_2 N_\nu$ and define the entropy

$$\mathcal{H}(\nu, \varepsilon) = \frac{1}{\log_2 N_\nu} \overline{H}(\nu, \varepsilon) = \frac{1}{N_\varepsilon \log_2 N_\nu} \sum_{i_\varepsilon=0}^{N_\varepsilon - 1} H(\mathcal{E}_{i_\varepsilon}). \tag{4.4}$$

$\log_2 N_\nu$ is the maximum value of $\overline{H}$ when $M \geq N_\nu$. Therefore, it is suitable for normalizing $\overline{H}$ when the statistical insufficiency is considered.

We measure the relation between the entropy and the compression ratio when a target efficiency J is given. The state-action map with $N = 40^2$ for the puddle world task is used for this evaluation. Figure 4.11 illustrates the results. The desired value of $-J$ is fixed at 20.52[step] and 21.00[step] in Fig. 4.11 (a) and (b) respectively. Incidentally, $-J$ of the uncompressed map is 20.51[step]. The VQ maps are build with every pair of (w_x, w_y), and the smallest one that fulfill the desired value of J is chosen. In the figures, the results are plotted by different marks according to the dimensions of vectors.

From these figures, we cannot find clear relation between the entropy and compression ratio, especially in Fig. (a). As mentioned before, the efficiency of a VQ map is unknown until it is used. Moreover, suitable ways of blocking are not always identical toward different levels of compression ratio. Therefore, the suitable way of blocking toward a required efficiency and a compression ratio cannot be answered by only one value of entropy.

However, it seems that pairs of (ν, ε) that have large entropy can be eliminated from the candidate of suitable ways of blocking. That is because the limitation of compression ratio clearly relates to the entropy.

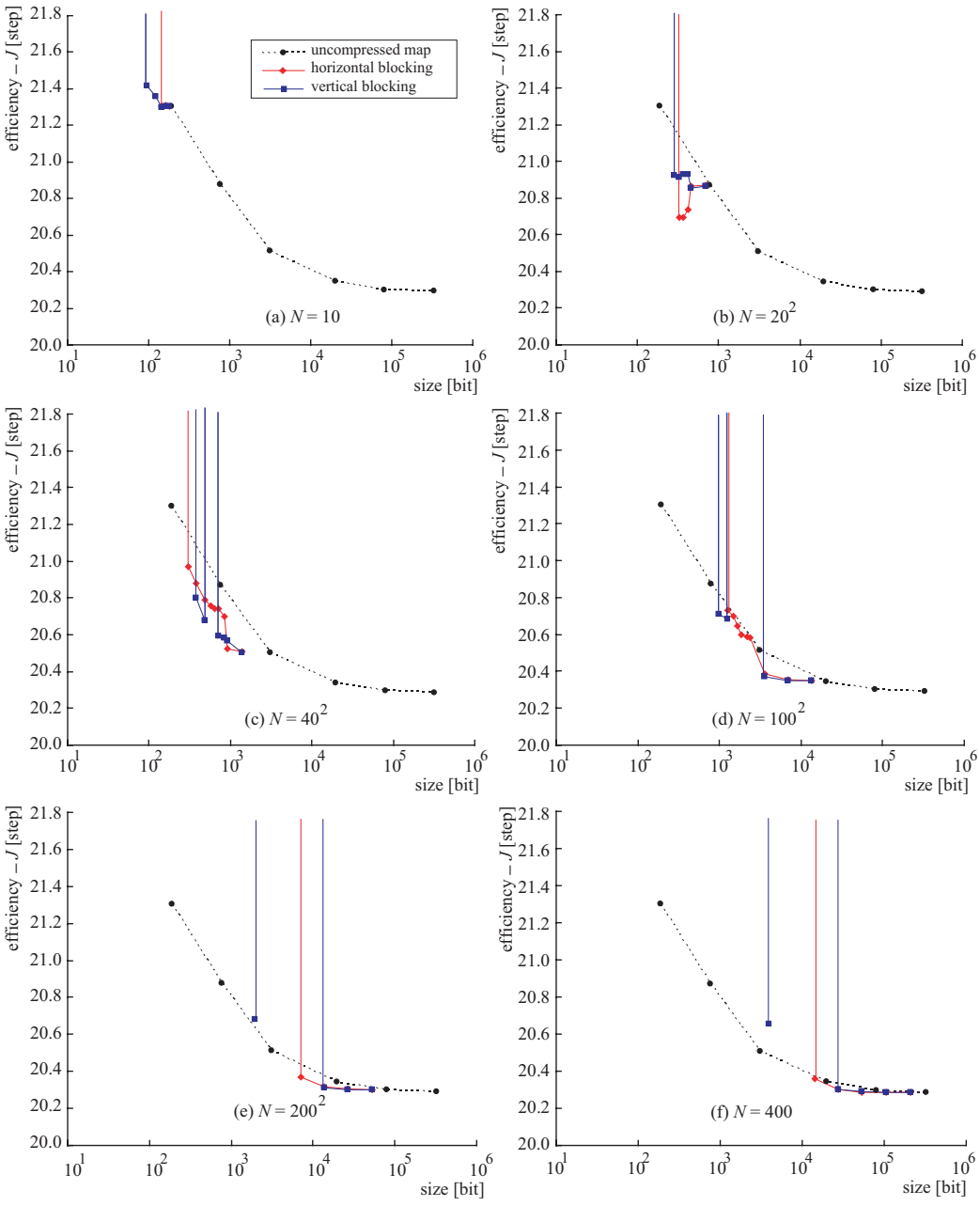


Fig. 4.10: Comparison of Two Ways of Blocking

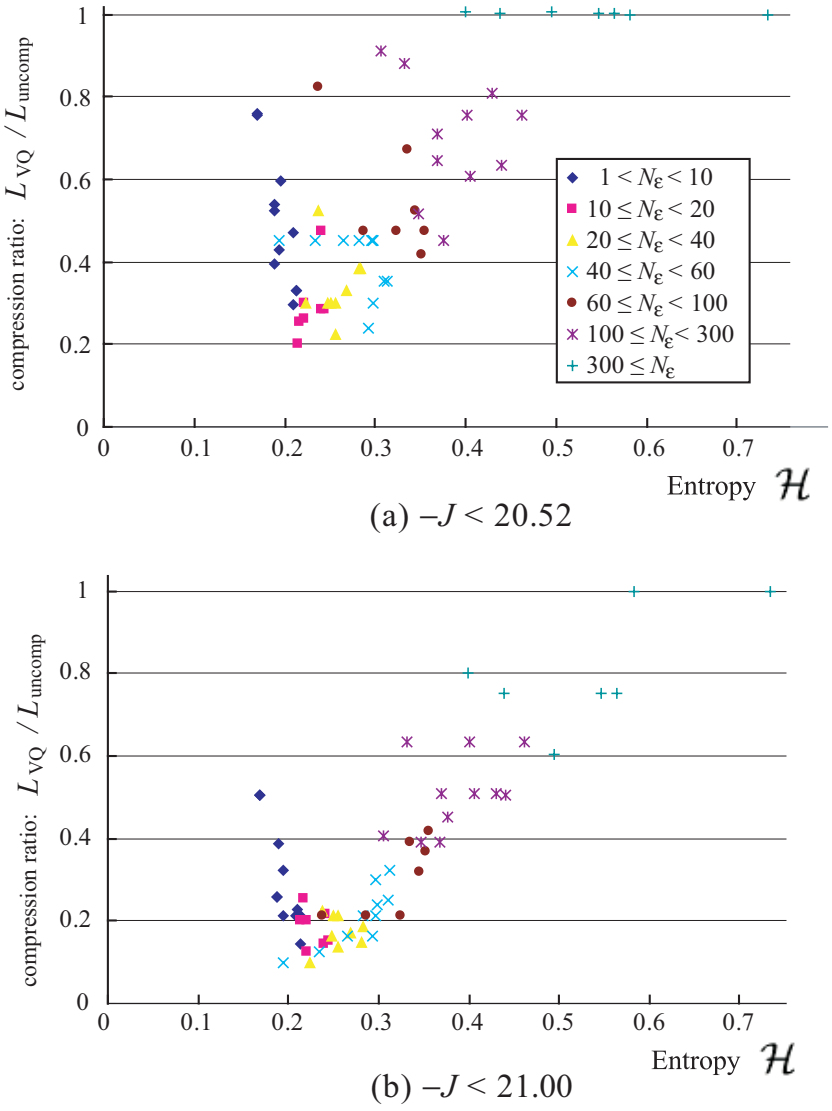


Fig. 4.11: Relation between Entropy and Compression Ratio

4.3 Multi Layered Vector Quantization

4.3.1 Double Layered VQ Map

Since redundancy remains in VQ maps in many cases, we attempt to reduce the size of VQ maps with further compression. Here we utilize VQ once more for compression of VQ maps. Since lossy compression of a VQ map is difficult, we should use VQ as a lossless compression.

In the process of vector quantization of a state-action map, a state-action map π_{MAP} is decomposed into four transforms:

- ν: transform from state index to vector index,
- ε: transform from state index to element index,
- ω: transform from vector index to representative vector index (= quantization table), and
- c: transform from a pair of representative vector index and an element index (= codebook).

Their relation has been defined as follows:

$$\pi_{\mathrm{MAP}}(i) = c(\omega(\nu(i)), \varepsilon(i)) = c(i_{\nu\omega}, i_\varepsilon). \tag{4.5}$$

Since ω and c are tables of indexes, they can be compressed by VQ. We define $\nu_\omega, \varepsilon_\omega, \omega_\omega$, and c_ω for VQ of ω. As with Eq. (4.5), their relation is defined as follows:

$$i_{\nu\omega} = \omega(i_\nu) = c_\omega\left(\omega_\omega(\nu_\omega(i_\nu)), \varepsilon_\omega(i_\nu)\right). \tag{4.6}$$

When codebook c is decomposed into $\nu_c, \varepsilon_c, \omega_c$, and $\mathcal{C}_2$, their relation is represented by

$$c(i_{\nu\omega}, i_\varepsilon) = c_c(\omega_c(\nu_c(i_{\nu\omega}, i_\varepsilon)), \varepsilon_c(i_{\nu\omega}, i_\varepsilon)) \tag{4.7}$$

By the above transforms, a VQ map is decomposed into the set of $\nu, \varepsilon, \nu_\omega, \omega_\omega, \varepsilon_\omega, c_\omega, \nu_c, \omega_c, \varepsilon_c$, and c_c. Action indexes are recorded only in c_c. The action index for s_i can be obtained by

$$\begin{aligned}
\pi_{\mathrm{MAP}}(i) &= c_c\Big(\omega_c(\nu_c(i_{\nu\omega}, i_\varepsilon)), \varepsilon_c(i_{\nu\omega}, i_\varepsilon)\Big) \\
&= c_c\Big(\omega_c(\nu_c(c_\omega\left(\omega_\omega(\nu_\omega(i_\nu)), \varepsilon_\omega(i_\nu)\right), i_\varepsilon)), \varepsilon_c(c_\omega\left(\omega_\omega(\nu_\omega(i_\nu)), \varepsilon_\omega(i_\nu)\right), i_\varepsilon)\Big) \\
&= c_c\Big(\omega_c(\nu_c(c_\omega\left(\omega_\omega(\nu_\omega(\nu(i))), \varepsilon_\omega(\nu(i))\right), \varepsilon(i))), \\
&\qquad \varepsilon_c(c_\omega\left(\omega_\omega(\nu_\omega(\nu(i))), \varepsilon_\omega(\nu(i))\right), \varepsilon(i))\Big)
\end{aligned} \tag{4.8}$$

Though this equation is too complicated to follow, we can understand from it that a state index can be turned into an action index by the 10 transforms. The structure of a policy represented by the 10 transforms are illustrated in Fig.4.12. We name this structure a double layered VQ map (a DLVQ map).

Figure 4.13 shows a concrete example of compression. In Fig. 4.13(b), a codebook is composed of three representative vectors that contain 20 elements. When this codebook is regarded as a 3×20 table and cut into 20 vectors that contain three elements as shown in (b), we notice that only seven kinds of vector exist in the 20 vectors. Therefore, this codebook can be replaced a pair of quantization table and codebook shown in Fig. 4.13(c).

The size of a DLVQ map can be calculated based on the amount of memory for recording the four tables: $\omega_\omega, \omega_c, c_\omega$, and c_c. The size of a vector quantized quantization table is

$$N_{\omega v}\lceil \log_2 N_{\omega c}\rceil + N_{\omega c}\frac{N_v}{N_{\omega v}}\lceil \log_2 N_v\rceil [\text{bit}] \tag{4.9}$$

when the quantization table ω is divided into $N_{\omega v}$ vectors and is classified into $N_{\omega c}$ representative vectors. The size of a vector quantized codebook is then

$$N_{cv}\lceil \log_2 N_{cc}\rceil + N_{cc}\frac{N_c N_\varepsilon}{N_{cv}}\lceil \log_2 m\rceil [\text{bit}] \tag{4.10}$$

when the codebook c is divided into N_{cv} vectors and is classified into N_{cc} representative vectors. If we do not consider the memory consumption for other transforms: $\nu, \varepsilon, \nu_\omega, \varepsilon_\omega, \nu_c$ and ε_c, the amount of memory for a DLVQ map is

$$\begin{aligned} L = {} & N_{\omega v}\lceil \log_2 N_{\omega c}\rceil + N_{\omega c}\frac{N_v}{N_{\omega v}}\lceil \log_2 N_v\rceil + \\ & N_{cv}\lceil \log_2 N_{cc}\rceil + N_{cc}\frac{N_c N_\varepsilon}{N_{cv}}\lceil \log_2 m\rceil . \end{aligned} \tag{4.11}$$

In the case of Fig. 4.13, the size of a codebook in (b) is 120[bit]. On the other hand, that of the VQ data in (c) is $60 + 42 = 102$[bit]. Though the number of vectors is reduced from 20 to 7, the compression ratio is not large because the size of the quantization table runs up. If we consider the number of bits for representing the transforms, this compression may not be effective. In practical uses, the effectiveness of multi layered VQ will be obtained when a state-action map is large.

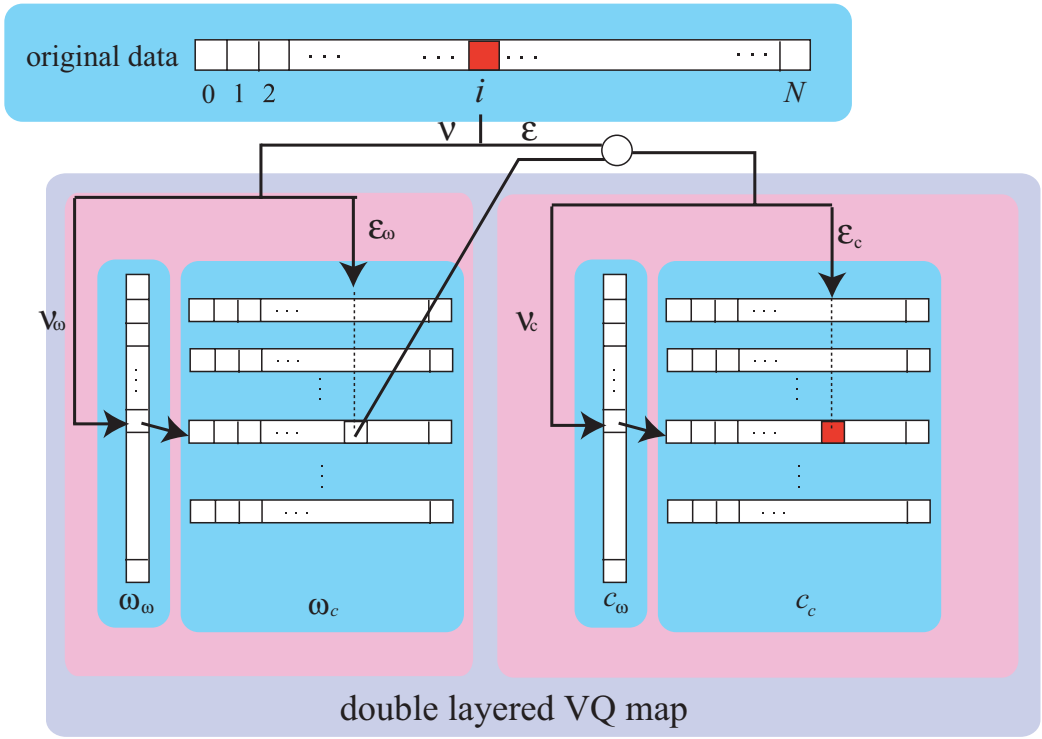


Fig. 4.12: Structure of double layered VQ

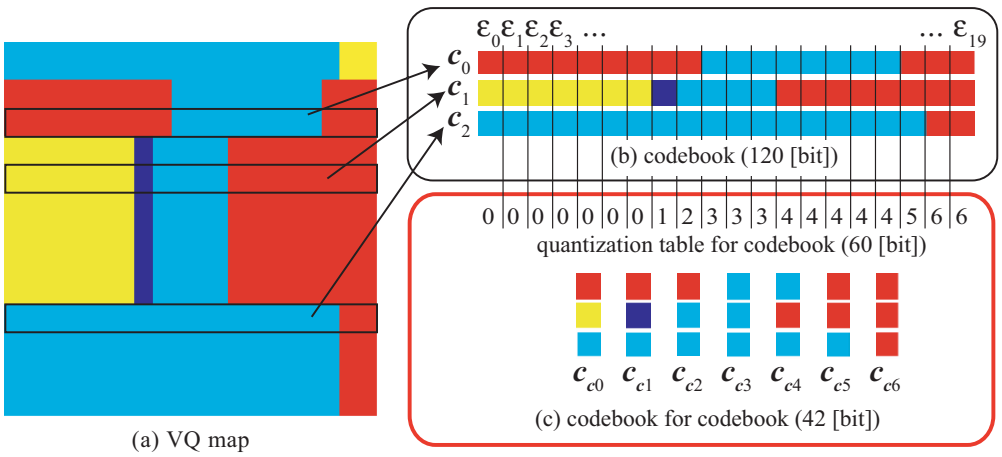


Fig. 4.13: Compression of Codebook

4.3.2 Multi Layered VQ Map

In theory, we can apply VQ to all tables in a DLVQ map. Therefore, triple layered VQ maps exist. The triple layered VQ map also can be applied VQ. In this manner, we can create n-layered VQ maps. We call the maps with $n \geq 2$ multi layered VQ maps. As shown in Fig. 4.14, any table in a multi layered VQ map can be decomposed into four transforms. Since two of them become a quantization table and a codebook again, they can be decomposed into four transforms respectively. In a multi layered VQ map, action indexes are recorded only in a table $c_{cc\ldots c}$. The other transforms are used for operation of indexes.

In fact, an $n+1$ layered VQ map is not always smaller than an n layered VQ map. When a table is small, a high compression ratio cannot be expected. Moreover, we cannot ignore the memory consumption of the executable code for reading a map if it contains many transforms that belong to the group of ν or ε. Therefore, the concept of multi layered VQ maps is not useful in the viewpoint of compression. However, the format of multi layered VQ maps is interesting because it is an extreme opposite of the format of state-action maps.

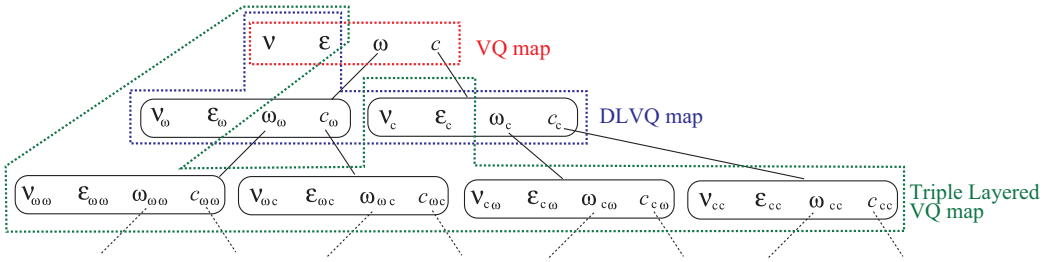


Fig. 4.14: Multi layered VQ

4.3.3 Evaluation

The VQ maps created in Sec. 3.5 are compressed to DLVQ maps here. In the case of a quantization table: $\omega = \{\omega(0), \omega(1), \ldots, \omega(N_v - 1)\}$, we create vectors based on

$$\begin{pmatrix} \nu_\omega(i) \\ \varepsilon_\omega(i) \end{pmatrix} = \begin{pmatrix} \lfloor i/N_{\omega v} \rfloor \\ \lfloor i\%N_{\omega v} \rfloor \end{pmatrix} \quad (i \in \mathcal{I}_\mathcal{V}). \tag{4.12}$$

It means that the sequence of representative indexes in a quantization table is divided uniformly, and each partial sequence is regarded as a vector. We try all possible $N_{\omega v}$ and choose the compressed quantization table (ω_ω and ω_c) that has the smallest size. This manner is impossible if N_v is a prime number. Moreover, the smallest compressed quantization table is sometimes larger than that of the original quantization table. In those cases, we use the original map ω as a part of a DLVQ map.

The blocking of a codebook c is then fixed as

$$\begin{pmatrix} \nu_c(i_{v\omega}, i_\varepsilon) \\ \varepsilon_c(i_{v\omega}, i_\varepsilon) \end{pmatrix} = \begin{pmatrix} i_\varepsilon \\ i_{v\omega} \end{pmatrix}. \tag{4.13}$$

This equation means that action indexes in a representative vector are assigned to different vectors so that the redundancy that exists in every representative vector is reduced. When the size of the compressed codebook is larger than that of the original codebook, we choose the original one as a part of a DLVQ map.

In Fig. 4.15, the size and the efficiency of each DLVQ map is compared to each original VQ map, which is improved by the value iteration algorithms in Sec. 4.1. Since a VQ map and its DLVQ map give the identical policy due to lossless compression, the graph of DLVQ maps shifts to the left from the graph of VQ maps. As shown in this figure, the larger N is, the more the compression ratio is improved. It suggests that representative vectors become redundant when their length is long. the compression ratio of DLVQ maps is larger when N is large. The numerical data is shown in Table 4.2. In this table, pairs of an uncompressed map and a VQ map that is more efficient than the uncompressed map are compared. When $N \geq 400^2$, the size of the DLVQ map becomes less than half that of the VQ map.

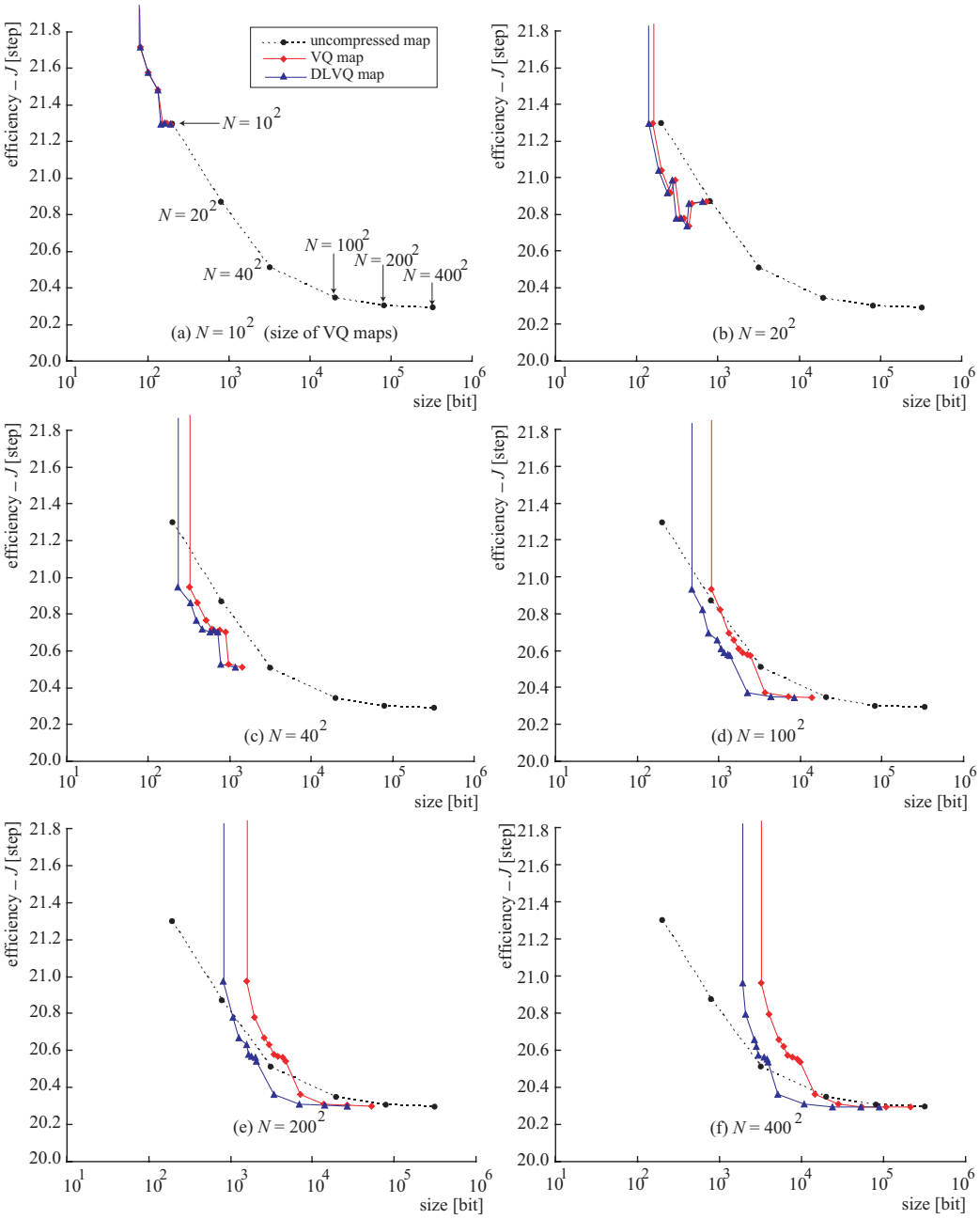


Fig. 4.15: Evaluation of DLVQ maps

Table 4.2: Practical Compression Ratio of DLVQ maps

uncompressed maps			more effective compressed maps					comp. ratio		
N	L_{uncomp} [bit]	$-J$ [step]	N	N_c	L_{VQ} [bit]	L_{DLVQ} [bit]	$-J$ [step]	$\frac{L_{\text{DLVQ}}}{L_{\text{VQ}}}$	$\frac{L_{\text{VQ}}}{L_{\text{uncomp}}}$	$\frac{L_{\text{DLVQ}}}{L_{\text{uncomp}}}$
10^2	200	21.29	20^2	3	160	142	21.29	0.89	0.80	0.71
20^2	800	20.87	40^2	4	400	326	20.86	0.82	0.50	0.41
40^2	3200	20.51	100^2	16	3600	2196	20.38	0.61	1.13	0.67
100^2	20k	20.35	200^2	32	13.8k	7072	20.31	0.51	0.69	0.35
200^2	80k	20.30	400^2	64	53.6k	23.6k	20.29	0.44	0.67	0.30

Figure 4.16 shows a graph for comparison of DLVQ maps and the binary-tree policies obtained in Sec. 3.6.2. In the comparison with VQ maps in Fig. 3.18, smaller and more efficient binary-tree policies exist in a part of the graph. When the redundancy of the VQ maps are cut, however, the part is covered by some DLVQ maps.

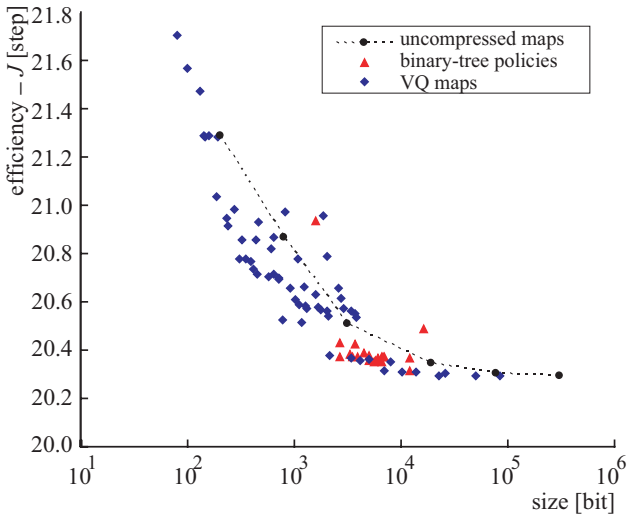


Fig. 4.16: Comparison of DLVQ maps and Binary-Tree Policies

4.3.4 Partitioning

The VQ method can apply not only to a whole state-action map but also to a part of the map. Therefore, we can imagine a compressed state-action map that is composed of some VQ maps. This compressed map has another

kind of tree structure.

To create such a map, we divide $\mathcal{S}$ into N_p subsets: $\mathcal{S}_j$ $(j = 0, 1, 2, \ldots, N_p - 1)$ at first. In this case, j is regarded as a subset indexes. We call this procedure *partitioning*. The set of subset indexes is represented by $\mathcal{I}_{\mathcal{S}_{\text{sub}}}$. In each set $\mathcal{S}_j$, states are ordered as $\mathcal{S}_j = \{s_k | k = 0, 1, 2, \ldots, N_{\mathcal{S}_i} - 1\}$, where $k \in \mathcal{I}_{\mathcal{S}_j}$ denotes the local state index in $\mathcal{S}_j$. $N_{\mathcal{S}_j}$ is the number of states in $\mathcal{S}_j$. The state index set $\mathcal{I}_{\mathcal{S}}$, the subset index set $\mathcal{I}_{\mathcal{S}_{\text{sub}}}$, and the local state index sets $\mathcal{I}_{\mathcal{S}_j}$ are related to the following transforms: $\kappa : \mathcal{I}_{\mathcal{S}} \to \mathcal{I}_{\mathcal{S}_{\text{sub}}}$, and $\iota : \mathcal{I}_{\mathcal{S}} \to \mathcal{I}_{\mathcal{S}_j}$.

A state-action map is also divided into sub maps: $\pi_{\text{MAP}i}$ $(i = 0, 1, 2, \ldots, N_p - 1)$. Each sub map can be compressed to a sub VQ map $\pi_{\text{VQ}i}$ $(i = 0, 1, 2, \ldots, N_p - 1)$ by the VQ method. This compressed map composed of the sets of sub VQ maps is named *a partitioned VQ map*. Figure 4.17 shows the partitioning of a state-action map. As shown in this figure, each sub maps can have different numbers of elements. The size of a partitioned VQ map is calculated from Eq. (3.18) as

$$L_{\text{P-VQ}} = \sum_{j=0}^{N_p-1} N_{\nu j} \lceil \log_2 N_{cj} \rceil + N_{\varepsilon j} N_{cj} \lceil \log_2 M \rceil, \tag{4.14}$$

where $N_{\nu j}$, N_{cj}, and $N_{\varepsilon j}$ denote the numbers of vectors, representative vectors, and elements of a vector respectively in $\mathcal{S}_j$.

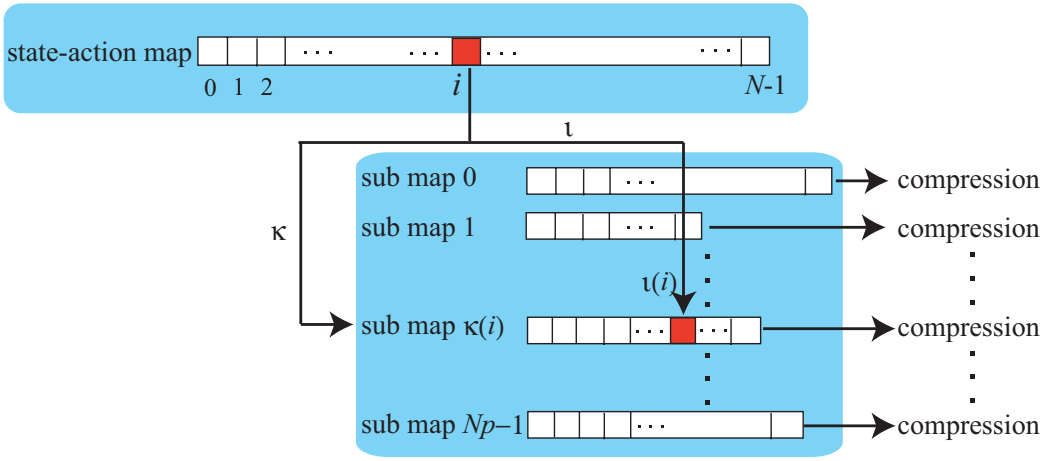


Fig. 4.17: Partitioning

By the introduction of partitioning, the flexibility of compression is enhanced. In some situations, some sub maps do not have to be compressed by VQ. If a partition has only one kind of action, for instance, VQ is not required. In that sense, partitioning is a bridge between the VQ method and other methods.

As another merit, some kinds of clustering algorithm can reduce the computing cost by partitioning. In the case of the PNN algorithm, the order of time complexity is reduced from $\mathrm{O}(MN^3/N_\varepsilon^2)$ to $\mathrm{O}(N_p M(N/N_p)^3/N_\varepsilon^2) = \mathrm{O}(MN^3/(N_\varepsilon N_p)^2)$ if N_ε is not changed.

4.4 Discussion

In this section, the value iteration algorithms for VQ maps, discussion about how to find good ways of blocking, and multi layered vector quantization are introduced.

Summary of Effectiveness on The Puddle World Task

In Table 4.3, the effectiveness of the value iteration algorithms for VQ maps and double layered vector quantization is summarized. In a row,

- data of an uncompressed state-action map,
- data of a VQ map without the techniques in this chapter,
- the improved efficiency of the VQ map by the value iteration algorithms,
- reduced size of the VQ map by the double layered VQ, and
- final compression ratio

are illustrated from left to right. Differently from Table 4.2, we have tried to adjust the efficiency of a VQ map to that of an uncompressed map with adjustment of N_c.

Table 4.3: Size Reduction and Efficiency Enhancement with The Techniques in This Chapter

uncomp. maps			compressed maps						
N	L_{uncomp} [bit]	$-J$ [step]	N	N_c	L_{VQ} [bit]	$-J$ [step]	$-J$ after VI [step]	L_{DLVQ} [bit]	comp. ratio
10^2	200	21.29	20^2	3	160	broken	21.29	142	0.71
20^2	800	20.87	40^2	4	400	20.88	20.86	326	0.41
40^2	3200	20.51	100^2	11	2600	20.42	20.41	1274	0.40
100^2	20k	20.34	200^2	21	9400	20.35	20.34	4180	0.21
200^2	80k	20.30	400^2	36	31.2k	20.30	20.30	12.9k	0.16

(VI: value iteration)

As shown in this table, a smaller and more efficient VQ map exists toward each uncompressed map from $N = 10^2$ to $N = 200^2$. When the techniques are not applied to VQ maps, the compression ratio is at most 0.65 as shown in Table 3.1. However, the compression ratio is certainly

enhanced by the value iteration and double layered VQ.

Especially in the case of where N is large, the enhancement of compression ratio is significant. Though this tendency is found only in the puddle world task at this moment, high compression ratios can be expected at large scale decision making problems.

Remaining Issues

We have interest in the forms of multi layered VQ maps explained in Sec. 4.3.2. This structure is related to tree structures. DLVQ maps can be regarded as a kind of tree structured policies, while they are compressed by VQ. Even a VQ map can be regarded as a tree structured policy whose node has many branches. The study on multi layered VQ maps will clarify the relation between the structure of VQ maps and tree structures.

Chapter 5

Application and Evaluation I: The Acrobot

We take up control of the Acrobot as the application of our method. The Acrobot is an underactuated robot and provides challenging control problems due to its nonlinearity. In this chapter, we try creating state-action maps and compressing them. A swinging up task of the Acrobot is solved by DP without heuristics. The result of DP is compressed to an arbitrary size of control policy by our VQ method.

The structure of this chapter is as follows. In Sec. 5.1, dynamics of the Acrobot and the background of its research are explained. The task that is handled in this chapter is defined in Sec. 5.2. State-action maps for the task are obtained in Sec. 5.3. The maps are compressed and evaluated in Sec. 5.4. This chapter is concluded in Sec. 5.5.

5.1 The Acrobot

The Acrobot, which is shown in Fig. 5.1, is a planar robot that is composed of two links and two joints. Link 1 can swing freely at this joint. Torque τ can be given at Joint 2. This robot can be parameterized as shown in Table 5.1. Its pose is then represented by the pair of angles (θ_1, θ_2) in the figure. Their relation are often compared to a gymnast who hangs from a bar, or to an arm of human beings. Control problems of the Acrobot have frequently studied in control engineering [Spong, 1994; Spong, 1995; Boone, 1997; Xin, 2002; Banavar, 2003; Xin, 2004] and learning [Sutton, 1996; Sutton, 1998; Yoshimoto, 2005] as a subject of underactuated robots. This term denotes a robot that has more degrees of freedom than the number of actuators.

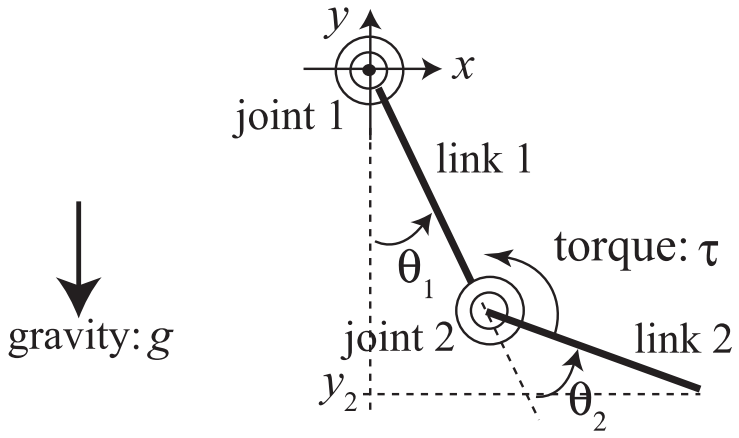


Fig. 5.1: The Acrobot

Table 5.1: Parameters of The Acrobot

description ($i = 1, 2$)	parameter
length of link i	ℓ_i
length from joint i to mass center of link i	ℓ_{ci}
mass of link i	m_i
moment of inertia of link i	I_i

The dynamics of this robot can be represented by

$$\ddot{\theta}_2 = \frac{(\tau + \phi_1 d_2/d_1 - m_2 \ell_1 \ell_{c2} \dot{\theta_1}^2 \sin\theta_2 - \phi_2)}{(m_2 \ell_{c2}^2 + I_2 - d_2^2/d_1)}, \text{ and} \tag{5.1}$$

$$\ddot{\theta}_1 = \frac{-(d_2 \ddot{\theta}_2 + \phi_1)}{d_1}, \tag{5.2}$$

where

$$d_1 = m_1 \ell_{c1}^2 + m_2(\ell_1^2 + \ell_{c2}^2 + 2\ell_1 \ell_{c2} \cos\theta_2) + I_1 + I_2,$$
$$d_2 = m_2(\ell_{c2}^2 + \ell_1 \ell_{c2} \cos\theta_2) + I_2,$$
$$\phi_1 = m_2 \ell_1 \ell_{c2}(-\dot{\theta_2}^2 - 2\dot{\theta}_2 \dot{\theta}_1) \sin\theta_2 + (m_1 \ell_{c1} + m_2 \ell_1) g \cos(\theta_1 - \pi/2) + \phi_2, \text{ and}$$
$$\phi_2 = m_2 \ell_{c2} g \cos(\theta_1 + \theta_2 - \pi/2).$$

Global optimization of its control is one of the most challenging problems due to nonlinearity of the Acrobot. In other words, to obtain an optimal state-action map in continuous state space is one of the most ultimate subjects around this robot. Though feedback linearization is a standard method [Spong, 1994; Xin, 2004], it does not aim to global optimization.

DP is a possible way to solve a state-action map though it is obtained in discrete space. However, use of DP is more compliant to the concept of optimal control than any linearization technique. In the case of DP, the state-action map is determined by an evaluation function, some boundary conditions, and discretization if heuristics are not used in the code. There is no synthetic rule of control in the state-action map.

No use of heuristics was however difficult due to the problem of computing complexity. For example, Boone has applied DP to minimum time control of the Acrobot [Boone, 1997] in 1997. In this study, increase of energy of the robot is used as a heuristic. However, it seems that DP can be used without heuristics if we consider the progress of computers in this ten years. It is very significant if we can create and observe a feasible global policy without heuristics.

We try creating global policies of a task of the Acrobot without heuristics and compressing it by using the VQ method. Differently from the puddle world task, control of the Acrobot is a problem of dynamic and nonlinear control. The evaluation of the VQ method is done in this section.

5.2 Height Task and Its Formulation

We take up the height task [Boone, 1997]. In the height task, the Acrobot is controlled so that the free end of Link 2 gets over a prescribed height. When the height of the free end of Link 2 is represented by y_2 in the xy coordinate system defined in Fig. 5.1, the relation between y_2 and the pair of (θ_1, θ_2) is

$$y_2 = -\ell_1 \cos\theta_1 - \ell_2 \cos(\theta_1 + \theta_2). \tag{5.3}$$

When the prescribed height is denoted by h on the y-axis, $y_2 \geq h$ is the purpose of control.

In this thesis, the task is regarded as a problem of minimum time control. In this case, the evaluation functional is defined as

$$J[\tau] = \int_0^T -1 dt = -T, \tag{5.4}$$

where T is the time when the purpose of control: $y_2 \geq h$ is completed. μ is regarded as a function that gives the torque at time t $(0 \leq t < T)$ in the above equation. We set the time interval for changing torque to 0.2[s].

We should pay attention to the robustness of control. When we can solve a global solution of an optimal control problem, however, this argument is less important than the cases of linearization. That is because a global policy can give an appropriate torque based on the state of the Acrobot after the behavior of the system is upset by some troubles. However, the policy will lack robustness if the parameters in the state equation of the system are inaccurate.

5.3 Obtaining State-Action Map

We set values of the parameters as shown in Table 5.2(a). The set of these values is also used in [Sutton, 1996; Boone, 1997]. In this section, global optimal control for the height task is solved and compressed under this condition. In this paper, the torque τ is limited to -1, 0, or 1[Nm]. We define set of $\mathcal{A}$ as the set of these torques.

Table 5.2: Setting of Parameters and Variables

(a)

parameter	value	
ℓ_1, ℓ_2	1.0	[m]
ℓ_{c1}, ℓ_{c1}	0.50	[m]
m_1, m_2	1.0	[kg]
I_1, I_2	1.0	[kg m^2]
g	9.8	[m/s^2]

(b)

variable	domain
θ_1	$(-\infty, \infty)$
θ_2	$(-\infty, \infty)$
$\dot{\theta}_1$	$[-720, 720]$[deg/s]
$\dot{\theta}_2$	$[-1620, 1620]$[deg/s]
τ	$-1, 0$, or 1[Nm]

For implementation of value iteration, we limit the domains of $\dot{\theta}_1$ and $\dot{\theta}_2$ as shown in Table 5.2(b). These values are set based on [Sutton, 1998]. In this case, the height task should be completed without excess of the range. We show the domain of each parameter in Table 5.2(b).

5.3.1 Value Iteration

Discretization

First of all, we construct the set of states $\mathcal{S} = \{s_i | i = 0, 1, 2, \ldots, N-1\}$. We limit the intervals of θ_1 and θ_2 to $[0, 360)$ in calculation and consider the space $\mathcal{X}$ that is direct product of the intervals of the four variables. Here $\mathcal{X}$ is regarded as a subset of $\Re^4$ forcibly though it does not correspond to the actual system of the Acrobot. We divide $\mathcal{X}$ in a reticular pattern and create discrete states. Each discrete state is identified by a combination of intervals $[\theta_1]_i, [\theta_2]_j, [\dot{\theta}_1]_k$ and $[\dot{\theta}_2]_h$, which are defied in Table 5.3. The variables are divided by 10[deg] or 10[deg/s] respectively. Though neighboring intervals overlap at their ends, it is not a problem in the implementation. The number of discrete states reaches $36 \cdot 36 \cdot (36 \cdot 4) \cdot (36 \cdot 9) = 60,466,176$. However, the Acrobot is symmetric. Giving torque τ at $(\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2)$ is equivalent to giving torque $-\tau$ at $(-\theta_1, -\theta_2, -\dot{\theta}_1, -\dot{\theta}_2)$. In computation, the number of states can be reduced to $60,466,176/2 = 30,233,088$ when we utilize this

symmetric property. The index of each state is defined as follows:

$$\begin{aligned} i &= 36 \cdot 144 \cdot 324 i_{\theta_1} + 144 \cdot 324 i_{\theta_2} + 324 i_{\dot{\theta}_1} + i_{\dot{\theta}_2} \\ &= 1,679,616 i_{\theta_1} + 46,656 i_{\theta_2} + 324 i_{\dot{\theta}_1} + i_{\dot{\theta}_2}, \end{aligned} \tag{5.5}$$

where i_{θ_1}, i_{θ_2}, $i_{\dot{\theta}_1}$, and $i_{\dot{\theta}_2}$ denote the index of the state variables of θ_1, θ_2, $\dot{\theta}_1$, and $\dot{\theta}_2$ respectively.

Table 5.3: Discretization of State Space

	definition of intervals	
θ_1	$[\theta_1]_i \equiv [10i, 10(i+1)]$[deg]	$(i = 0, \ldots, 17)$
θ_2	$[\theta_2]_i \equiv [10i, 10(i+1)]$[deg]	$(i = 0, \ldots, 35)$
$\dot{\theta}_1$	$[\dot{\theta}_1]_i \equiv [10i - 720, 10i - 710]$[deg/s]	$(i = 0, \ldots, 143)$
$\dot{\theta}_2$	$[\dot{\theta}_2]_i \equiv [10i - 1620, 10i - 1610]$[deg/s]	$(i = 0, \ldots, 323)$

Final States and Their Values

In our code, two types of final states are defined. We define $\mathcal{S}_{\text{success}} \subset \mathcal{S}$ as the set of final states at which the task is completed. $\forall s \in \mathcal{S}_{\text{success}}$ satisfies $y_2 \geq h$ at any continuous state in s.

When one of the angular velocities is ont of its domain, the task is regarded as a failed trial. We consider that the state of the Acrobot is in another kind of final state, which is called a failure state in this paper. We define $\mathcal{S}_{\text{failure}}$ as the set of failure states ($\mathcal{S}_{\text{failure}} \notin \mathcal{S}$).

We set $V(s) = 0$ to $\forall s \in \mathcal{S}_{\text{success}}$ and $V(s) = -1000$[s] to every state in $\mathcal{S}_{\text{failure}}$. The value for states in $\mathcal{S}_{\text{failure}}$ does not make a large difference on state-action maps if it is a large negative value.

State Transitions and Their Reward

Each discrete states and torques are related to each other as $\mathcal{P}^{\tau}_{ss'}$ ($\forall s \in \mathcal{S} - \mathcal{S}_{\text{f}}, \forall s' \in \mathcal{S}, \forall \tau \in \mathcal{A}$). Every $\mathcal{P}^{\tau}_{ss'}$ is calculated as follows in our code.

At first, four continuous states $\boldsymbol{x}_i \in s$ ($i = 1, 2, 3, 4$) are chosen. When s is defined as the direct product of $[\theta_1]$, $[\theta_2]$, $[\dot{\theta}_1]$ and $[\dot{\theta}_2]$, the values of $(\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2)$ of each continuous state are fixed as follows.

- θ_1 and θ_2: median values of $[\theta_1]$ and $[\theta_2]$ respectively
- $\dot{\theta}_1$ and $\dot{\theta}_2$: maximum/minimum values of $[\dot{\theta}_1]$ and $[\dot{\theta}_2]$

There are four combinations of $(\dot{\theta}_1, \dot{\theta}_2)$. Each combination is assigned to each $\boldsymbol{x}_i$ respectively. The translations of $\boldsymbol{x}_i$ $(i = 1, 2, 3, 4)$ are calculated based on Eq.(5.1) and (5.2). We use the 4th order Runge-Kutta method for computing the posterior states after 0.2[s] when torque τ is given in the time interval. The discrete states in which the posterior states exist are obtained. We name them s'_i $(i = 1, 2, 3, 4)$ respectively. The transition probabilities are defined as

$$\mathcal{P}^{\tau}_{ss'_i} = \frac{1}{4} \qquad (i = 1, 2, 3, 4). \tag{5.6}$$

More continuous states from s should be chosen if we want to calculate the transition probabilities more accurately. It is very important to enhance the efficiency and robustness of state-action maps. However, we try creating state-action maps with the above way in this paper.

Rewards should be computed based on Eq.(1.9). Since time for one transition is fixed 0.2[s] in the above definition, the reward is set to

$$\mathcal{R}^{\tau}_{ss'} = -0.2[\text{s}]. \tag{5.7}$$

5.3.2 Obtained State-Action Maps

We obtain a state-action map for $h = 1.9$. Value iteration is continued until the maximum difference of the both side of Eq.(2.11) is larger than 10^{-5} on a sweep. Total time for creating the map was 17 hours on a computer that has 1.5GB RAM and a 1.5GHz CPU. Time for every sweep was 643[s] and 95 sweeps were required for convergence.

Some parts of the state-action map are shown in Fig. 5.2 and 5.3 with each related part of the state-value function. Every partial map, which is shown in the left side, is a slice of the four-dimensional state-action map π_{MAP}. A pair of $([\theta_1], [\theta_2])$ is fixed and torques for all combinations of $([\dot{\theta}_1], [\dot{\theta}_2])$ are illustrated as gray scale images. Black, gray, and white denote $1, 0, -1$[Nm] respectively.

Each slice of the state-value function V, shown in the right side, is a gray scale image too. White and black denotes $V(s) = 0$[s] and $V(s) \leq -60$[s] respectively. Density of gray is in proportion to the value in the range

between $0 \geq V(s) \geq -60$[s].

In Fig. 5.2, we choose some slices in the area of $0 \leq \theta_1 < 10$[deg] and $0 \leq \theta_2 < 110$[deg]. As the value of θ_2 varies in the intervals, the marble pattern of π_{MAP} changes fluently. In each slice of the state-value function, the black color is filled in the upper right parts and in the lower left part. It means that π_{MAP} cannot lead the states in this area to $\mathcal{S}_{\mathrm{success}}$ within 60[s]. In the states of those parts, two joints of the Acrobot have high angular velocity in the same direction. It is difficult to reduce both of $\dot{\theta}_1$ and $\dot{\theta}_2$ within the limitations of velocity from that state.

Nevertheless, the Acrobot requires velocity for swinging up. We can see some spots or areas of light grey color in the black part. Each spot is a cut plane of a tube in the four dimensional state space. When the acceleration is successfully done, the state of the Acrobot runs through one of the four-dimensional tubes and reaches to $\mathcal{S}_{\mathrm{success}}$.

Figure 5.3 shows some slices in the area of $170 \leq \theta_1 < 180$[deg] and $150 \leq \theta_2 < 360$[deg]. In this area, the tip of Link 1 is swung upward. The pattern of π_{MAP} in this area is different from that in the area of Fig. 5.2. Every slice in Fig. 5.2 has a black area at the lower right part of the origin and a white area at the upper left part of the origin.

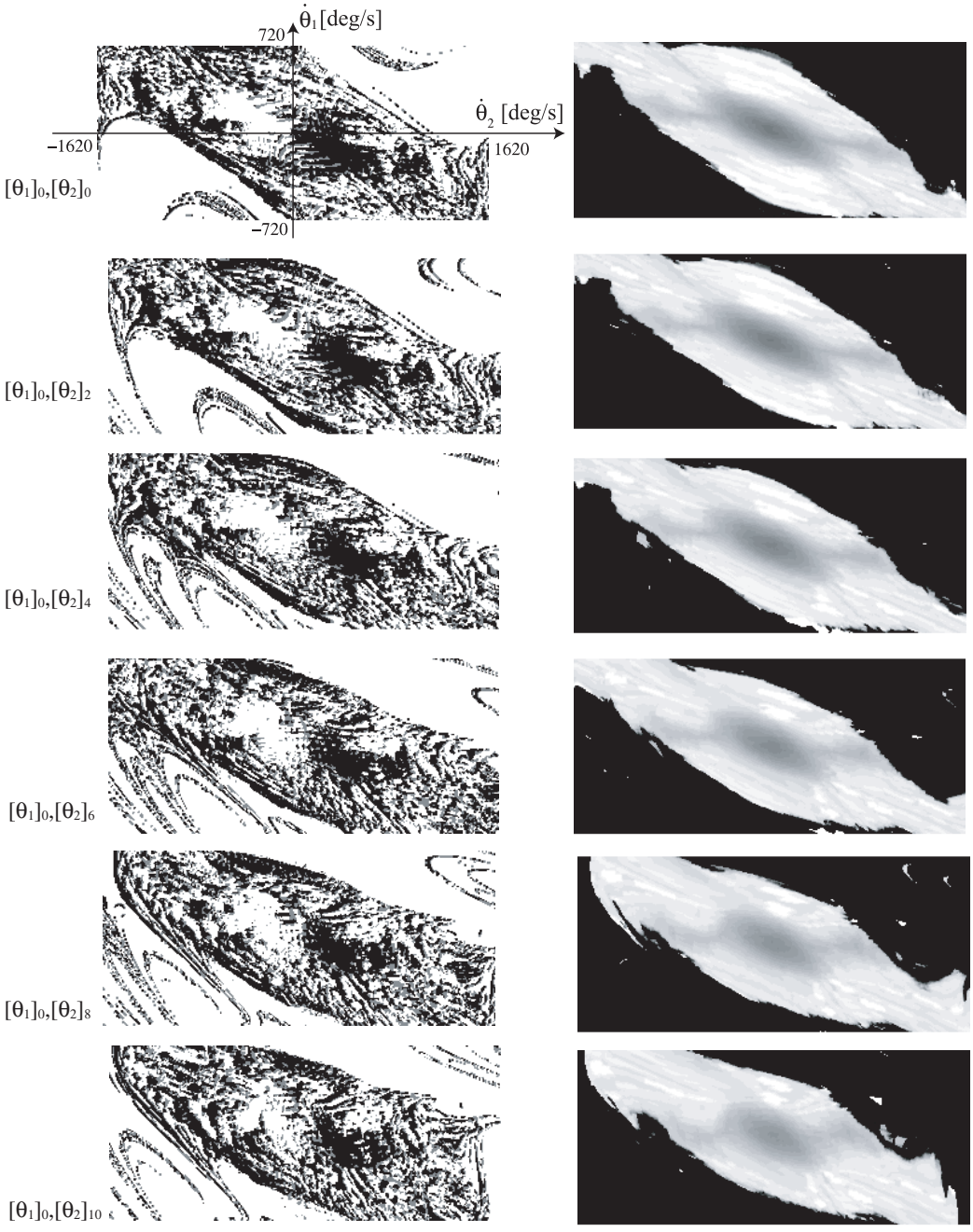


Fig. 5.2: Slices of State-Action Map and State-Value Function (Link 1 trails down.)

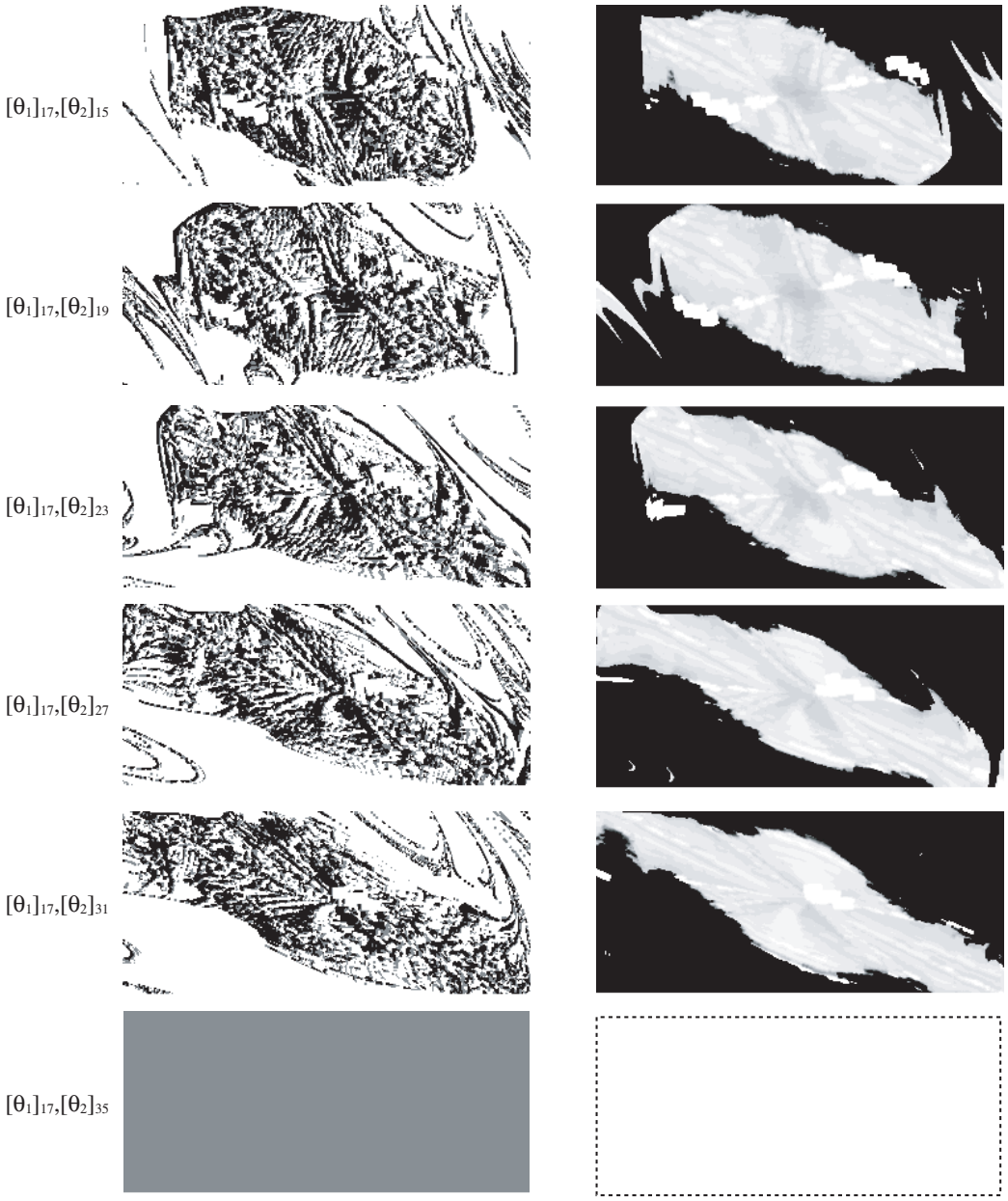


Fig. 5.3: Slices of State-Action Map and State-Value Function (Link 1 swings upward.)

5.3.3 Behavior with The State-Action Map

We observe behavior of the Acrobot from several initial states by using this state-action map. The dynamics of the Acrobot is simulated by the 4th order Runge-Kutta method. The time interval for computation is set to 0.02[s], which is one-tenth of the time interval for decision of torque.

In Fig. 5.4 and 5.5, the motion of the Acrobot from $(\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2) = (0, 0, 0, 0)$ and that from $(\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2) = (0.1[\text{deg}], 0, 0, 0)$ are illustrated respectively. Figures are drawn as sets of poses in a half cycle of Link 1. Note that Link 1 is swung by torque τ at Joint 2.

From Fig. 5.4, we can observe the following motion.

- $0 \leq t \leq 15.2$][s]: pendular movement occurs. The Acrobot is given momentum.
- $15.2 < t \leq 16.8$[s]: Link 2 passes over Link 1.
- $16.8 < t \leq 19.8$[s]: the pendular movement occurs again.
- $19.8 < t \leq 21.6$[s]: Link 2 passes over Link 2 twice. The motion can be likened to baton twirling. In this case, Link 1 and 2 are the arm of a twirler and the baton respectively.
- $21.6 < t \leq 22.8$[s]: The Acrobot stretches its body and the task is just finished when its body straightens.

Such a strategic motion can be obtained from only one state-action map. From some other initial states, five kinds of characteristic motions shown in Fig. 5.6 can be observed. Various complicated motions that cannot be described in words are also observed.

Though these motion are not the best for this task as mentioned later, the state-action map can bring the Acrobot to the goal by using some interesting modes of motion. Moreover, we notice that the modes are changed by a small difference of initial states when Fig. 5.4 and 5.5 are compared. Since the state-action map prepares all torques toward all states, it can deal with the chaotic nature of the Acrobot.

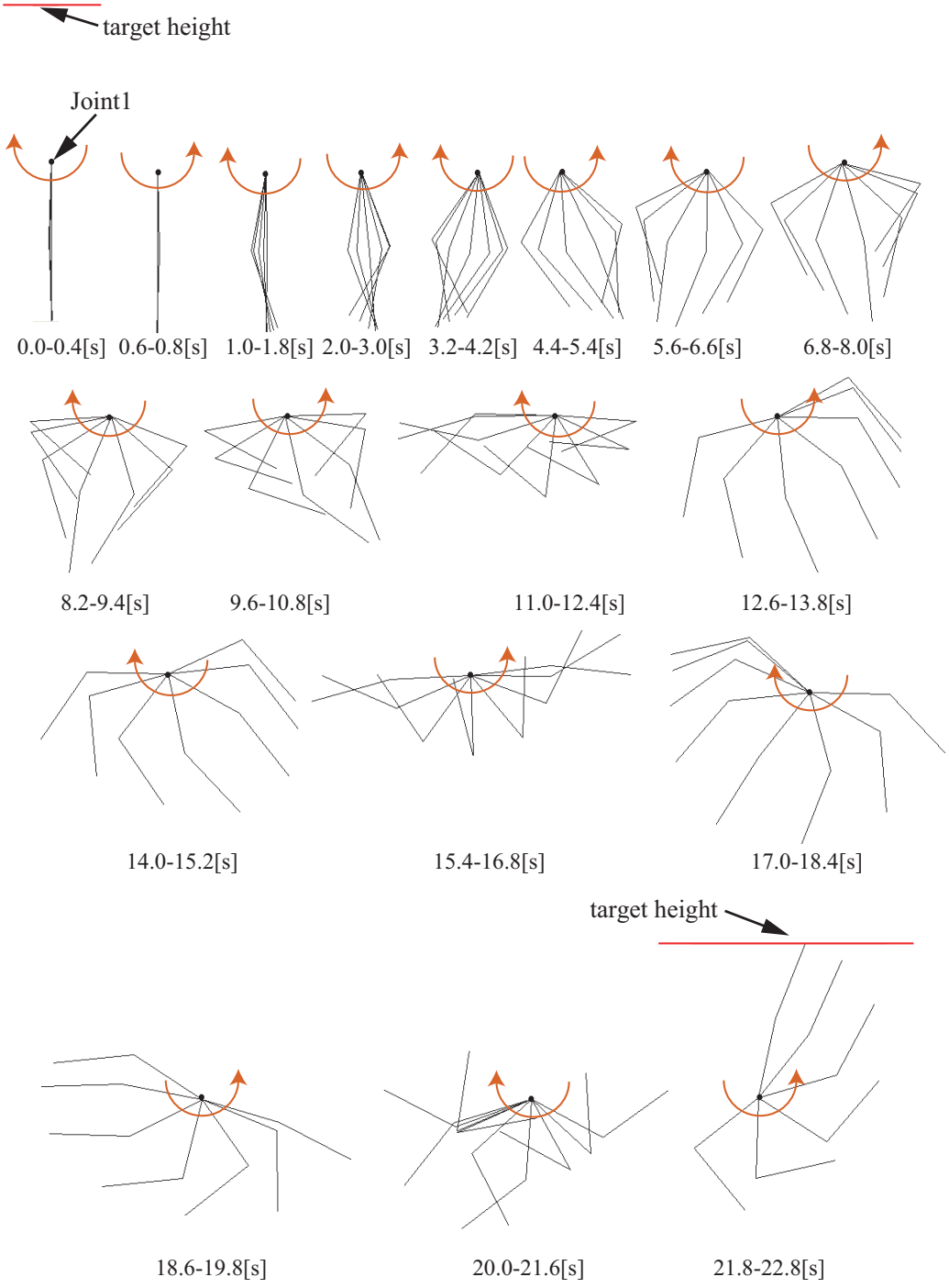


Fig. 5.4: Motion from $(\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2) = (0, 0, 0, 0)$

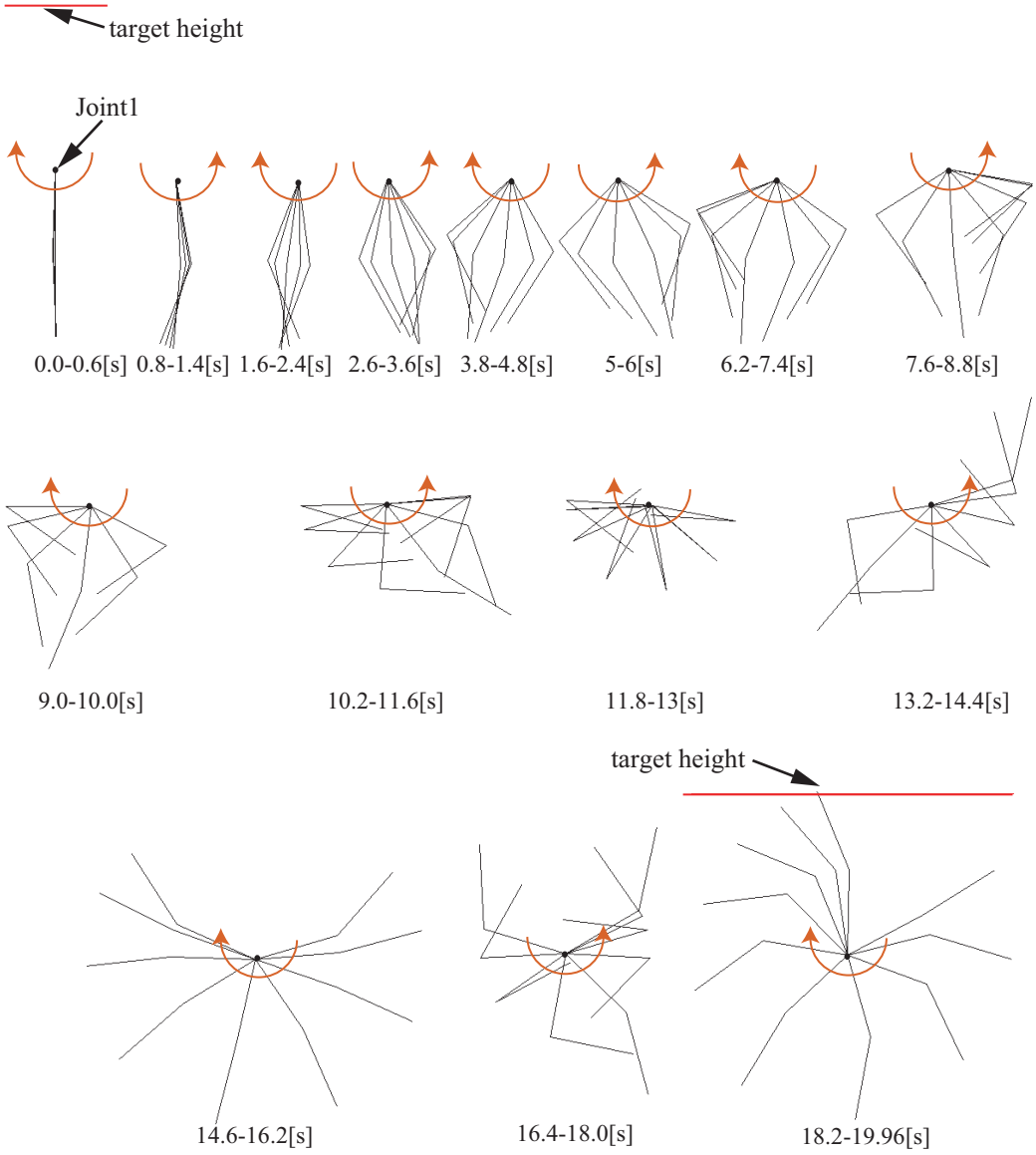


Fig. 5.5: Motion from $(\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2) = (0.1[\text{deg}], 0, 0, 0)$

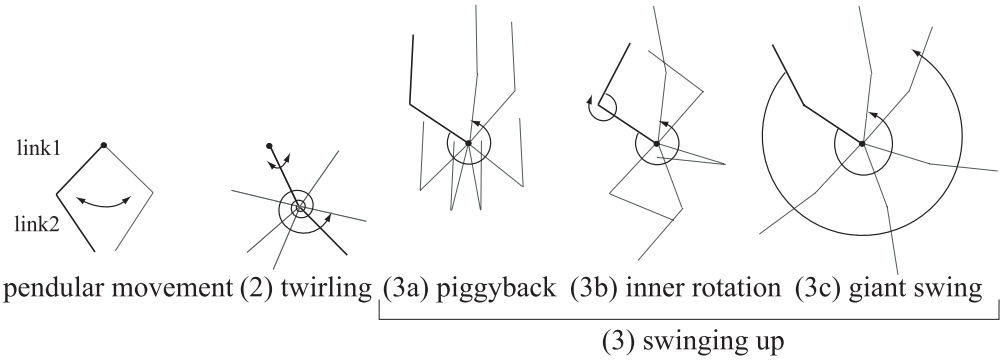


Fig. 5.6: Typical Motions of the Acrobot with the State-Action Map

5.3.4 Evaluation of The Density

The obtained state-action map, which is named a normal map here, is compared with other state-action maps that have different density of discretization. This evaluation is important to verify the adequacy of the discretization in Table 5.3. In other words, we evaluate whether the efficiency of control is obtained or not in compensation for the huge use of memory here.

In a trial of the simulation, the time from an initial state to a final state in $\mathcal{S}_{\text{success}}$ or $\mathcal{S}_{\text{failure}}$ is recoded. The trials are held with various initial states. We prepare the following set of initial states: $(\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2) = (3i[\text{deg}], 3j[\text{deg}], 0, 0)$ $(i, j = 0, 1, 2, \ldots, 119)$. All final states are removed from the set and $14,167$ trials are held for each map.

We make the following state-action maps that have twice or half numbers of intervals on some axes than the normal map.

- **dense map A:** dense on θ_1 axis
- **dense map B:** dense on θ_1 and θ_2 axes
- **dense map C:** dense on θ_1, θ_2, and $\dot{\theta}_1$ axes
- **coarse map a:** coarse on θ_1 axis
- **coarse map b:** coarse on θ_1 and θ_2 axes

A minimum map whose size is only 2[bit] is also prepared. This map gives

the following policy:

$$\tau = \begin{cases} -1[\mathrm{Nm}] & \text{if } \dot{\theta}_1 > 0 \\ 1[\mathrm{Nm}] & \text{otherwise.} \end{cases} \tag{5.8}$$

By this policy, the pendular movement in Fig. 5.6(1) is enabled when the energy is low.

They are evaluated by the simulation. The average time and worst time in all of the trials are written in Table 5.4. In the case of the minimum map, the state of the Acrobot reaches to a failure state in $\mathcal{S}_{\text{failure}}$ at 717 (5[%]) trials. 658 (5[%]) failure trials occur in the case of the coarse map b.

Table 5.4: Comparison with Dense/Coarse Maps and Minimum Map

map/method	size [bit]	average [s]	worst [s]
dense map C	484M	8.8	36.1
dense map B	242M	9.4	31.1
dense map A	121M	10.2	34.1
normal map	60.5M	11.3	39.6
coarse map a	30.2M	14.2	55.3
coarse map b	15.1M	18.0	failure
minimum map	2	17.0	failure

It is the most important thing that all of the trials with the five large maps have finished successfully. There is no overlimit of the angular velocities and no unfinished trial in all of the trials. Though the motion of the Acrobot is chaotic, feasible state-action maps can be created by DP. On the other hand, the performance of the coarse map b is worse than that of the minimum map. We need to invest a large amount of memory for creating a state-action map that can outperform the 2-bit map. Of course there is a possibility that the amount of memory can be reduced by a refinement of the value iteration algorithm in Sec. 5.3.2.

5.4 Compression of State-Action Maps

5.4.1 Algorithm for Obtaining VQ Maps

The obtained state-action map is compressed by our VQ method. At first, the map is sliced by the same way as it was done in Fig. 5.2. Each slice is regarded as a block, or as a vector. When this blocking is explained with the state indexes defined in Eq. (5.5),

$$i_\nu = \nu(i) = \lfloor i/46656 \rfloor \quad (i_\nu = 0, 1, 2, \ldots, 647) \tag{5.9}$$

$$i_\varepsilon = \varepsilon(i) = i\%46656 \quad (i_\varepsilon = 0, 1, 2, \ldots, 46655). \tag{5.10}$$

The vectors are classified into N_c clusters. Vectors in each cluster are replaced by a representative vector and a VQ map is created.

We use the Lloyd algorithm that is shown in Fig. 3.9 for clustering. For reducing the computation time, all of the distortions $d(s, a)$ are previously calculated and stored on a file, whose size is $NM \cdot 64 = 5,804,752,896$[bit]. 64 bits are used for storing a value of distortion. Before the Lloyd algorithm starts, this file is loaded from a hard-disk drive (HDD).

When values of distortion are calculated, the distortion at every state whose value is worse than -100[s] is regarded as zero toward any change of torque. Since the Acrobot may not reach $\mathcal{S}_{\text{success}}$ from these states, we do not care what is written in the map.

Table 5.5: Computation Time (with 1.5 GHz Pentium M CPU, 1.5GB RAM)

N_c	time [min]	# of iterations
256	31.9	2
128	16.4	2
64	8.1	2
32	4.3	2
16	2.3	2
8	3.4	5
4	3.4	8
2	7.7	25
1	0.6	2
computation of all $d(s, a)$	5.9	—

We create VQ maps with $N_c = 2^i$ $(i = 0, 1, 2, \ldots, 8)$. A VQ map that is created with i representative vectors is named the $N_c = i$ map here. As shown

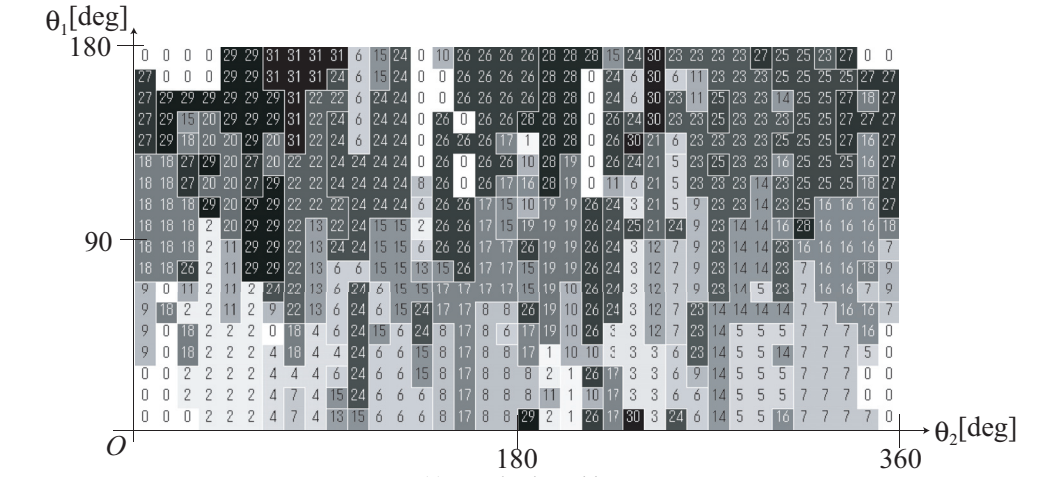
in Table 5.5, it takes 5.9[min] to calculate and store the values of distortion. The number of iteration is only two when $N_c = 1, 16, 32, 64, 128, 256$.

5.4.2 Obtained VQ Maps

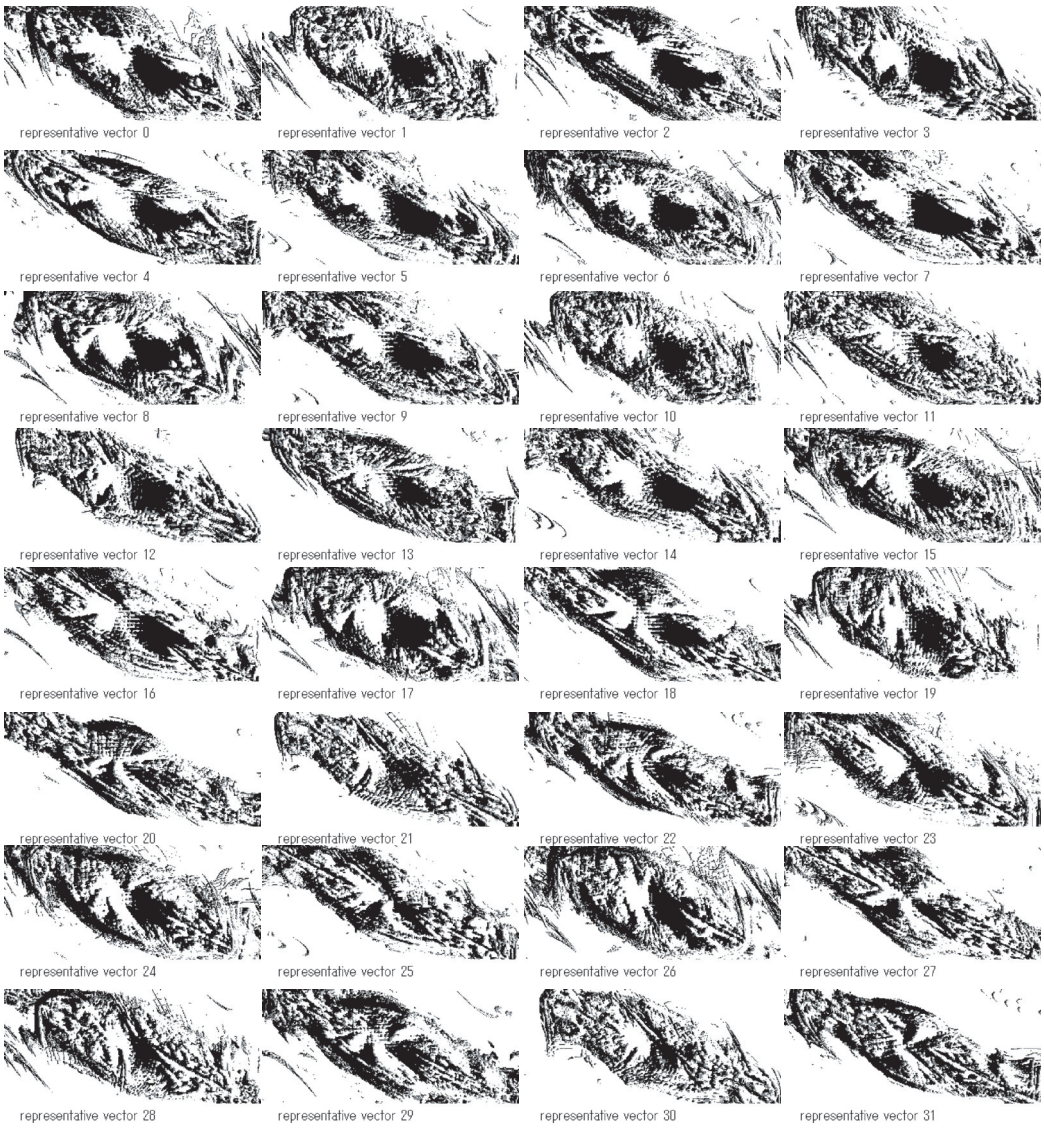
The $N_c = 32, 16, 8, 4, 2, 1$ maps are shown in Fig. 5.7-5.12 respectively. (a) and (b) in each figure except Fig. 5.12 represent the quantization table and representative vectors respectively. In the case of Fig. 5.12, no quantization table is shown because it is not necessary to choose only one representative vector. Each quantization table in (a) is regarded as a two-dimensional table in $\theta_1\theta_2$-space. The representative vector index is allocated every cell of $\theta_1\theta_2$-space. Every representative vector in (b) belongs to $\dot{\theta}_1\dot{\theta}_2$-plane as in the case of vectors in Fig. 5.2-5.3.

The smaller N_c is, the more the representative vectors are averaged as shown in (b) of each figure. The representative vector in Fig. 5.12 is the ultimate in the simplification.

In the case of $N_c = 1$, the only one representative vector can be regarded as a state-action map on $\dot{\theta}_1\dot{\theta}_2$-plane. From Fig. 5.12, we can estimate rough motion of the Acrobot with the $N_c = 1$ map. There are white ($\tau = -1$[Nm]) part and black ($\tau = 1$[Nm]) part around the point of $(\dot{\theta}_1, \dot{\theta}_2) = (0, 0)$. In this part, the Acrobot will the torque in the travelling direction of Link 2. The motion of the Acrobot seems to be the pendular movement in Fig. 5.6. The black and white pattern is reversed at the outside of the pendular movement area. When the pair of angular velocities of Link 1 and Link 2 belongs to the outside area, the torque is added to the reverse direction of Link 2's motion. It seems that the Acrobot tries to stretch its links in a straight line.

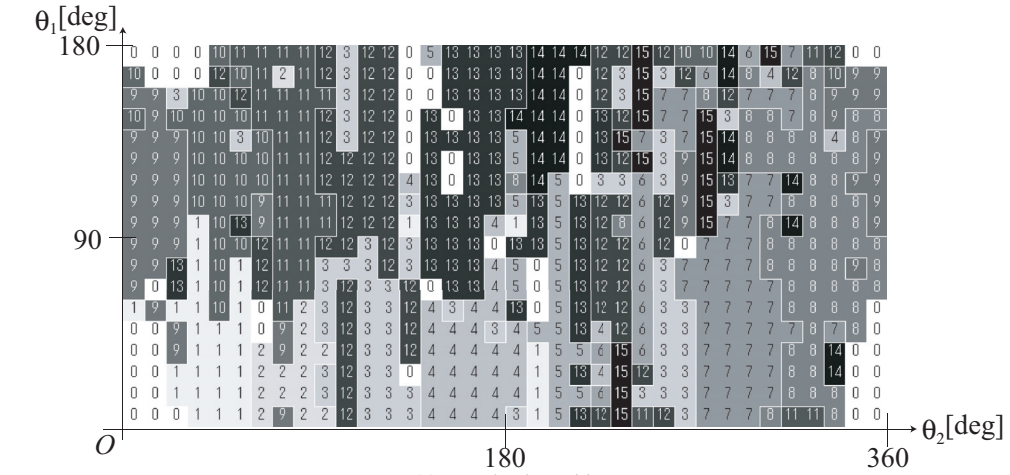


(a) quantization table

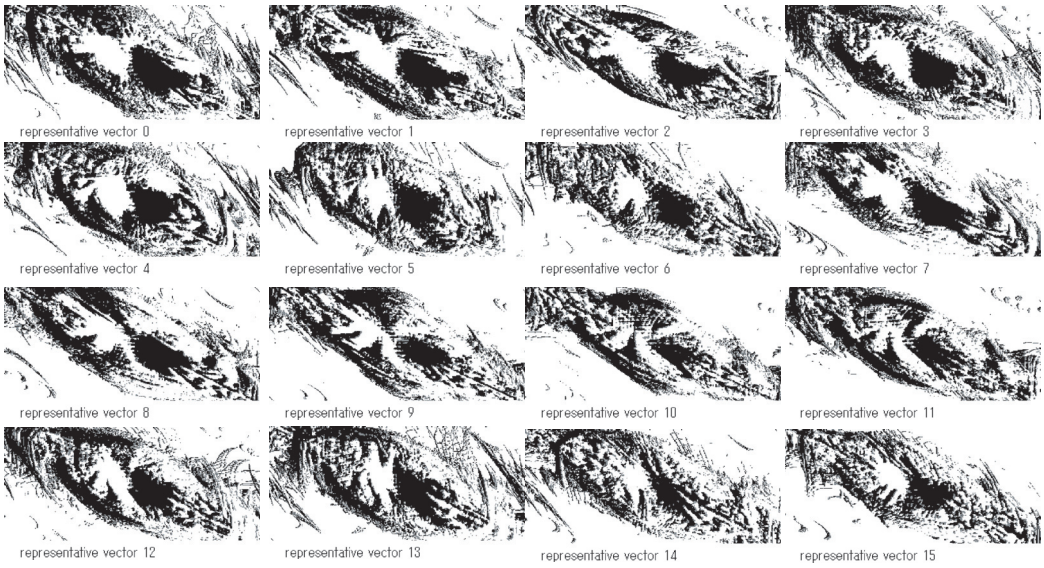


(b) representative vectors (black: τ=1[Nm], gray: τ=0[Nm], white: τ=−1[Nm])

Fig. 5.7: $N_c = 32$ Map

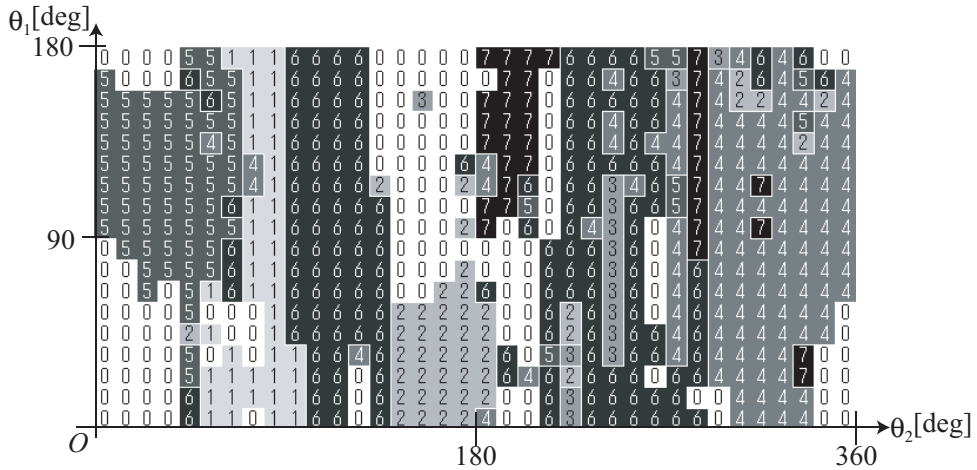


(a) quantization table

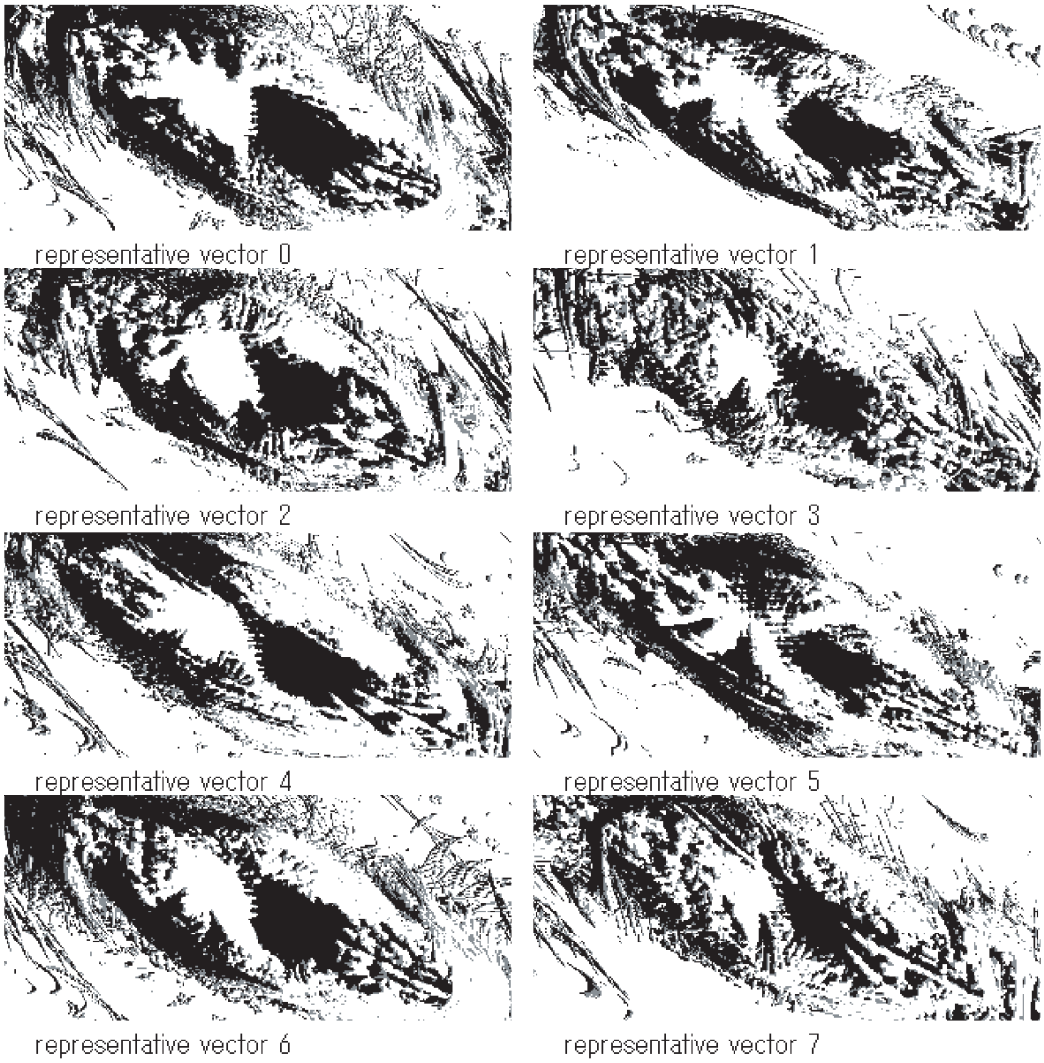


(b) representative vectors (black: τ=1[Nm], gray: τ=0[Nm], white: τ=–1[Nm])

Fig. 5.8: $N_c = 16$ Map

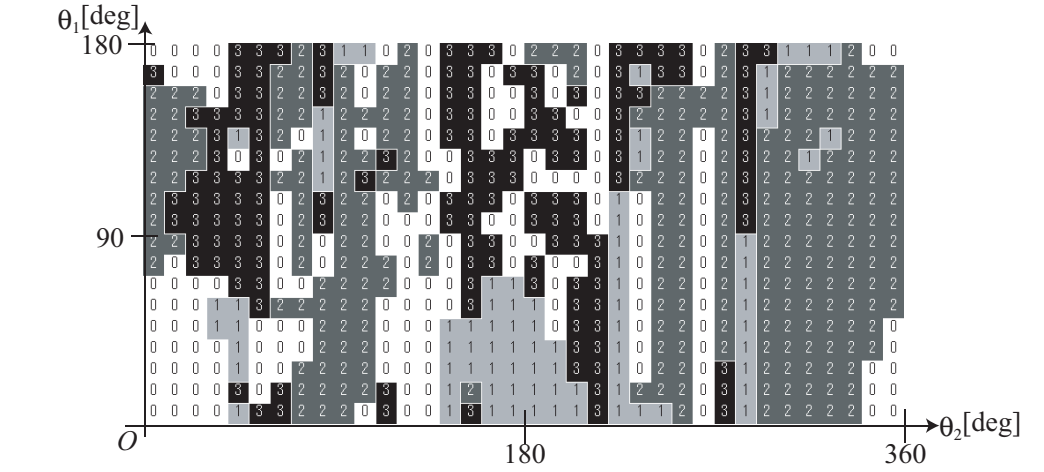


(a) quantization table

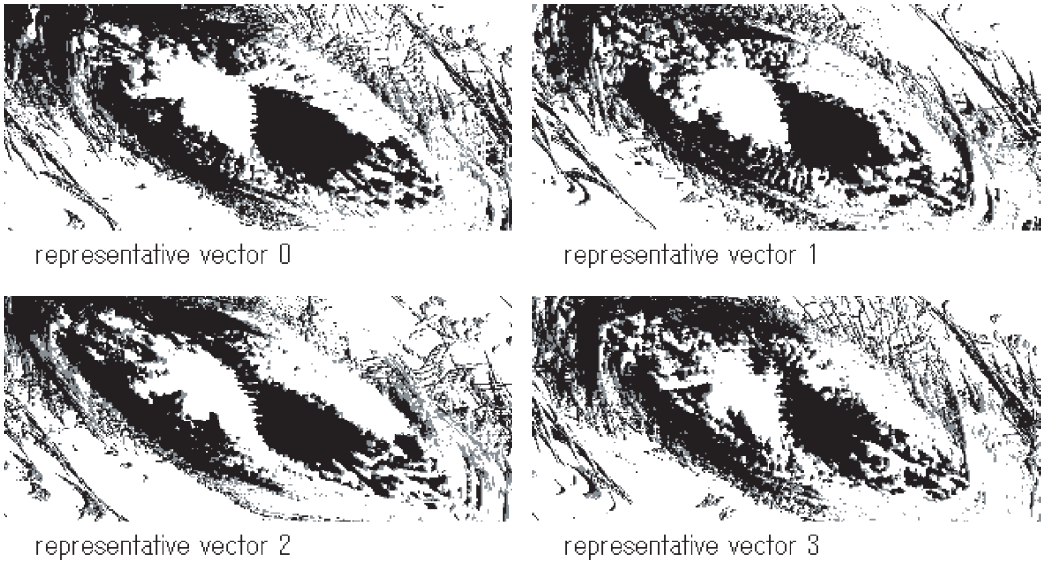


(b) representative vectors (black: τ=1[Nm], gray: τ=0[Nm], white: τ=−1[Nm])

Fig. 5.9: The $N_c = 8$ Map

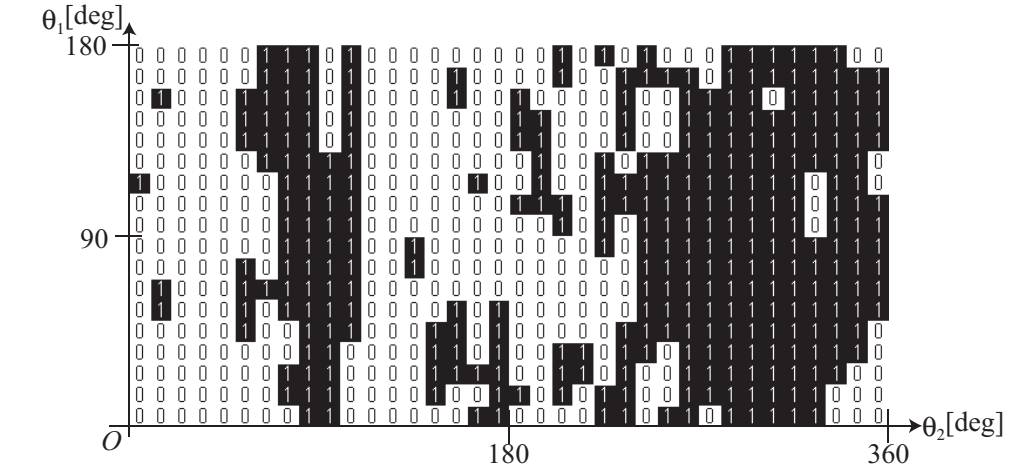


(a) quantization table

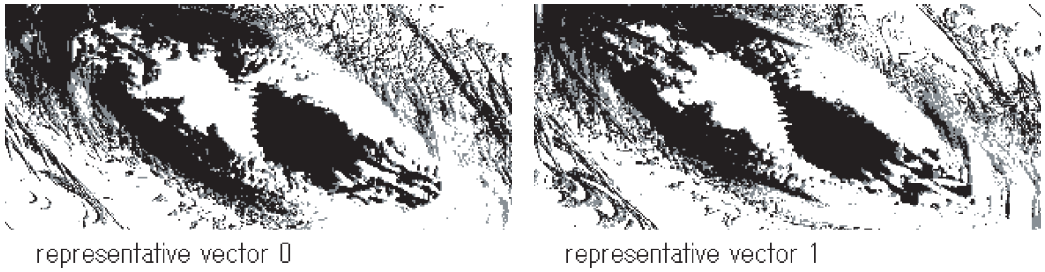


(b) representative vectors (black: τ=1[Nm], gray: τ=0[Nm], white: τ=−1[Nm])

Fig. 5.10: $N_c = 4$ Map



(a) quantization table



(b) representative vectors (black: τ=1[Nm], gray: τ=0[Nm], white: τ=−1[Nm])

Fig. 5.11: $N_c = 2$ Map

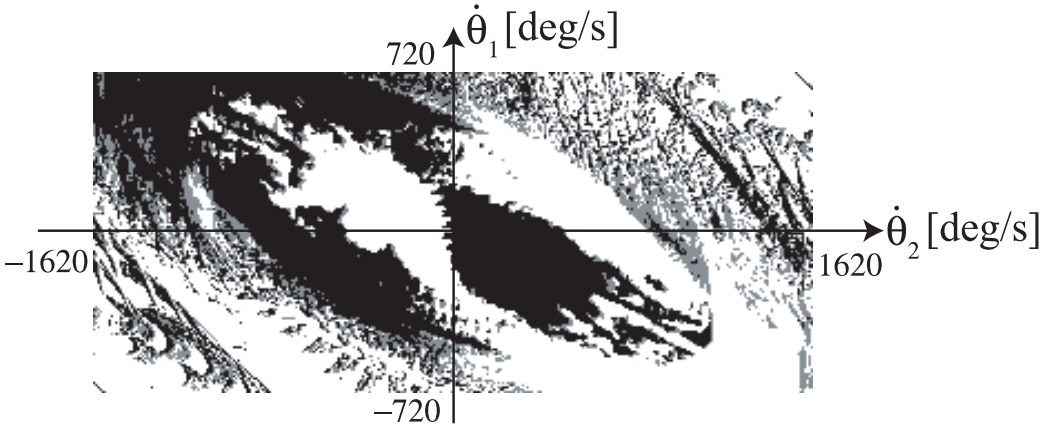


Fig. 5.12: The Representative Vector when $N_c = 1$ (black: $\tau = 1$[Nm], gray: $\tau = 0$[Nm], white: $\tau = -1$[Nm])

5.4.3 Comparison of Motion with Uncompressed Map

Observation of The Motion

In Fig. 5.14, we show sequences of torque τ and height y_2 on trials with four kinds of pair of an initial state and a map. The sequences in (a) and (b) are obtained by the uncompressed map and the others are obtained by the $N_c = 1$ map. The initial states are $(\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2) = (0, 0, 0, 0)$ in (a) and (c), and $(\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2) = (1[\text{deg}], -1[\text{deg}], 0, 0)$ in the others. In the figures of y_2, types of motion that are observed in some time periods are written in the time-y_2 graphs as the numbers defined in Fig. 5.6.

The pendular movement shown in Fig. 5.6(1) can be always observed in the first ten seconds of all of the trials. However, behavior of the Acrobot is changed by the small difference of initial states after 10[s]. Since the influence of the difference of initial states is large, the difference of the uncompressed map and the VQ map cannot be observed from the figures.

Evaluation of VQ Maps

The VQ maps in Sec. 5.4 are evaluated by the simulation. The results are shown in Table 5.6. Comparison to the results of uncompressed maps on Table 5.4 is illustrated in Fig. 5.15. As shown in this figure, the average time of every VQ maps is shorter than that of the coarse map a. Though the coarse map is 324 times as large as the $N_c = 1$ map, the average time of the average time with the VQ map is 0.5[s] shorter than that of the coarse map.

Though the $N_c = 1$ map considers only $\dot{\theta}_1$ and $\dot{\theta}_2$, this map is created from the state-value function in the four-dimensional state space. It is significant that the VQ method can create a control policy whose dimension is reduced within the framework of optimal control.

As shown in Table 5.6, the efficiency of the $N_c = 1, 2, 4$ maps are not different to each other. As shown in Fig. 5.11 and 5.10, the representative vectors are similar to the VQ map in Fig. 5.12. Therefore, it is understandable that their efficiencies are also similar to the efficiency of the $N_c = 1$ map.

Every representative vector of the VQ maps with $N_c \leq 4$ is similar to vectors that are used when the tip of the Acrobot is in a low position. In other words, the compression method discards the control policy at states where the tip is high. We can recognize it when Fig. 5.2, 5.3, 5.12, 5.11,

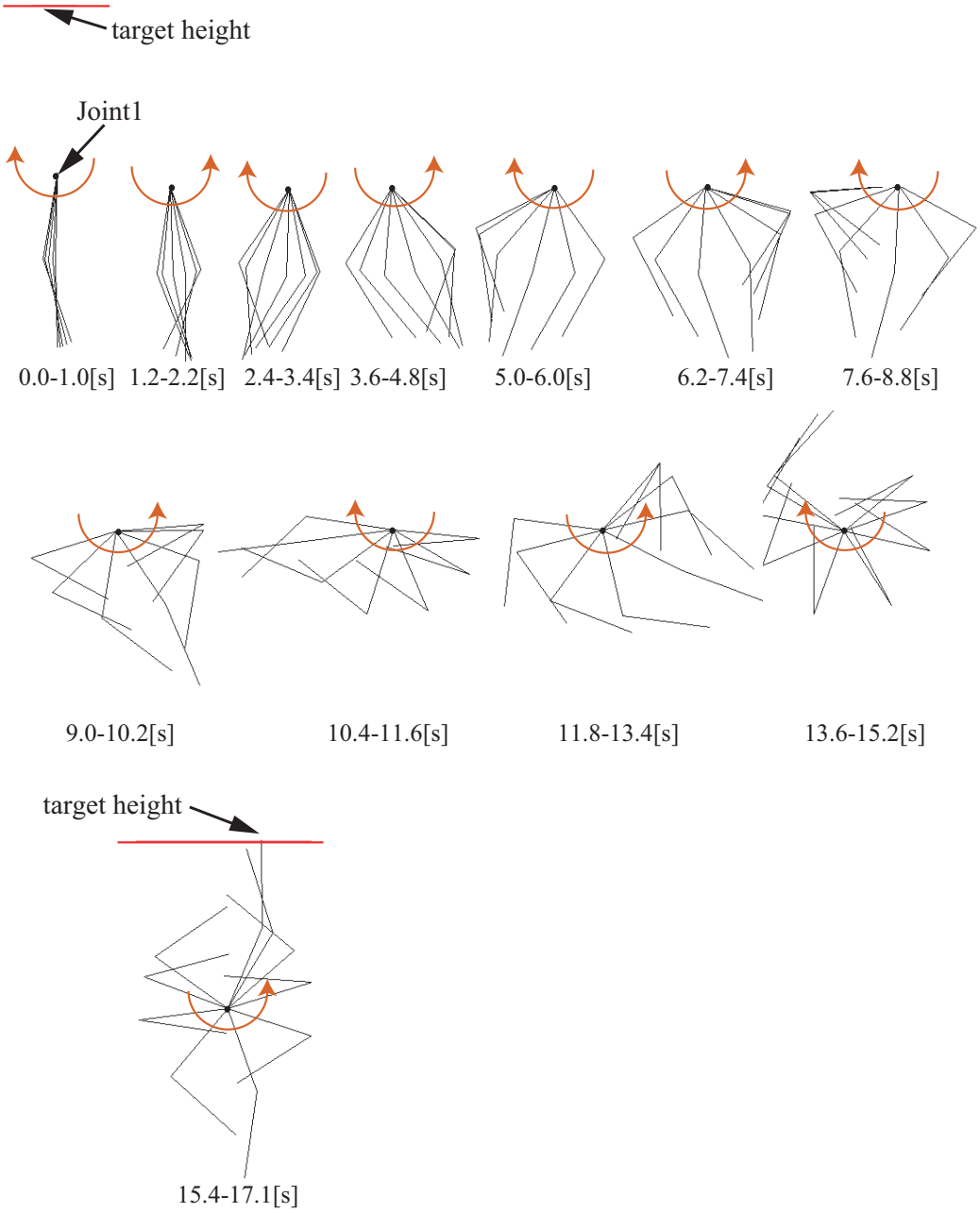


Fig. 5.13: Motion from $(\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2) = (0, 0, 0, 0)$

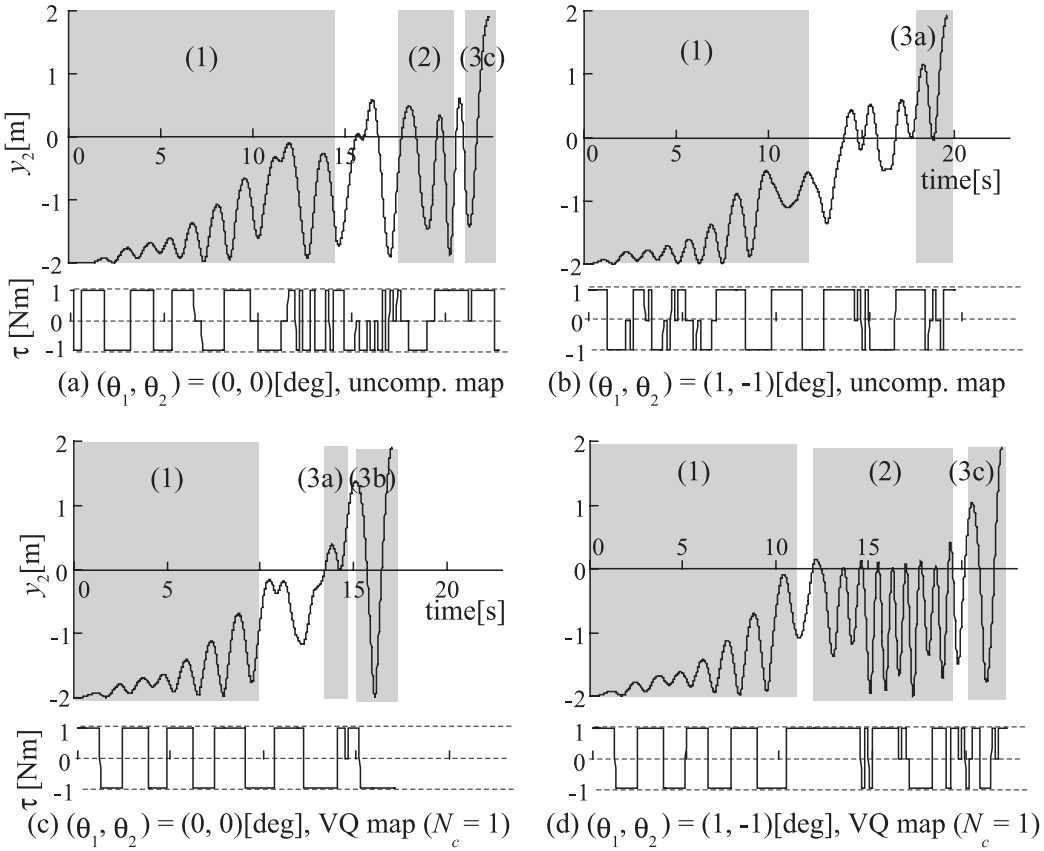


Fig. 5.14: Record of Torque and Motion

Table 5.6: Evaluation Result of VQ Maps

N_c	size [bit]	average [s]	worst [s]
256	23.9M	11.4	45.6
128	11.9M	11.3	60.4
64	5.98M	11.8	51.6
32	2.99M	12.4	63.5
16	1.50M	12.8	78.7
8	748k	13.4	55.5
4	375k	13.7	71.3
2	187k	13.7	62.1
1	93.3k	13.7	85.0

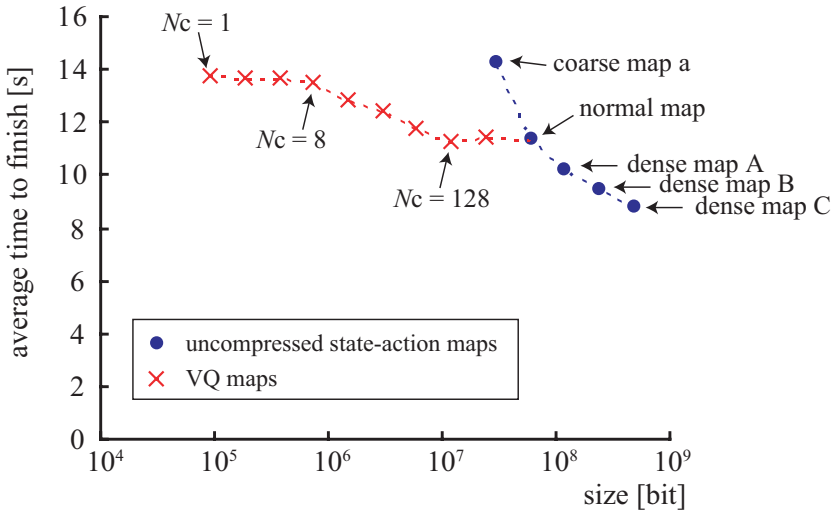
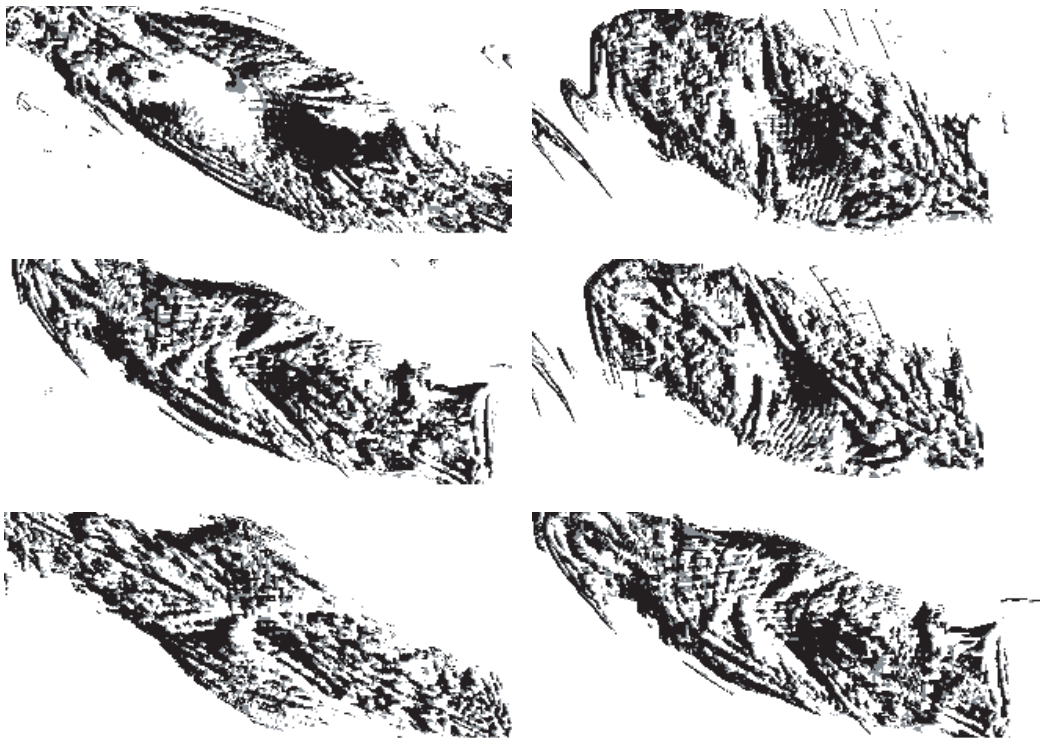


Fig. 5.15: Comparison between Uncompressed Maps and VQ Maps

and 5.10 are compared. The uncompressed state-action map adjusts the control policy as appropriate according to the pose of the Acrobot. On the other hands, the VQ maps consider only the angular velocities for the control.

The $N_c = 8$ map in Fig. 5.9 has various representative vectors differently from the $N_c = 2, 4$ maps. The representative vector $\boldsymbol{c}_5$ particularly has a different appearance to the representative vectors of the $N_c = 1, 2, 4$ maps. $\boldsymbol{c}_5$ is mainly allocated to the $90 \leq \theta_1 < 180$[deg] and $0 \leq \theta_2 < 70$[deg] area. It seems that $\boldsymbol{c}_5$ is a special control policy when the height of the tip is high. Such a representative vector is cut down in the VQ map with $N_c \leq 4$. We show some of representative vectors of the $N_c = 128$ map. As shown in this figure, representative vectors vary more widely than those of the VQ maps that have smaller N_c.



(black: τ=1[Nm], gray: τ=0[Nm], white: τ= –1[Nm])

Fig. 5.16: Representative Vectors of the $N_c = 128$ Map

5.5 Discussion

In this section, we have applied dynamic programming to swinging up control of the Acrobot and its result has compressed by our vector quantization method.

Summary of Simulation Result

In the simulation, we have obtained the following result.

- Though the obtained state-action maps by DP are chaotic, they can give policies that make the Acrobot finish the swinging up task successfully from all of the static ($\dot{\theta}_1 = \dot{\theta}_2 = 0$) states.
- The VQ maps which also give feasible policies can be obtained. The VQ map with $N_c = 1$ is 93.3kB and its compression ratio is $1 : 0.0015$.
- The compression ratio of the VQ map is $1 : 0.0031$ toward the coarse map a, which marks worse evaluation than the VQ map.

Discussion from the Result

A chaotic but performable control policy can be obtained in the global state space of the Acrobot by DP. Moreover, we can verify that our VQ algorithm can reduce the dimension of the control policy from four to two in the process of compression. It is significant that its dimension is reduced by the VQ method.

Though the swinging up task is procedural and complicated one, its secret of success is how to enhance the energy of the Acrobot. We think that the success of the experiment in this chapter depends on that simple principle in some degree. However, the consideration of energy is not done in our method. In other words, our method can implicitly utilize the principle through the functional and the state-value function.

Discussion about Application for Actual Robot in Future

We should pay attention to the measuring errors of the parameters and the state variables shown in Table 5.2. We have evaluated the loss of efficiency toward some kinds of parametric errors in Sec. A.3. Though more cases should be tried for certain evaluation, we are optimistic about the robustness from the tables in Sec. A.3.

Chapter 6

Application and Evaluation II: RoboCup

RoboCup (robot soccer world cup) was proposed mainly by Japanese researchers [Kitano, 1997; Veloso, 1998; Asada, 1999] as a novel standard problem for artificial intelligence (AI) and robotics. The world cup has been held annually since 1997 and its scale of operation has been expanded year by year. In RoboCup 2005 Osaka, for instance, 330 teams participated from 31 countries.

As we have explained in Chapter 1, nowadays AI on computers can defeat human champions in chess and backgammon. However, it does not mean that the AI can perform some jobs in the real world. The RoboCup project is just on trying making AI act in the real world with the grand challenge problem: *By the year 2050, develop a team of fully autonomous humanoid robots that can win against the human world soccer champion team.*

In this chapter, the proposed method is applied to tasks of robots in RoboCup four legged robot league. In this league, ERS-7 and ERS-210 are used for competition. They have only 16MB of flash memory for installing the executable code that is complied by another computer. In other word, all of executable codes and data should be stored on the limited amount of memory.

This chapter is composed of seven sections. RoboCup four legged robot league, which is the test environment in this chapter, is explained in Sec. 6.1. Some recognition and localization algorithms that have already been implemented in our robots are also presented in this section. In Sec. 6.2, DP and the VQ method are applied to a task in which an ERS-210

approaches to a ball from suitable direction for shoot. A VQ map is used in an ERS-210 for experiment. In this section, we use the entropy function and the vector quantization method for VQ maps, which are presented in Chapter 4.

In Sec. 6.3-6.6, another task, which has the largest state-space in the tasks in this thesis, is handled. The task is to make two ERS-210s score within minimum time. We define eight dimensional state-space and 610 million discrete states and apply our method to this task. In Sec. 6.3, the task is defined. A huge state-action map that contains 610 million discrete states is created in Sec. 6.4. The huge map is compressed in Sec. 6.5, and obtained VQ maps are evaluated in Sec. 6.6.

We discuss the simulation and experimental results of the two tasks in Sec. 6.7, and conclude this chapter.

6.1 RoboCup Four Legged Robot League

The RoboCup four legged robot league has been held since 1998 [Asada, 1999]. The photo in Fig. 6.1 is a scene of a game in RoboCup 2002. Every year, rule of this league has been modified in accordance with technological improvement. Since simulation and experiment in this section are held on the environment in 2002 and 2003, we explain the robot and the environment of this league based on the rules in the years [Fujita, 2003].

In a game, four sets of ERS-210, which are quadruped robots, make up a team. One of them is the goalkeeper and the others are called field players. They can communicate with their teammates through wireless LAN, while they do not controlled by any external computer. It means that they should behave autonomously. Each ERS-210 must recognize the state of a game, which varies from second to second, and must choose its action tactfully. In that condition, state-action maps are very useful for quick decision making.

The length of a game is 20 minutes. Detailed rules are fixed up by a rule book as well as human soccer. Moreover, there is a characteristic rule that any hardware of ERS-210 must not be altered.

Fig. 6.1: A Game Scene of RoboCup 2002 Fukuoka

6.1.1 Autonomous Robot ERS-210

In this chapter, an autonomous quadruped robot, ERS-210, made by SONY is used for experiments. Its appearance is shown in Fig.6.2. Its height, width and breadth are 200[mm], 150[mm], and 250[mm] at a standard pose. This robot is fully autonomous. A computer that is composed of a 192MHz MIPS CPU and 32MB of RAM is equipped in the body. On its nose, there is a color CMOS camera whose resolution is 176 by 144. This robot has three DoF in each leg and in its head as shown in Fig.6.2. Its executable code is compiled and linked another computer. The executable code is installed on an ERS-210 through a 16MB flash RAM. Therefore, the size of the code and other data on the flash RAM must be smaller than 16MB.

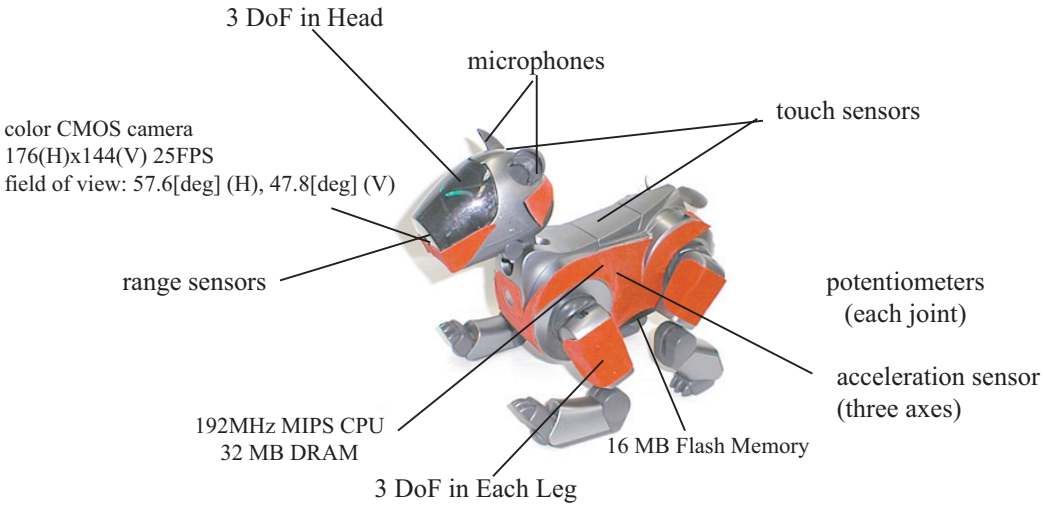


Fig. 6.2: ERS-210 Made by SONY

6.1.2 Soccer Field and Accompanying Items

Figure 6.3 illustrates a soccer field for this league. The size of the field except two inside areas of goals is 4.2[m] by 2.7[m]. This area is covered with a green carpet. The field is framed in by sloping walls that prevent the ball from going out except two goalmouths. Moreover, four corners of the field are covered by isosceles triangle walls. The length of a hidden edge of the green carpet is 300[mm].

Each goalmouth is 600[mm] across and the depth of the goal is 300[mm]. Each goal area is also covered with the green carpet and is surrounded by a

vertical wall on three sides. One of the wall, which is called a goal board in this thesis, is painted in sky-blue. Another goal board is painted in yellow. A goal board in showing in Fig. 6.4.

There are six landmarks around the field. They are used for self-localization of robots. The shape of every landmark is a cylinder whose height and outside diameter are 400[mm] and 103[mm] respectively. As shown in Fig. 6.4, the upper half of a landmark is colored by two colors. The six landmarks have its own pattern of the colors so as to be distinct from each other.

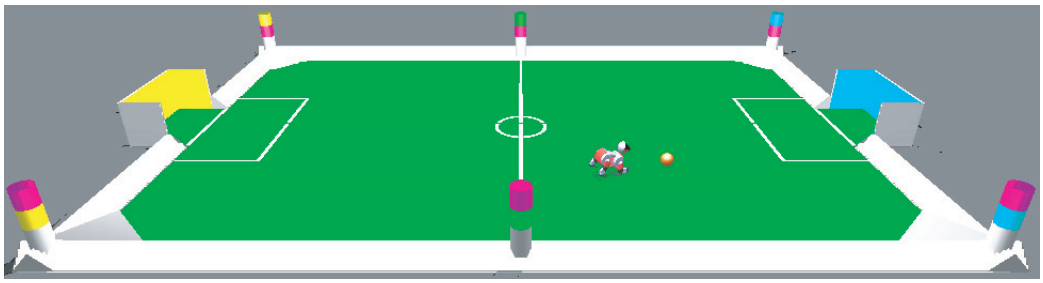


Fig. 6.3: Field for RoboCup 2003

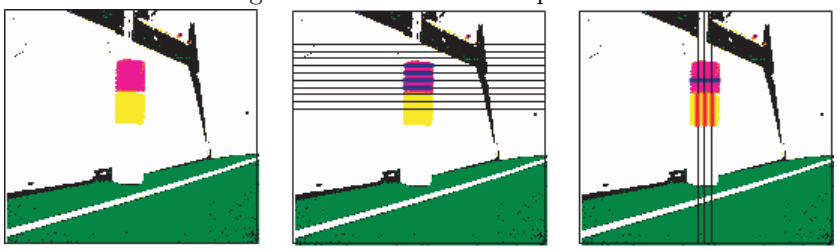


Fig. 6.4: Goal and Landmark

We define a field coordinate system Σ_{field} and a robot coordinate system Σ_{robot} as shown in Fig. 6.5. The origin of Σ_{field} is the center point of the field. Its x-axis is parallel to the touch lines and faces to the sky-blue goal. Its y-axis is on the halfway line.

The robot coordinate system Σ_{robot} exists in the same plane with Σ_{field} though the robot has a three-dimensional body. Its x-axis exists on a line that runs through two points that are located directly below shoulders of the robot. The y-axis of Σ_{field} is perpendicular to the x-axis. Their direction is

as shown in the figure. The origin of Σ_{robot} exists under the midpoint of the two points.

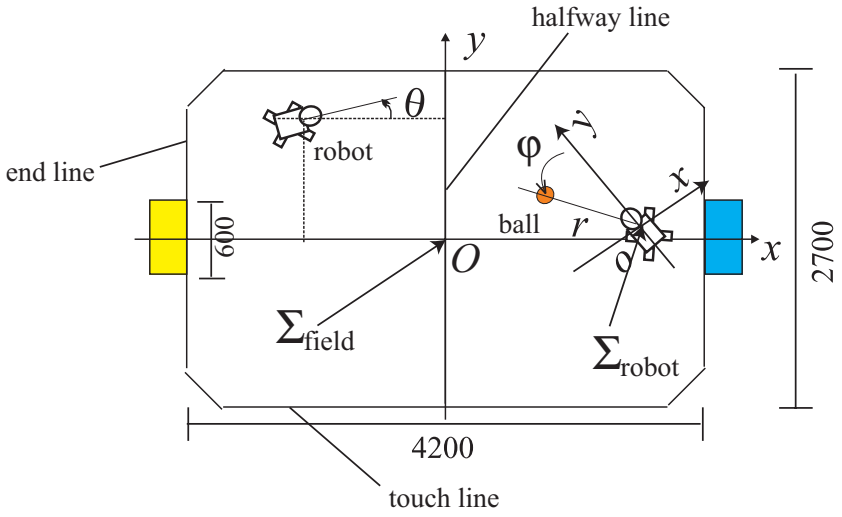


Fig. 6.5: Coordinate Systems

6.1.3 Recognition

Color Extraction

An ERS-210 obtains required information for soccer from its CMOS camera mainly. To detect the colored objects on CMOS camera images easily, we utilize a look-up table that is called *a color table*. A color table is a three-dimensional array, which corresponds to the YCbCr space. As shown in Fig. 6.6, the types of color are plotted on a color table. Images from the CMOS camera are changed into the simplified images such as the upper right image of Fig. 6.6 by a color table. The ball, a landmark, two robots, the field, and walls can be detected from the color extracted image easily.

To calculate the position of an object in a color extracted image on Σ_{robot}, the followings should be known:

(1) the pose of the CMOS camera toward the robot coordinate system Σ_{robot}

(2) one of the followings

  (a) the size and shape of the object

(b) the height of the object from the ground

One of (2a) or (2b) is required for measurement because images are two-dimensional. When (2a) and (2b) are unknown, only the direction of the object can be measured. In our algorithm, (1) is calculated from joint angles of the head and pose of the body.

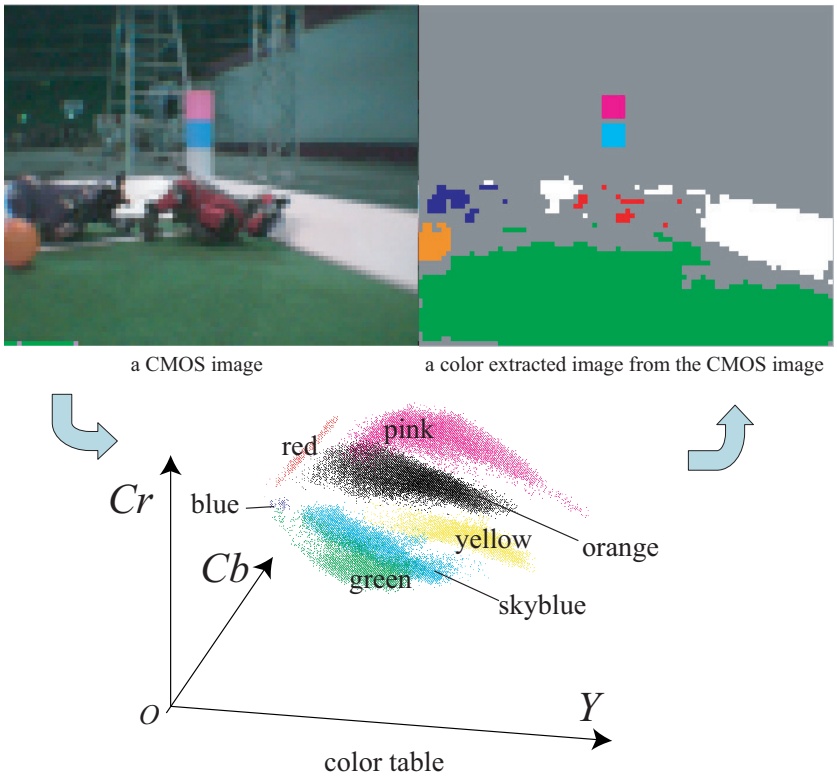


Fig. 6.6: Color Extraction

Self-Localization

Color extracted images of landmarks and goals are utilized for self-localization of the robot. The position of the robot, which is the origin of Σ_{robot}, is represented by (x, y) on Σ_{field} and its orientation, θ, is defined as the direction of y-axis of Σ_{robot} toward x-axis of Σ_{field}. An estimated position and orientation of the robot, which is called the pose of the robot simply hereafter, is continuously computed by a probabilistic self-localization algorithm. This algorithm is called the uniform Monte Carlo localization (uniform MCL) [Ueda, 2002]. This localization method gives an estimated

pose of the robot: $(\hat{x}, \hat{y}, \hat{\theta})$.

The Uniform MCL method uses a lot of candidates ξ_i $(i = 1, 2, \ldots, N)$ to approximate the probability distribution of where the actual pose of the robot exists in the $xy\theta$-space. The candidates are called particles. The particles are distributed in the $xy\theta$-space. An example of distribution of particles is illustrated in Fig.6.7. The large arrow denotes the actual pose of a robot. The small arrows denote the poses of particles. This distribution is obtained after an observation of the upper-right landmark in the figure. The robot knows the direction of the landmark and the maximum distance from an image processing algorithm, which measures the orientation and width of a landmark or a goal in a color extracted image. Particles that are inconsistent with this information are erased. The other samples remain and this distribution is obtained. When the robot walks, the particles also move with the displacement of the pose. At this phase, erased particles are restored between the remaining particles. When the robot can obtain another observation, inconsistent particles are erased again. By the iteration of these processes, the distribution of particles shrinks around the actual pose.

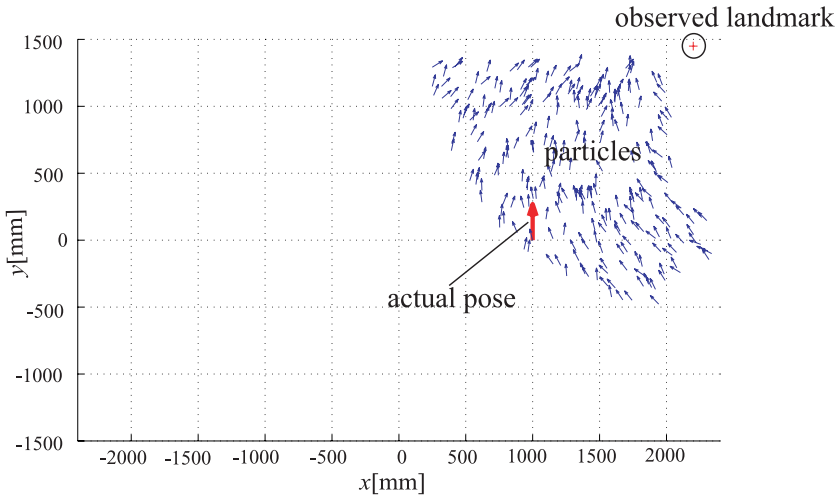


Fig. 6.7: Distribution of Particles in Uniform Monte Carlo Localization

When an ERS-210 localizes itself without walking by this algorithm, the error of position is 233[mm] on average and that of orientation is 6.5[deg] on average [Asanuma, 2004]. The error of position denotes $e = \sqrt{(\hat{x} - x)^2 + (\hat{y} - y)^2}$ when (x, y) is the actual position. Note that the robot moves with this extent of localization error on the experiments in this

chapter.

Incidentally, the accuracy of self-localization in the four legged robot league are improving year by year. Our current localization algorithm is explained in Appendix B. With painstaking calibration, its error becomes several centimeters and a few degrees on average in the environment of RoboCup 2005 (a 6[m] by 4[m] field with four landmarks).

Measurement of Ball

The algorithm for measurement of the ball calculates the position of the ball from a color extracted image. When the ball is small, a width of the orange blob on the image is used for measuring its position. When the number of the pixels is large or when the CMOS camera is looking down at the field, the position is calculated from the position of the center of the orange blob.

The accuracy of the measurement algorithm depends a lot on the accuracy of color extraction. When the robot walks fast, the error becomes large due to jolt of its body. Roughly speaking, we must expect 30% error when the algorithm returns the position of the ball.

6.2 Task of Going to Ball

At first, we apply DP and VQ to one of the most fundamental tasks in robot soccer. This task is to approach the ball from a proper direction with the minimum number of steps (actions). In this task, the robot should consider the position of the ball from itself and its direction from itself on the field. Moreover, the robot should consider its position on the field because the walls around the field prevent the robot from going to the ball throughout the field. The robot must then be able to observe the ball at any time in its task. In other words, the ball must always be in front of the robot.

6.2.1 State Space

The total state variables are the following five: $(x, y, \theta, r, \varphi) = \boldsymbol{x}$. The domain and the way of discretization of each state variable are defined as shown in Table 6.1. The state space, $\mathcal{X}$, can be defined as the direct product of the domains in Table 6.1(a). r is divided unevenly into nine divisions by the equation in the table (b). $[r]_0$, $[r]_1$, and $[r]_2$ have 100[mm] width respectively. The widths of others increase as $200, 300, \ldots, 600$, and 700[mm]. By the discretization shown in Table 6.1(b), the number of discrete states N reaches $765,450$.

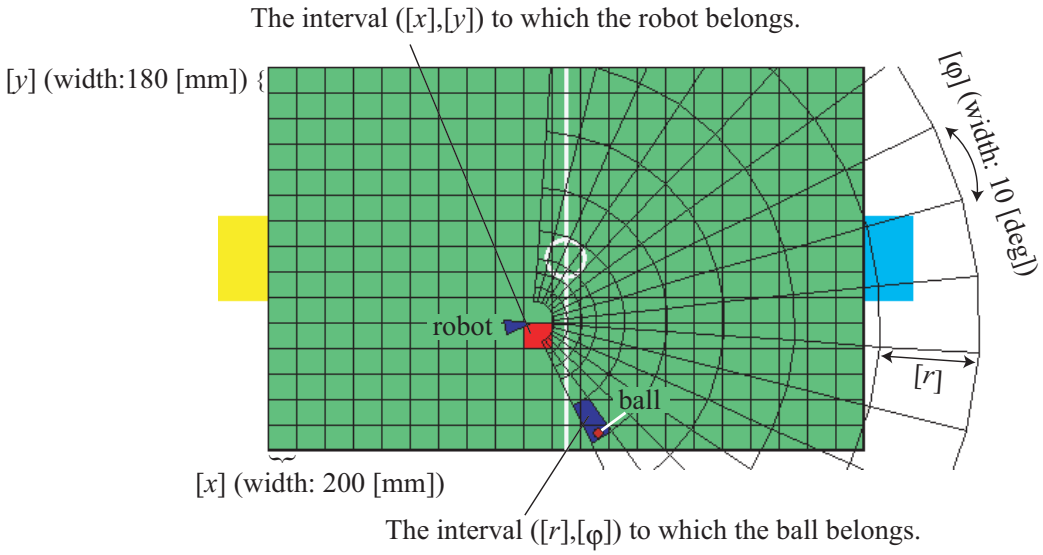


Fig. 6.8: Discretization for Task of Going to Ball

Table 6.1: Domain and Discretization of State Space

(a) domain

	definition of intervals
x	$[-2100, 2100)$[mm]
y	$[-1350, 1350)$[mm]
θ	$[-180, 180)$[deg]
r	$[150, 3150)$[mm]
φ	$[-75, 75)$[deg]

(b) discretization

	definition of intervals	
x	$[x]_i \equiv \big[200i - 2100, 200(i+1) - 2100\big)$[mm]	$(i = 0, 1, \ldots, 20)$
y	$[y]_i \equiv \big[180i - 1350, 180(i+1) - 1350\big)$[mm]	$(i = 0, 1, \ldots, 14)$
θ	$[\theta]_i \equiv \big[20i - 180, 20(i+1) - 180\big)$[deg]	$(i = 0, 1, \ldots, 17)$
r	$[r]_i \equiv \begin{cases} \big[100i + 150, 100(i+1) + 150\big)\text{[mm]} & (i = 0, 1, \ldots, 2) \\ \left[50\left(i - \frac{3}{2}\right)^2 + \frac{675}{2}, 50\left(i - \frac{1}{2}\right)^2 + \frac{675}{2}\right)\text{[mm]} & (i = 3, 4, \ldots, 8) \end{cases}$	
φ	$[\varphi]_i \equiv \big[10i - 75, 10(i+1) - 75\big)$[deg]	$(i = 0, 1, \ldots, 14)$

(c) transform from value to index

	definition of intervals	
x	$i_x = \lfloor (x + 2100)/200 \rfloor$	$(-2100 \le x < 2100)$[mm]
y	$i_y = \lfloor (y + 1350)/200 \rfloor$	$(-1350 \le y < 1350)$[mm]
θ	$i_\theta = \lfloor (\theta + 180)/20 \rfloor$	$(-180 \le \theta < 180)$[deg]
r	$i_r = \begin{cases} \lfloor (r - 150)/100 \rfloor & (150 \le r < 450)\text{[mm]} \\ \lfloor 3/2 + \sqrt{2r - 675}/10 \rfloor & (450 \le r < 3150)\text{[mm]} \end{cases}$	
φ	$i_\varphi = \lfloor (\varphi + 75)/10 \rfloor$	$(-75 \le \varphi < 75)$[deg]

Table 6.1(c) shows how to solve the indexes that are defined in Table (b) from the values in the continuous state space $\mathcal{X}$. When a discrete state $s_i \in \mathcal{S}$ is represented by $([x]_{i_x}, [y]_{i_y}, [\theta]_{i_\theta}, [r]_{i_r}, [\varphi]_{i_\varphi})$, we define relation between the state index i and the indexes of the axes as

$$\begin{aligned} i &= 15 \cdot 18 \cdot 9 \cdot 15 i_x + 18 \cdot 9 \cdot 15 i_y + 9 \cdot 15 i_\theta + 15 i_r + i_\varphi \\ &= 36450 i_x + 2430 i_y + 135 i_\theta + 15 i_r + i_\varphi. \end{aligned} \tag{6.1}$$

When we use the equations in Table 6.1(c), the transform from a continuous state to a state index can be defined as

$$i = {}^{\mathcal{I}_s}\mathcal{T}_{\mathcal{X}}(\boldsymbol{x}) = 36450 i_x + 2430 i_y + 135 i_\theta + 15 i_r + i_\varphi \tag{6.2}$$

$$\begin{aligned} &= 36450 \lfloor (x + 2100)/200 \rfloor + 2430 \lfloor (y + 1350)/200 \rfloor \\ &+ 135 \lfloor (\theta + 180)/20 \rfloor + 15 i_r + \lfloor (\varphi + 75)/10 \rfloor, \end{aligned} \tag{6.3}$$

where

$$i_r = \begin{cases} \lfloor (r - 150)/100 \rfloor & (150 \le r < 450) \\ \lfloor 3/2 + \sqrt{2r - 675}/10 \rfloor & (450 \le r < 3150) \end{cases}.$$

Final State

The final states are then defined. At first, we define a following set $\mathcal{S}_{\text{ball\_f}}$ of discrete states:

$$\mathcal{S}_{\text{ball\_f}} = \left\{ s = ([x], [y], [\theta], [r], [\varphi]) \,\middle|\, [r] = [r]_0, [\varphi] = [\varphi]_i \ (4 \leq i \leq 10) \right\}. \quad (6.4)$$

$[x], [y]$ and $[\theta]$ are arbitrary intervals. We consider that the robot has reached to the ball when $s \in \mathcal{S}_{\text{ball\_f}}$. The robot must face to the sky-blue goal when it has reached to the ball. We define $\mathcal{S}_{\text{robot\_f\_1}}$ as the set of discrete states where the robot faces to the goal. It is defined as

$$\begin{aligned} \mathcal{S}_{\text{robot\_f\_1}} = &\left\{ s \,\middle|\, [\theta] = [\theta]_i \ (7 \leq i \leq 10) \right\} \\ &\cup \left\{ s \,\middle|\, -40[\text{deg}] \leq \varphi_{\text{G}}(\boldsymbol{x}) < 40[\text{deg}] \ (\forall \boldsymbol{x} \in s) \right\} \end{aligned} \quad (6.5)$$

where $\varphi_{\text{G}}(\boldsymbol{x})[\text{deg}]$ means the direction of the goal from the robot, where the point $(x, y) = (2100, 0)[\text{mm}]$ on Σ_{field} is regarded as the position of the goal.

The set of final states can be defined as $\mathcal{S}_{\text{ball\_f}} \cap \mathcal{S}_{\text{robot\_f\_1}}$. However, this condition should be relaxed near the wall since it prevents the robot going to one of the final states. We therefore prepare another set:

$$\begin{aligned} \mathcal{S}_{\text{robot\_f\_2}} = &\left\{ s \,\middle|\, [y] = [y]_0, [\theta] = [\theta]_i \ (6 \leq i \leq 9) \right\} \\ &\cup \left\{ s \,\middle|\, [y] = [y]_{14}, [\theta] = [\theta]_i \ (8 \leq i \leq 11) \right\} \\ &\cup \left\{ s \,\middle|\, [x] = [x]_0, [\theta] = [\theta]_i \ (3 \leq i \leq 14) \right\}. \end{aligned} \quad (6.6)$$

The set of final states is then defined as

$$\mathcal{S}_{\text{f}} = \mathcal{S}_{\text{ball\_f}} \cap (\mathcal{S}_{\text{robot\_f\_1}} \cup \mathcal{S}_{\text{robot\_f\_2}}). \quad (6.7)$$

Under that definition, the number of final states in $\mathcal{S}$ becomes $8,175$. It means that 1.07% of discrete states are final states.

Action and State Transition

38 kinds of actions, shown in Table 6.2, are used for this task. As shown in Fig. 6.9, each action is parameterized with $(\delta_x, \delta_y, \delta_\theta)$. These parameters denote the displacement of the robot with an action from Σ_{robot}, which is the robot coordinate system before the action. Though displacements with an action vary each time, the dispersion is ignorable when compared with

Table 6.2: Actions for Task of Going to Ball

symbol	name	δ_x[mm]	δ_y[mm]	δ_θ[deg]
a_0	Forward	0.0	113.0	0.0
a_1	TurnRight	20.0	5.0	-32.5
a_2	TurnLeft	-15.0	5.0	28.0
a_3	RightForward	77.0	75.0	0.0
a_4	LeftForward	-79.0	72.5	0.0
a_5	RightSide	114.0	0.0	5.0
a_6	LeftSide	-107.0	0.0	-5.0
a_7	RollRight	30.0	35.0	17.0
a_8	RollLeft	-30.0	35.0	-30.0
a_9	ShortForward	0.0	66.0	0.0
a_{10}	ShortRightForward	58.0	55.0	0.0
a_{11}	ShortLeftForward	-65.0	56.0	0.0
a_{12}	ShortRightSide	80.0	0.0	0.0
a_{13}	ShortLeftSide	-66.0	0.0	0.0
a_{14}	ShortRollRight	25.0	30.0	18.0
a_{15}	ShortRollLeft	-25.0	30.0	-30.0
a_{16}	RightBackward	84.0	-83.0	-2.8
a_{17}	LeftBackward	-73.0	-69.0	2.0
a_{18}	RightForward15	16.0	84.0	2.0
a_{19}	LeftForward15	-14.0	102.0	-2.6
a_{20}	RightForward30	36.0	74.0	0.0
a_{21}	LeftForward30	-32.0	102.0	-4.4
a_{22}	RightForward60	102.0	48.0	0.0
a_{23}	LeftForward60	-102.0	46.0	0.0
a_{24}	RightForward75	116.0	34.0	0.0
a_{25}	LeftForward75	-121.0	26.0	0.0
a_{26}	RightBackward15	36.0	-94.0	0.0
a_{27}	LeftBackward15	-35.0	-95.0	0.0
a_{28}	RightBackward30	56.0	-86.0	0.0
a_{29}	LeftBackward30	-55.0	-84.0	0.0
a_{30}	RightBackward60	110.0	-40.0	0.0
a_{31}	LeftBackward60	-106.0	-42.0	-4.0
a_{32}	RightBackward75	115.0	-26.0	0.0
a_{33}	LeftBackward75	-95.0	-23.0	-5.0
a_{34}	RightForwardTurnLeft	30.0	50.0	17.0
a_{35}	LeftForwardTurnRight	-25.0	70.0	-15.0
a_{36}	RightForwardTurnRight	92.0	80.0	-15.0
a_{37}	LeftForwardTurnLeft	-100.0	80.0	15.0

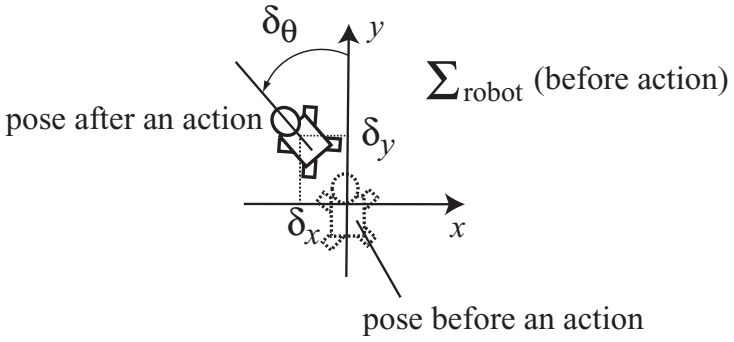


Fig. 6.9: Definition of Parameters for Actions

the granularity of discretization mentioned later.

We then define state transitions, $\mathcal{P}^a_{ss'}$. When the robot chooses an action that cause $(\delta_x, \delta_y, \delta_\theta)$ of displacement from $\boldsymbol{x} = (x, y, \theta, r, \varphi)$, the posterior states $\boldsymbol{x}' = (x', y', \theta', r', \varphi')$ fulfills

$$\begin{pmatrix} x' \\ y' \\ \theta' \\ r' \\ \varphi' \end{pmatrix} = \begin{pmatrix} x + \delta_x \cos\theta - \delta_y \sin\theta \\ y + \delta_x \sin\theta + \delta_y \cos\theta \\ \theta + \delta_\theta \\ \sqrt{(r\cos\varphi - \delta_x)^2 + (r\sin\varphi - \delta_y)^2} \\ \arctan[(r\sin\varphi - \delta_y)/(r\cos\varphi - \delta_x)] - \delta_\theta \end{pmatrix} \tag{6.8}$$

As in the case of the puddle world task, the collision between the robot and the wall is not considered in this phase.

In the discrete state space $\mathcal{S}$, this transition is represented statistically. State transition probabilities $\mathcal{P}^a_{ss'}$ can be computed from iterations of the following procedure:

- choose $\boldsymbol{x} \in \mathcal{X}$ from $s \in \mathcal{S}$
- calculate $\boldsymbol{x}'$ from the chosen $\boldsymbol{x}$ based on Eq. (6.8)
- record s', to which $\boldsymbol{x}'$ belongs,

which is a kind of Monte Carlo integration. If we compute the choice probability of every s', we can obtain the probabilities of all state transitions toward a set of s and a.

However, we should pay attention to reduce the computational complexity for this computation; otherwise $(765,450-8,175)\cdot 38\cdot 765,450 = 2.20\cdot 10^{13}$ state transitions must be computed.

We cut off state transitions whose probabilities are less than 0.01. This pruning is useful for reducing the number of state transitions, while the completeness of the state-action map is lost. Possible posterior states s' toward a set of s and a are limited to 100 with this pruning. Therefore, the number of combinations of state transitions are reduced to $(765,450-8,175)\cdot 100\cdot 38 = 2.88\cdot 10^9$. Though this number is much smaller than 10^{13}, gigabytes of memory is required for recording probabilities of all state transitions.

The computational complexity can be reduced more if the state transitions of the pose of the robot and those of the ball are recorded separately. Equation (6.8) can be separated into the two equations:

$$\begin{pmatrix} x'-x \\ y'-y \\ \theta' \end{pmatrix} = \begin{pmatrix} \delta_x\cos\theta - \delta_y\sin\theta \\ \delta_x\sin\theta + \delta_y\cos\theta \\ \theta+\delta_\theta \end{pmatrix}, \text{ and} \tag{6.9}$$

$$\begin{pmatrix} r' \\ \varphi' \end{pmatrix} = \begin{pmatrix} \sqrt{(r\cos\varphi-\delta_x)^2+(r\sin\varphi-\delta_y)^2} \\ \arctan[(r\sin\varphi-\delta_y)/(r\cos\varphi-\delta_x)]-\delta_\theta \end{pmatrix}. \tag{6.10}$$

From these equations, the following probabilities

$$P_{\text{robot}}\left([x]_{i-\Delta i},[y]_{j-\Delta j},[\theta]_{k'}\middle|[x]_i,[y]_j,[\theta]_k,a\right), \text{ and } P_{\text{ball}}\left([r]_{h'},[\varphi]_{\ell'}\middle|[r]_h,[\varphi]_\ell,a\right)$$

can be computed toward action a, prior state $s = ([x]_i,[y]_j,[\theta]_k,[r]_h,[\varphi]_\ell)$, and posterior state $s' = ([x]_{i-\Delta i},[y]_{j-\Delta j},[\theta]_{k'},[r]_{h'},[\varphi]_{\ell'})$. $\mathcal{P}^a_{ss'}$ is then calculated as

$$\begin{aligned}\mathcal{P}^a_{ss'} &= P_{\text{robot}}(s'|s,a)P_{\text{ball}}(s'|s,a)\\ &= P_{\text{robot}}\left([x]_{i-\Delta i},[y]_{j-\Delta j},[\theta]_{k'}\middle|[x]_i,[y]_j,[\theta]_k,a\right)\\ &\quad\cdot P_{\text{ball}}\left([r]_{h'},[\varphi]_{\ell'}\middle|[r]_h,[\varphi]_\ell,a\right).\end{aligned} \tag{6.11}$$

The expression of P_{robot} implies that the transition rule on x-axis and y-axis depends not on index i and index j if we do not consider the collision of the robot and the wall. We can obtain such a table as Table 6.3(a), which records every probability P_{robot} toward every combination of $a, k, \Delta i, \Delta j$, and k', by the Monte Carlo method. This table has $3,741$ lines. In the case

of The expression of P_{ball}, a transition is represented by the following values: $(h, \ell, h', \ell', a, P)$. We can also obtain Table 6.3(b), which contains $20,013$ probabilities toward all combinations of a, h, ℓ, h', and ℓ'.

Table 6.3: Tables of State Transitions

(a) Pose of Robot

action	k	Δi	Δj	k'	P_{robot} [%]
a_0	0	-1	-1	0	6
a_0	0	-1	0	0	52
a_0	0	0	-1	0	4
a_0	0	0	0	0	38
a_0	1	-1	-1	1	15
a_0	1	-1	0	1	35
			⋮		
a_{37}	16	0	0	16	7
a_{37}	16	0	0	17	21
a_{37}	17	-1	-1	0	17
a_{37}	17	-1	-1	17	5
a_{37}	17	-1	0	0	18
a_{37}	17	-1	0	17	8
a_{37}	17	0	-1	0	23
a_{37}	17	0	-1	17	4
a_{37}	17	0	0	0	19
a_{37}	17	0	0	17	6

(b) Position of Ball

action	h	ℓ	h'	ℓ'	P_{ball} [%]
a_0	0	0	out		≥ 1
a_0	0	1	out		≥ 1
a_0	0	2	out		≥ 1
a_0	0	3	out		≥ 1
			⋮		
a_{12}	2	7	2	8	88
a_{12}	2	7	2	9	7
a_{12}	2	7	3	8	5
			⋮		
a_{37}	8	13	8	11	41
a_{37}	8	13	8	12	40
a_{37}	8	14	7	12	9
a_{37}	8	14	7	13	8
a_{37}	8	14	8	12	40
a_{37}	8	14	8	13	43

Reward

The immediate evaluation $\mathcal{R}^a_{ss'}$ can be defined from the task. The goal is to reduce the steps of the robot in the task. A state of the robot and the ball should always be in the state space. Therefore, $\mathcal{R}^a_{ss'}$ is defined as:

$$\mathcal{R}^a_{ss'} = \begin{cases} -\infty & (\text{if } s' \notin \mathcal{S}) \\ -1 \text{ [step]} & (\text{if } s' \in \mathcal{S}). \end{cases} \tag{6.12}$$

Computation and Its Result

We apply the value iteration algorithm in Fig. 2.5 to creating a state-action map. We use a fixed-point variable for recording values. Its minimum step is 0.01[step]. Sweeps are iterated until $\Delta > 0$. Δ, which is shown in Fig. 2.5, is the maximum variation of values in a sweep. A computer with a 3.6 GHz Pentium IV CPU is used for value iteration.

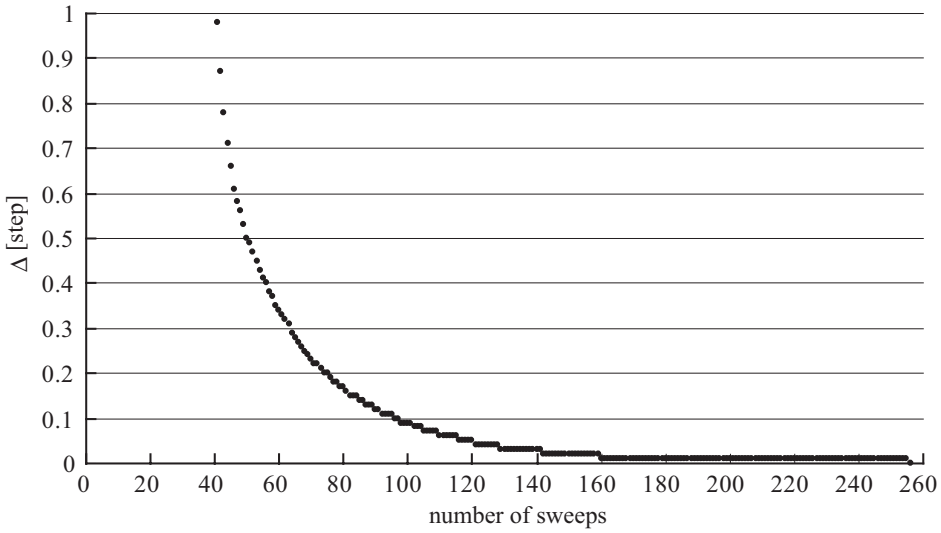


Fig. 6.10: Convergence of State-Value Function

Table 6.4: Computation Time (with a 3.6GHz Pentium IV CPU)

procedure	time
Creation of Table 6.3(a) (10^5 samples)	13.6[s]
Creation of Table 6.3(b) (10^4 samples)	16.0[s]
value iteration (one iteration)	5.9[s]
total time (253 sweeps)	1.3×10^3[s]

Δ and time for computation are recorded at every sweep in the computation.

Figure 6.10 shows the relation between the number of sweeps and Δ. As shown in this figure, this value reduces to zero asymptotically in the iteration of sweeps. Δ becomes less than one at the end of 41st sweep. Though Δ reaches 0.01[step] at the 160th sweep, 256 sweeps are required for making it zero. Computation time of each process is shown in Table 6.4. Though there are $7{\cdot}10^5$ states in $\mathcal{S}$, a state-action map can be created with 25 minutes.

6.2.2 Evaluation of Obtained State-Action Map

Figure 6.11 shows part of the state-value function when the ball is at point $(x, y) = (0, -1200)$ and the robot observes it toward the front ($\varphi = 0$). The state-action map of this part is shown in Fig. 6.12. 23 kinds of action

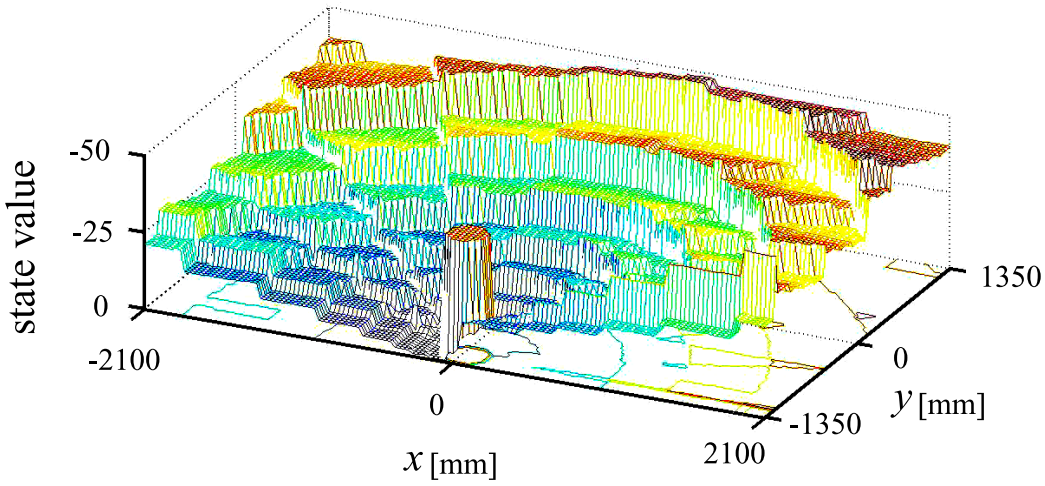


Fig. 6.11: Part of State-Value Function

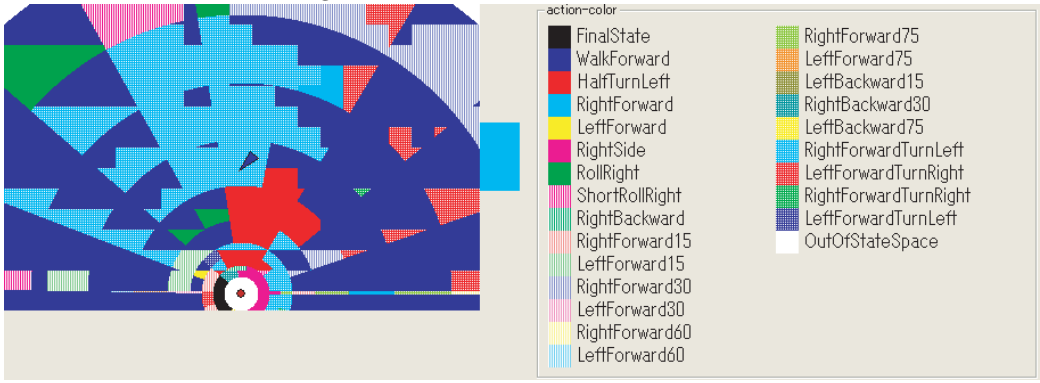


Fig. 6.12: Part of state-action map

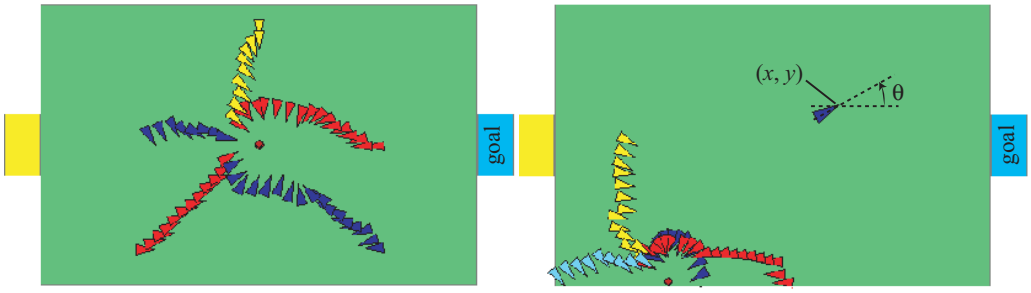


Fig. 6.13: Simulated Trajectories Based on The State-Action Map

are intricately allocated to each state. It is especially complicated within a radius of 450[mm] from the ball since the robot must approach the final states while avoiding the off-limits area around the ball. In Fig.6.13, some simulated trajectories are illustrated. We see from this figure that the robot adjusts θ to offensive direction by little and little en route to the ball. It is difficult for us to program such detailed decision making by hand-coding. It seems that the complexity of the state-action map represents the difficulty.

Since we do not consider state transitions that occur with smaller percentages than 1[%], there is no assurance that the robot can reach a final state. In other words, the state will be out of the state space on some occasions. Therefore, we should exceptionally add the success rate of the task to evaluate this map. This rate of this map is 99.9[%], which is measured by $5,000$ trials of simulation from $5,000$ different initial states.

6.2.3 Compression of The State-Action Map

The state-action map is compressed by the VQ method. Here, we also investigate the relation of the entropy defined by Eq. (4.4) and the loss of the optimality. The map obtained in Sec. 6.2 is compressed by various pairs of (ν, ε). Each VQ map is then evaluated its entropy $\mathcal{H}$ and efficiency J.

Entropy on Each Axis

At first, the entropy $\mathcal{H}$ is computed for five ways of blocking. In each way of blocking, the state-action map is sliced by perpendicular planes to one of the five axes. Table 6.5 shows the entropy toward each way of blocking. w_x denotes the number of intervals on x-axis in a vector. w_x, w_y, w_θ, w_r, and w_φ also denote that on each axis respectively.

Table 6.5: Entropy Evaluation for Blocking (small: good)

direction of slices	w_x	w_y	w_θ	w_r	w_φ	$\mathcal{H}$
perpendicular to x-axis	1	15	18	9	15	0.75
perpendicular to y-axis	21	1	18	9	15	0.93
perpendicular to θ-axis	21	15	1	9	15	1.50
perpendicular to r-axis	21	15	18	1	15	1.82
perpendicular to φ-axis	21	15	18	9	1	1.82

As shown in the table, we can expect effective compression with small w_x

and w_y. For actual compression, however, we must divide the state-action map into smaller vectors.

Partitioning

Here we define some blocking ways that are actually used in compression. VQ maps with various compression ratios are built toward each way of blocking. When we create many VQ maps from a state-action map with different compression ratios, the PNN algorithm is more suitable than the Lloyd algorithm. That is because the number of clusters decreases one by one in the algorithm, while the number of clusters is fixed in the Lloyd algorithm. However, the computation time becomes huge to built all VQ maps that are evaluated in this section.

To reducing the computing time, we apply the partitioning technique in Sec. 4.3.4. The uncompressed map is equally divided into three partitions perpendicular to θ-axis. The reason to choose θ-axis is that the number of intervals, 18, has the most number of divisors in the numbers of intervals of the axes.

Toward $s_i = ([x]_{i_x}, [y]_{i_y}, [\theta]_{i_\theta}, [r]_{i_r}, [\varphi]_{i_\varphi})$, indexes for sub maps are defined as

$$\begin{aligned}\begin{pmatrix} i_\kappa \\ i_\iota \end{pmatrix} = \begin{pmatrix} \kappa(i) \\ \iota(i) \end{pmatrix} &= \begin{pmatrix} \lfloor i_\theta/6 \rfloor \\ 15 \cdot 6 \cdot 9 \cdot 15 i_x + 6 \cdot 9 \cdot 15 i_y + 9 \cdot 15 (i_\theta \% 6) + 15 i_r + i_\varphi \end{pmatrix} \\ &= \begin{pmatrix} \lfloor i_\theta/6 \rfloor \\ 12,150 i_x + 810 i_y + 135 (i_\theta \% 6) + 15 i_r + i_\varphi \end{pmatrix} \end{aligned} \tag{6.13}$$

from Eq. (6.1). From this equation, action $\pi(s_i)$ belongs to i_κth sub map and is given index i_ι in the sub map.

Definition of Blocking

Under the partitioning, the map is compressed with all combinations of the following numbers: $\{w_x = 1, 3, 21\}, \{w_y = 1, 3, 18\}, \{w_\theta = 1, 3, 6\}, \{w_r = 1, 3, 9\}, \{w_\varphi = 1, 3, 15\}$. In every sub map, the definition of (ν, ε) is defined

as

$$\begin{pmatrix} i_\nu \\ i_\varepsilon \end{pmatrix} = \begin{pmatrix} \nu(i_\iota) \\ \varepsilon(i_\iota) \end{pmatrix} = \begin{pmatrix} \dfrac{12,150}{w_{y\theta r\varphi}} \left\lfloor \dfrac{i_x}{w_x} \right\rfloor + \dfrac{810}{w_{\theta r\varphi}} \left\lfloor \dfrac{i_y}{w_y} \right\rfloor + \dfrac{135}{w_{r\varphi}} \left\lfloor \dfrac{i_\theta \% 6}{w_\theta} \right\rfloor + \dfrac{15}{w_\varphi} \left\lfloor \dfrac{i_r}{w_r} \right\rfloor + \left\lfloor \dfrac{i_\varphi}{w_\varphi} \right\rfloor \\ w_{y\theta r\varphi}(i_x \% w_x) + w_{\theta r\varphi}(i_y \% w_y) + w_{r\varphi}\{(i_\theta \% 6) \% w_\theta\} + w_\varphi(i_r \% w_r) + i_\varphi \% w_\varphi \end{pmatrix}, \tag{6.14}$$

where $w_{y\theta r\varphi} = w_y w_\theta w_r w_\varphi$, $w_{\theta r\varphi} = w_\theta w_r w_\varphi$, and $w_{r\varphi} = w_r w_\varphi$. Incidentally, the following equations:

$$N_p = 3, N_\varepsilon = w_x w_y w_\theta w_r w_\varphi, \text{ and } N_\nu = \frac{N}{N_p N_\varepsilon}$$

are fulfilled in each partition.

Though the above definition can be differentiated for each sub map, the identical definition is applied to each of them. That is because we do not want to increase the number of VQ maps that are evaluated excessively. Combinations with $N_\varepsilon < 100$ are then eliminated because we cannot expect a high compression ratio as well as calculation amount is large. Moreover, combinations with $N_\varepsilon > 5,000$ are also eliminated since the compression ratio cannot be adjusted to a target. In the above condition, the uncompressed map is compressed with 135 combinations of blocking. Toward the 135 combinations, we have built VQ maps with the following number of representative vectors: $N_c = 1, 2, \ldots, 9, 10, 20, \ldots, 90, 100, 200, \ldots, 900$.

6.2.4 Evaluation of The Entropy Function

The following simulation is repeated 5,000 times with every VQ map and the uncompressed map.

Step 1: choose an initial state at random

Step 2: count the number of steps from the initial state to a final state with the uncompressed map

Step 3: count the number with a VQ map from the same initial state.

If the steps exceed 100 or the state in a trial is out of the state space, the situations are regarded as failures.

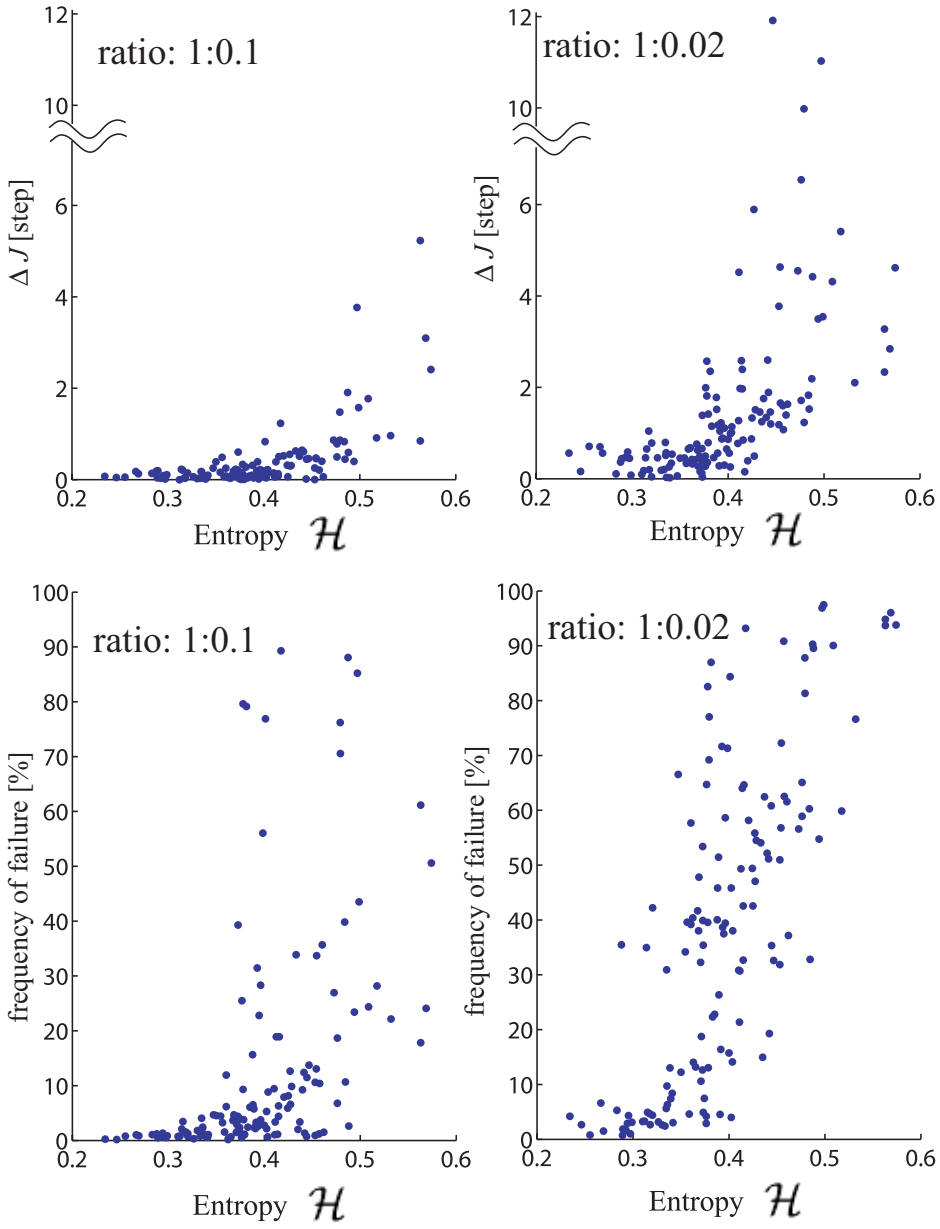


Fig. 6.14: The Relation between Entropy and Efficiency Loss

Figure 6.14 shows the results with $L_{\mathrm{VQ}} : L_{\mathrm{uncomp}} = 1 : 0.1$ and $1 : 0.02$. L_{VQ} is the total sizes of three partial maps in this case. The upper figures illustrate the relation between entropy and the increase of steps. The increase of steps is the difference of the average numbers of steps between the uncompressed map and a compressed map. This value is calculated from trials in which both the Step 2 and 3 are successful. The average number of steps with the uncompressed map is 10.9, and the successful percentage is 99.9%. $\mathcal{H}$ in the figures means the average entropy obtained from Eq.(4.4) for each partial maps. The lower graph shows the relation between entropy and the percentage of failed trials.

As shown in these figures, both the risk of efficiency loss and the risk of increase of failures are small if we choose pairs of (ν, ε) that yield small entropy. As things turned out, a VQ map with high efficiency and success ratio could be obtained only if at most top 10 pairs of (ν, ε) in the entropy evaluation were tried.

We show the top five VQ maps on each criterion in Table 6.6 and Table 6.7. In Table 6.6(a) and Table 6.7(a), the rankings of failure rate and step loss are also written, while the ranking of entropy is written in (b) and (c). These tables are suitable to figure out the three-dimensional relation of the criteria.

When the compression ratio is $1 : 0.1$, the ranking of entropy evaluation is identical with that of failure rate at the first and second place. When the compression ratio is $1 : 0.02$, two VQ maps appear both in Table 6.7(a) and (b).

On the other hand, the efficiency loss does not relate to the entropy evaluation as shown in (a) and (c) of Table 6.6 and Table 6.7. A high failure rate is caused by partial destruction of a map. When the value of the entropy is high, various vectors exist and they must be categorized into a limited number of clusters. When the vectors are replaced by a representative vector in this case, the destruction seems to happen frequently.

Table 6.6: Rankings of VQ Maps (compression ratio: 1 : 0.1)

(a) Small Entropy

order	$(w_x, w_y, w_\theta, w_r, w_\varphi)$	failure rate (ranking)	step loss (ranking)	entropy $\mathcal{H}$
1	$(1, 1, 1, 9, 15)$	0.26[%] (1)	0.06[step] (47)	0.234
2	$(1, 1, 6, 9, 15)$	0.34[%] (2)	0.09[step] (54)	0.246
3	$(1, 1, 3, 9, 15)$	0.78[%] (18)	0.06[step] (46)	0.255
4	$(1, 3, 1, 9, 15)$	0.90[%] (22)	0.13[step] (66)	0.267
5	$(3, 1, 1, 9, 15)$	0.56[%] (9)	0.12[step] (60)	0.269

(b) Small Failure Rate

order	$(w_x, w_y, w_\theta, w_r, w_\varphi)$	failure rate	step loss	entropy $\mathcal{H}$ (ranking)
1	$(1, 1, 1, 9, 15)$	0.26[%]	0.06[step]	0.234 (1)
2	$(1, 1, 6, 9, 15)$	0.34[%]	0.09[step]	0.246 (2)
3	$(1, 1, 3, 3, 15)$	0.36[%]	0.03[step]	0.289 (8)
4	$(3, 1, 1, 3, 15)$	0.38[%]	0.02[step]	0.300 (13)
5	$(21, 1, 1, 3, 15)$	0.38[%]	0.10[step]	0.362 (41)

(c) Small Efficiency Loss (Increase of Steps)

order	$(w_x, w_y, w_\theta, w_r, w_\varphi)$	failure rate	step loss	entropy $\mathcal{H}$ (ranking)
1	$(3, 3, 3, 3, 15)$	2.30[%]	-0.03[step]	0.391 (71)
2	$(21, 1, 3, 3, 15)$	0.90[%]	-0.02[step]	0.414 (89)
3	$(1, 3, 3, 1, 15)$	0.90[%]	-0.02[step]	0.310 (16)
4	$(1, 3, 3, 3, 15)$	1.94[%]	-0.01[step]	0.330 (24)
5	$(3, 3, 6, 3, 3)$	1.52[%]	-0.01[step]	0.435 (101)

Table 6.7: Rankings of VQ Maps (compression ratio: 1 : 0.02)

(a) Small Entropy

order	$(w_x, w_y, w_\theta, w_r, w_\varphi)$	failure rate (ranking)	step loss (ranking)	entropy $\mathcal{H}$
1	$(1, 1, 1, 9, 15)$	4.18[%] (19)	0.56[step] (49)	0.234
2	$(1, 1, 6, 9, 15)$	2.66[%] (9)	0.16[step] (13)	0.246
3	$(1, 1, 3, 9, 15)$	0.80[%] (2)	0.71[step] (63)	0.255
4	$(1, 3, 1, 9, 15)$	6.62[%] (31)	0.70[step] (62)	0.267
5	$(3, 1, 1, 9, 15)$	1.50[%] (4)	0.56[step] (50)	0.269

(b) Small Failure Rate

order	$(w_x, w_y, w_\theta, w_r, w_\varphi)$	failure rate	step loss	entropy $\mathcal{H}$ (ranking)
1	$(1, 1, 3, 3, 15)$	0.70[%]	0.44[step]	0.289 (8)
2	$(1, 1, 3, 9, 15)$	0.80[%]	0.71[step]	0.255 (3)
3	$(3, 1, 1, 3, 15)$	0.98[%]	0.45[step]	0.297 (13)
4	$(3, 1, 1, 9, 15)$	1.50[%]	0.56[step]	0.269 (5)
5	$(1, 1, 6, 3, 15)$	1.68[%]	0.46[step]	0.292 (10)

(c) Small Efficiency Loss (Small Increase of Steps)

order	$(w_x, w_y, w_\theta, w_r, w_\varphi)$	failure rate	step loss	entropy $\mathcal{H}$ (ranking)
1	$(3, 1, 6, 9, 15)$	1.90[%]	-0.09[step]	0.289 (9)
2	$(1, 3, 3, 9, 15)$	2.90[%]	-0.07[step]	0.294 (11)
3	$(3, 1, 6, 3, 15)$	7.42[%]	-0.02[step]	0.339 (31)
4	$(1, 3, 6, 3, 15)$	6.30[%]	-0.03[step]	0.336 (29)
5	$(1, 3, 6, 9, 3)$	4.90[%]	-0.04[step]	0.373 (53)

6.2.5 High Ratio Compression

Table 6.8 shows the top three sets of parameters that mark small values of entropy. The entropy function tells that x and y should be divided fine, and that r and φ should not be divided at the blocking phase for efficient compression. It means that the state-action map is redundant on xy-plane. This result coincides with Table 6.5.

Table 6.8: VQ Maps for High Ratio Compression

	$(w_x, w_y, w_\theta, w_r, w_\varphi)$	entropy $\mathcal{H}$	N_ε
A)	$(1, 1, 1, 9, 15)$	0.234	135
B)	$(1, 1, 6, 9, 15)$	0.246	810
C)	$(1, 1, 3, 9, 15)$	0.255	405

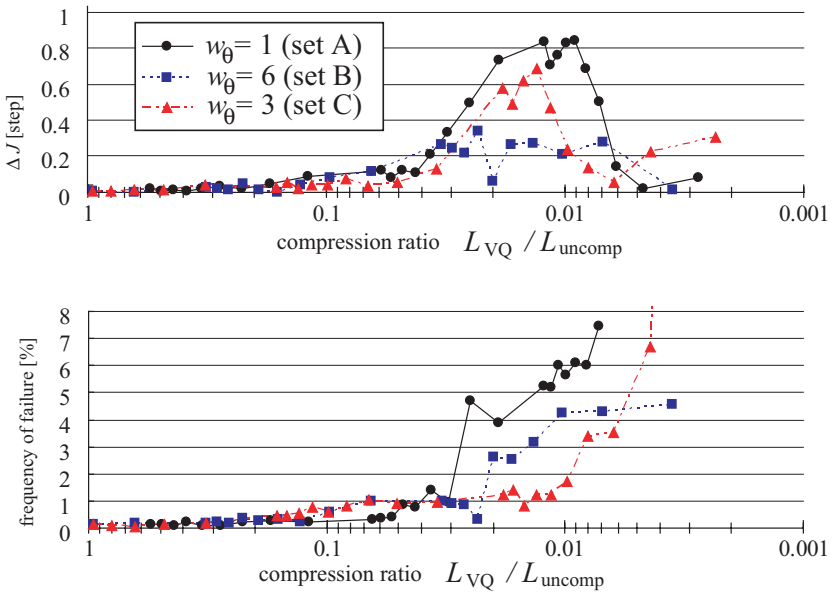


Fig. 6.15: Relation between Compression Ratio and Efficiency Loss

The efficiency losses of the VQ maps with these ways of blocking measured by the simulation in Sec. 3.5 are illustrated in Fig.6.15. The frequency of failures gradually increases to 1[%] in association with the compression ratio with every set of parameters. Once the compression ratio exceeds its limit, the frequency of failures jumps a whopping some percentage from 1[%]. It seems that the number of representative vectors

runs short definitely at the jump of the frequency of failures. We can regard a VQ map is broken at this moment. The VQ map with the set A is broken with a lower compression ratio than the others though this set yields the smallest entropy. It seems that the compression ratio at breakdown of a VQ map is related to N_ε when the values of entropy are not much different than each other.

Then we observe how the uncompressed map is changed in the clustering process. When the parameters are $(w_x, w_y, w_\theta, w_r, w_\varphi) = (1, 1, 6, 9, 15)$, we can observe it on xy-plain. That is because each pair of (i_x, i_y) is related to one vector, which can be regarded as a block in $\theta r\varphi$-space. Figure 6.16 shows the progression of clustering in one of the partial maps. Each area painted by the same color has the same representative vector.

As shown in the figures from (a) to (e), every edge of the field is clustered into one area. In the clustering process from (a) to (e), the internal areas are merged one after another. When $N_c = 6$, an edge area and an internal area are merged. At that moment, the VQ map is broken. The frequency of failures, shown in Fig .6.15, jumps from 0.26[%] to 2.54[%] when the number of clusters changes from $N_c = 7$ to $N_c = 6$. That is because the policy for (θ, r, φ) is made to be common in the edge area and in the internal area though they are utterly different.

Figure 6.17 is a part of the VQ map with $N_c = 7$ for the same position of the ball in Fig. 6.12. We notice that the VQ map is simplified from the original state-action map when these figures are compared. That is because the decision making toward the ball is generalized on the seven parts of xy-plain respectively by the clustering shown in Fig. 6.16. $218,906$ in the VQ map actions are changed from the original state-action map, which contains $765,450$ actions.

6.2.6 Experiment with Actual Robot

We choose the smallest one from the VQ maps that mark more than 99% success at the simulation for experiment. The VQ map has the following parameters: $(w_x, w_y, w_\theta, w_r, w_\varphi) = (1, 1, 3, 9, 15)$ and $N_c = 8$ (in each sub map). The compression ratio of this map is $N_{\mathrm{VQ}}/N_{\mathrm{uncomp}} = 0.015$. We also create a DLVQ map from this VQ map. Its size is 87[%] toward the VQ map. In the experiment, we use not the DLVQ map but the VQ map.

We put a robot and a ball on the field, and the robot approached the

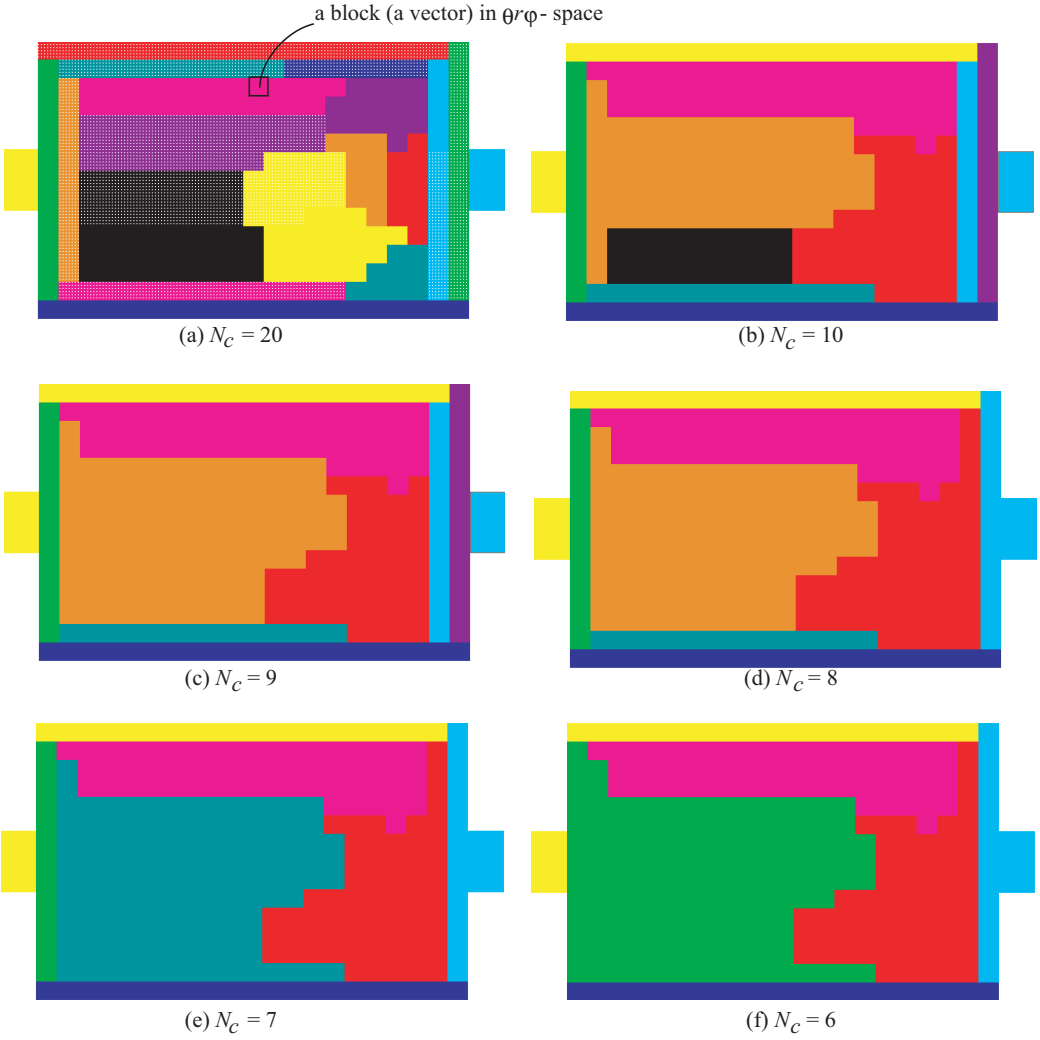


Fig. 6.16: Progression of clustering (set B, sub map with $N_p = 1$)

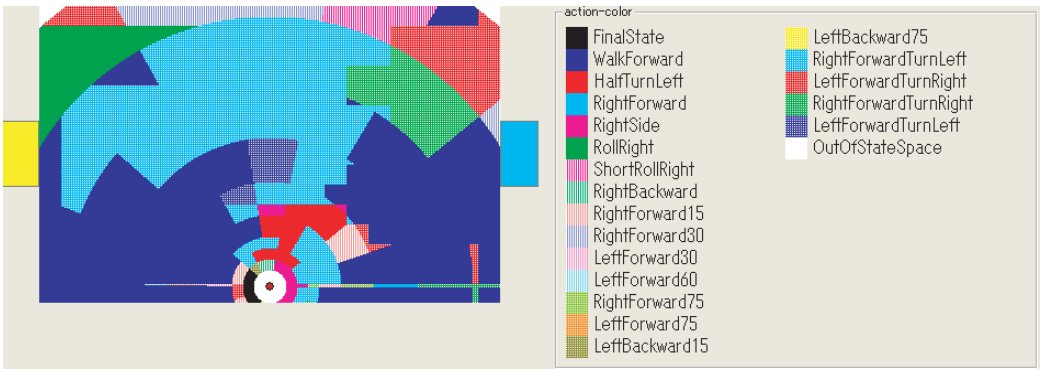


Fig. 6.17: Part of VQ map

ball based on a self-localization result [Ueda, 2002] and measurement of the ball's position with its camera. It is difficult for a robot to stop at a final state with perfect estimation of the state variables. The robot then marched forward when it sensed a final state to push the ball, and the trial was determined to be a success if the ball rolled to the side of the target goal (i.e., if x value of the ball increased).

Since a robot sometimes goes out of the state space due to sensing errors, actions are assigned for states in the $r < 120$, $\varphi \leq 75$ or $\varphi < -75$ area. The robot returns to the state space with the assigned actions. Nevertheless, if the ball goes to one of the robot's blind spots or rolls in another direction as a result of a careless touch from the robot, the trial is a failure.

Three conditions with different types of initial states are prepared.

Condition I The robot is put at a point on the $x = 0$ line with $\theta = 0$. The ball is put at a point on the $x = 1750$ line that is in front of the target goal. The points are chosen at random. The number of steps between the compressed map and the uncompressed map is compared.

Condition II The robot is put at a point on the $x = 1750$ line with $\theta = 180$, and the ball is placed at a point on the $x = 0$ line. In this task, the state tends to transfer the rim of the state space when the robot reverses its orientation while traveling. That part of the compressed map is checked through this task.

Condition III This task uses the most difficult initial states for compressed maps with small α. The robot is placed next to a wall at $y = -1350$ or $y = 1350$ with $\theta = 180$. x is chosen at random. The ball is placed 200[mm] from the wall and 300[mm] in front of the robot.

Figure 6.18 shows an example of each condition. These trajectories of the robot were obtained with the compressed map. These were ideal trajectories. As a matter of fact, the robot sometimes chose fruitless actions in trials (especially, in Condition III) due to measurement errors of the ball's position.

Table 6.9 shows the results of the 50 trials. The two maps showed quite similar results in each condition. In condition I and II, the measurement errors of the ball affected the failure percentages much more than the ability of maps.

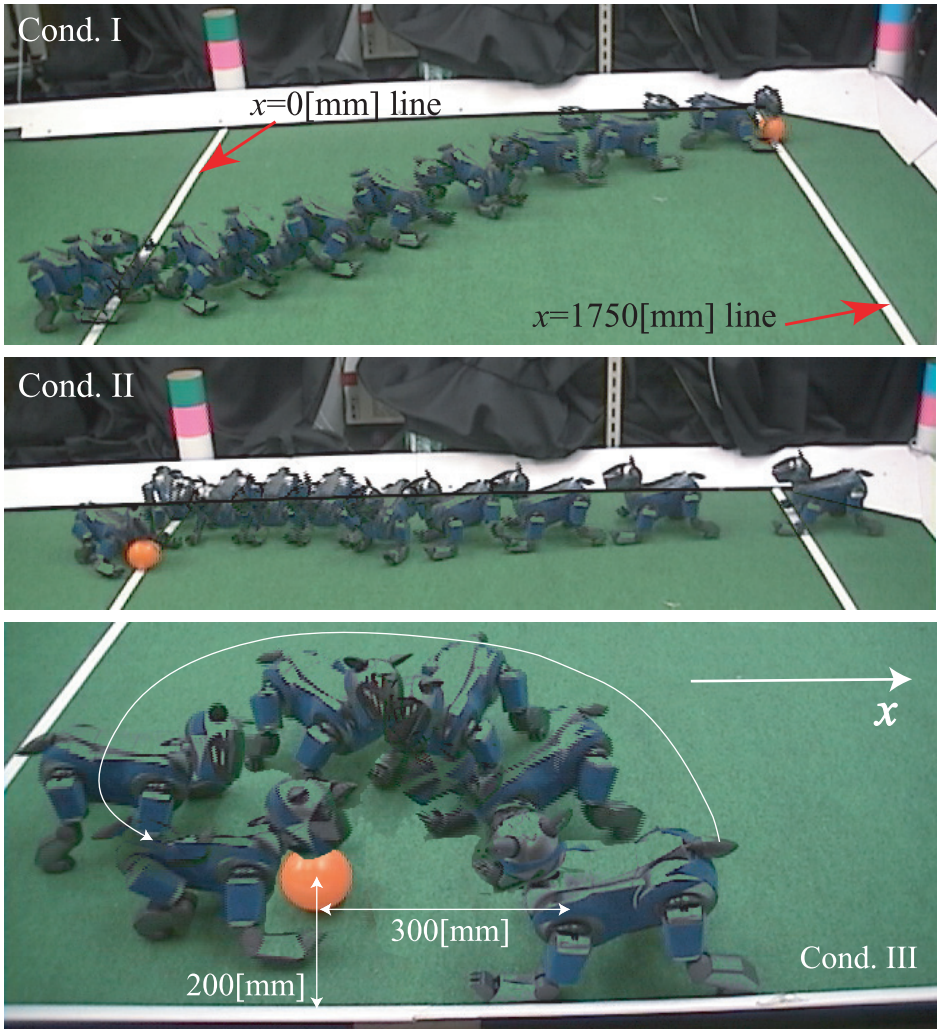


Fig. 6.18: Trajectories of ERS-210 with The VQ Map

Table 6.9: Results of the Experiment with Actual ERS-210

	Uncompressed		Compressed	
	Rate of success	Average steps	Rate of success	Average steps
Condition I	100%	21.1	100%	21.1
Condition II	96%	30.0	94%	29.2
Condition III	68%	28.5	68%	27.2

6.3 Scoring Task by Two Robots

6.3.1 Objective

In the later part of this chapter, a purpose of this thesis, which is to install a huge state-action map in limited memory of a robot, is demonstrated. We venture to create a huge state-action map for scoring by two ERS-210s. The state-space for the cooperative scoring task is spanned by all of the state variables of two robots and one ball. The space is discretized to 600 million states and the value iteration algorithm is executed. In each discrete state, a walking action or a kicking action is allocated. All of the actions are not designed for cooperative behavior. However, we think that state-action sequences for cooperative behavior can be obtained by the state-action map if they are required for maximizing the evaluation of optimal control. If the state-action map can be compressed to a VQ map whose size is smaller than the amount of flash memory of an ERS-210 without large loss of optimality, we consider that the purpose is accomplished.

We have to say beforehand that maps for this task are evaluated not on actual ERS-210s but on simulated robots. Though development of accurate kicking actions are inevitable for the cooperative scoring task, we have never developed them for ERS-210s. We verify whether the VQ method can create an efficient VQ map that can be installed on the robots or not in this thesis.

6.3.2 Related Works and Our Stance

Almost all of studies in cooperative robotics handle the curse of dimensionality implicitly and explicitly. In many studies of multi-agent systems, some kinds of basic behavior are planned for each robot beforehand, and cooperative behavior is planned as the combination of them. For example, there are many cases of such studies [Nitschke, 2006; Khojastech, 2004; Fraser, 2005; Fujii, 2004] in RoboCup. In this case, the problem is divided into each subproblems of each robot. State space of all robots are also divided into subspace of each robot. Those studies have made it possible to create various types of cooperative behavior of robots. The proposed method in this thesis and the value iteration algorithm can be applied to some of the above methods. They can create a state-action map for a subtask in subspace.

The above methods are very useful for creating cooperative behavior of robots and inevitable for control of more than ten robots. From the

viewpoint of optimal control, however, excessive assist for emergence of cooperation does not always effective for finding an optimal policy for an evaluation function. Especially in RoboCup, offense and defense can be done by one robot. Cooperation is merely an option. In such a case, we can know whether cooperative behavior is effective or not only by solving an optimal control problem completely.

We therefore try creating a huge state-action map without any technique on the study of multi-robot systems in this thesis. Currently, this approach is not realistic when the number of the robots is more than two. It is, however,meaningful if cooperative behavior occurs from the framework of optimal control problems.

6.3.3 Problem Definition and Assumption

We assume that there are only two teammate ERS-210s on the field as shown in Fig. 6.19. Their task is to bring the ball into the sky-blue goal as soon as possible. Each robot measures its pose (x, y, θ) and the position of the ball (r, φ) as an ERS-210 does on the ball approaching task. The speed of the ball cannot be measured. They can exchange the measurements by using their wireless LAN devices.

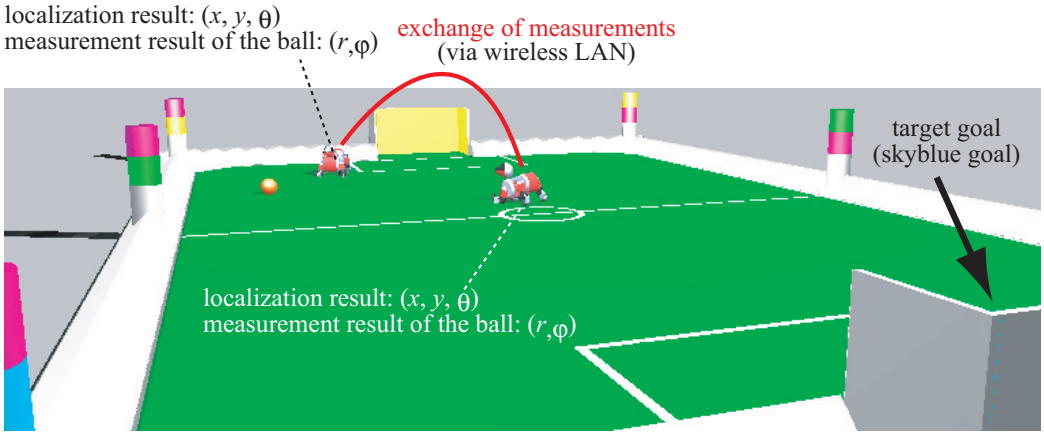


Fig. 6.19: Condition of the Task

As explained in Sec. 6.1.1, 16[MB] is the upper limit of memory for installing executable code, state-action maps, and other data through a flash memory. When we use a state-action map, 10[MB] is a rough upper limit of

its size due to the existence of other data.

Toward the above problem, we try applying DP and our compression method. Before that, we have to confirm the following assumptions. At first,

- the robots have a common VQ map whose size is less than 10[MB], respectively. Moreover, they have common setting of software and hardware.

The robots do not have to use a common VQ map if each robot is assigned its task in advance. However, such a method restricts dynamic task assignment and becomes unnecessary restriction for solving the optimal control problem. The VQ map contains an appropriate pair of actions of two robots toward every state. Here a state does not mean respective states of the robots and the ball, but that of the whole system to be controlled.

Then,

- they exchange their measurements and share common perceptions of an estimated state. The cycle of exchanges is much more frequent than that of decision making.

Based on the shared measurements, each robot must choose its action from the pair of actions. At the time, each robot has its role. Since the action of each robot can be fixed by the measurements uniquely, they do not need to confirm their roles with each other.

6.4 Value Iteration for Scoring Task

6.4.1 State Space

The state variables are defined as shown in Table 6.10. Figure 6.20 illustrates their definition intuitively. A robot is called Robot1. (x, y, θ) of Robot1 is renamed (x_1, y_1, θ_1). Another is called Robot2 and its pose is renamed (x_2, y_2, θ_2). When both of the robots can observe the ball and measure its position, the measurement results of the ball become redundant. We use the measurements of Robot2 as the position of the ball (r, φ), which is defined in the task of going to ball, for composing the state space. The position of the ball is sometimes represented by $(x_\mathrm{b}, y_\mathrm{b})$ on Σ_field. The case where both of the robots cannot observe the ball is not considered in DP. We prepare a hand-coding policy in that case as mentioned later.

Table 6.10: State Variables for Scoring Task

state variable	domain
x_1, x_2	$[-2100, 2100)$[mm]
y_1, y_2	$[-1350, 1350)$[mm]
θ_1, θ_2	$[-180, 180)$[deg]
r	$[150, 3150)$[mm]
φ	$[-95, 95)$[deg]

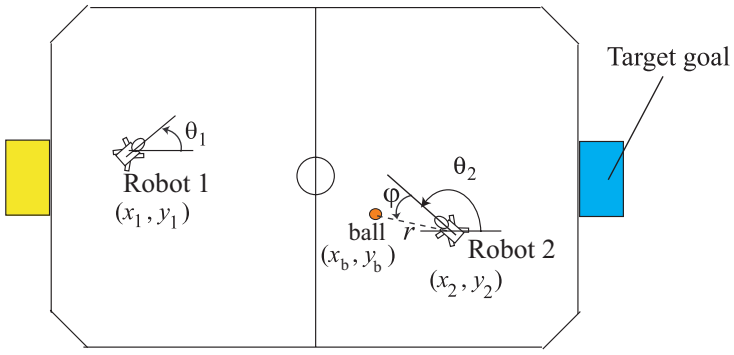


Fig. 6.20: Coordinates and State Variables

The domain of every state variable is defined as shown in the table. The domain of φ is wider than that of the task of going to ball. It means that the risk of missing of the ball is larger than that in the task of going to ball.

Since we prepare the hand-coding policy as mentioned above, the domain can be expanded. The state space, $\mathcal{X}$, is spanned by $x_1, x_2, y_1, y_2, \theta_1, \theta_2, r$, and φ in the domain.

In the task, the observer of the ball can be changed from Robot2 to Robot1; otherwise their roles are fixed. The robots change their names and new observer becomes Robot2 in that case. Robot2 is chosen by the following rule.

- A robot can observe the ball, and its measurement (r, φ) is in the domain defined in Table 6.10, the robot can be Robot2.
- If both of the robots can be Robot2, the robot whose measurement of r is smaller than that of the other is chosen as Robot2.
- If both of them cannot be Robot2, DP does not support the state.

6.4.2 Actions

Robot1 and Robot2 take their actions synchronously. An action in value iteration is then defined as a pair of their actions. When Robot1 and Robot2 choose actions from $\mathcal{A}^{\mathrm{R1}} = \{a_i^{\mathrm{R1}} | i = 0, 1, 2, \ldots, M^{R1} - 1\}$ and $\mathcal{A}^{\mathrm{R2}} = \{a_j^{\mathrm{R2}} | j = 0, 1, 2, \ldots, M^{R2} - 1\}$ respectively, the set of action in value iteration is defined as their direct product: $\mathcal{A} = \mathcal{A}^{\mathrm{R1}} \times \mathcal{A}^{\mathrm{R2}}$.

We define 14 walking actions as shown in Table 6.11(a). We assume that every robot executes each action once in a time step. 12 actions in this table are chosen from Table 6.2. *Stay* and *Backward* are newly added. Though we want to use all of the actions in Table 6.2, we have to reduce the number of actions; otherwise the number of actions in $\mathcal{A}$ reaches 38^2. Since calculation amount of value iteration is in proportion to the number of actions, that number is not realistic.

As shown in the table, each robot uses a different set of actions. The set for Robot2, $\mathcal{A}^{\mathrm{R2}}$, is chosen as a suitable set of actions for going to ball. We select them from the state-action map for the task of going to ball. Moreover, three kicking actions belong to $\mathcal{A}^{\mathrm{R2}}$ as shown in Table 6.11(b). A kicking action changes the position of the ball. r_{after} and φ_{after} mean the distance and the direction of the ball from Robot2 after a kicking action. The pose of Robot2 does not change after any kicking action. Robot2 always chooses a kick action when $150 \leq r < 250$[mm] and $-35 \leq \varphi < 35$[deg]. In the case of an actual ERS-210, fine position adjustment is required for a kick

Table 6.11: Actions for Cooperative Scoring Task

(a) walking actions

	name	δ_x[mm]	δ_y[mm]	δ_θ[deg]
only for Robot1	Stay	0.0	0.0	0.0
	Backward	0.0	-87.0	0.0
	RightSide	114.0	0.0	0.0
	LeftSide	-107.0	0.0	0.0
for both robots	Forward	0.0	113.0	0.0
	TurnRight	20.0	5.0	-32.5
	TurnLeft	-15.0	5.0	28.0
only for Robot2	ShortForward	0.0	66.0	0.0
	ShortRollRight	25.0	30.0	18.0
	ShortRollLeft	-25.0	30.0	-30.0
	RightForward15	16.0	84.0	2.0
	LeftForward15	-14.0	102.0	-2.6
	RightBackward15	36.0	-94.0	0.0
	LeftBackward15	-35.0	-95.0	0.0

(b) kicking actions

	name	r_{after}[mm]	φ_{after}[deg]
only for Robot2	KickForward	2000	0.0
	KickRight	2000	-75.0
	KickLeft	2000	75.0

from that range of (r, φ). In the simulation, however, we do not consider it for simplicity.

As for Robot1, actions in $\mathcal{A}^{\text{R1}}$ are selected as the robot can run in four directions and turn to both directions. We limit Robot1's action to Stay when when Robot2 kicks the ball.

The above setting looks like separation of roles of two robots. However, it is never a definite role assignment because the robots can switch their names each other in the task. Robot1 can go to the ball so as to kick. In this case, Robot1 and Robot2 change their names when the distance of the ball from Robot1 becomes nearer than that of Robot2. The number of actions in $\mathcal{A}$ is $M = 73$ with the above setting.

6.4.3 State Space Discretization and Final State Definition

We discretize the state space $\mathcal{X}$ as shown in Table 6.12. The number of states reaches $14^2 \cdot 9^2 \cdot 15^2 \cdot 9 \cdot 19 = 610,829,100$.

Table 6.12: Discretization of the state space

	definition of intervals	
x_1	$[x_1]_i \equiv \big[300i - 2100, 300(i+1) - 2100\big)$[mm]	$(i = 0, 1, \ldots, 13)$
x_2	$[x_2]_i \equiv \big[300i - 2100, 300(i+1) - 2100\big)$[mm]	$(i = 0, 1, \ldots, 13)$
y_1	$[y_1]_i \equiv \big[300i - 1350, 300(i+1) - 1350\big)$[mm]	$(i = 0, 1, \ldots, 8)$
y_2	$[y_2]_i \equiv \big[300i - 1350, 300(i+1) - 1350\big)$[mm]	$(i = 0, 1, \ldots, 8)$
θ_1	$[\theta_1]_i \equiv \big[24i - 180, 24(i+1) - 180\big)$[deg]	$(i = 0, 1, \ldots, 14)$
θ_2	$[\theta_2]_i \equiv \big[24i - 180, 24(i+1) - 180\big)$[deg]	$(i = 0, 1, \ldots, 14)$
r	$[r]_i \equiv \begin{cases} \big[100i + 150, 100(i+1) + 150\big)\text{[mm]} & (i = 0, 1, \ldots, 2) \\ \left[50\left(i - \frac{3}{2}\right)^2 + \frac{675}{2}, 50\left(i - \frac{1}{2}\right)^2 + \frac{675}{2}\right)\text{[mm]} & (i = 3, 4, \ldots, 8) \end{cases}$	
φ	$[\varphi]_i \equiv \big[10i - 95, 10(i+1) - 95\big)$[deg]	$(i = 0, 1, \ldots, 18)$

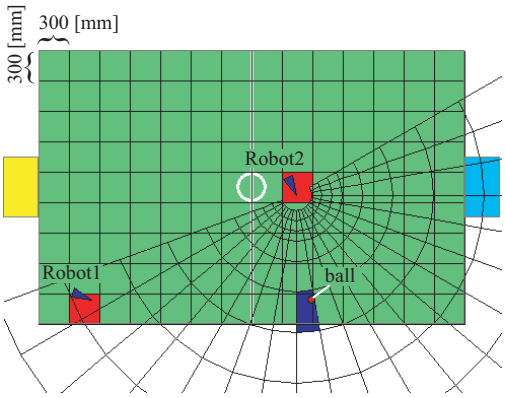


Fig. 6.21: Discretization for Scoring Task

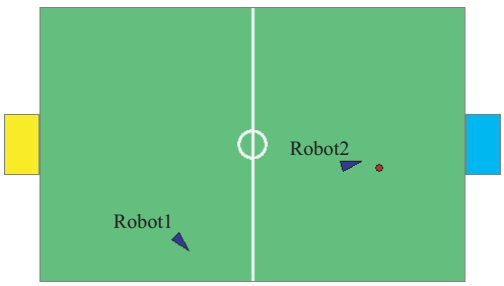


Fig. 6.22: An Example of Final State

As mentioned before, Robot2 chooses a kick action when $150 \leq r < 250$[mm] and $-35 \leq \varphi < 35$[deg]. The final states are also chosen when (r, φ) is in that range. When Robot2 has more than 50% of probability for scoring after a kicking action from a discrete state, we regard the state as a final state. Whether a discrete state s is a final state or not is judged by the following Monte Carlo simulation:

1. ζ poses of Robot2 (x_2, y_2, θ_2) are chosen from a discrete state at random.

2. Every kicking action is tried from all of the chosen poses and the number of goals are counted respectively.

If one of the kicking actions can yield more than $\zeta/2$ goals, the discrete state is regarded as a final state. Figure 6.22 illustrates an example of final states on our definition. There is a very high possibility of scoring in this state at KickForward of Robot2.

We should explain the reason why the case where the ball is in the goal is not defined as the final state. The reason is that the actual robots cannot make accurate judgments whether the ball is in the goal or not due to the measurement errors their poses and the ball. Even if the measurement is complete, moreover, Robot2 does not make a shot unless the goal is absolutely certain. Under the discretization in Table 6.12, such discrete states are rare indeed.

In the case of our definition of final states, the task is finished truly when Robot2 can score. If not so, the state is changed to another and the robots continue the task.

6.4.4 State Transition

A state transition can be represented by

$$\begin{aligned} \mathcal{P}^a_{ss'} &= P(s'|s,a) \quad \text{where, } a \in \mathcal{A}, \\ s &= ([x_1], [x_2], [y_1], [y_2], [\theta_1], [\theta_2], [r], [\varphi]), \text{and} \\ s' &= ([x_1]', [x_2]', [y_1]', [y_2]', [\theta_1]', [\theta_2]', [r]', [\varphi]'). \end{aligned} \tag{6.15}$$

All of the state transition probabilities are calculated and stored on memory before value iteration; otherwise a probability of the same state transition is calculated redundantly.

Reduction and Decomposition of State Transitions

By simple arithmetic, however, the combinations of (s, s', a) reaches $N^2M = 610,829,100^2 \cdot 73 = 27,237,189,826,697,130,000$. Thus, whether this number can be reduced or not determines the feasibility of DP.

This number can be reduced by cutoff of state transitions that fulfill $\mathcal{P}^a_{ss'} < \eta$. In this case, the number of state transitions is not over $1/\eta$ toward a set of (s, a). We choose $\eta = 0.01$ for the scoring task. In this case, the number is reduced to $610,829,100 \cdot 73 \cdot 100 = 4,459,052,430,000$.

When we can decompose state transitions into some independent events, moreover, each state transition probability can be represented by the multiplication of the probabilities. In this case, the amount of memory for recording the state transitions can be reduced.

We can consider the following events in the task.

(i) displacement of a robot by an walking action

(ii) relative displacement of the ball by Robot2's walk

(iii) displacement of the ball by Robot2's kick

(i) and (ii) can be handled in the same way as the state transitions in the going to ball task if no collision of the two robots is considered. P_{robot} and P_{ball} are obtained by Monte Carlo integrations respectively, and stored on the look-up tables shown in Table 6.3. A state transition probability toward a set of walking actions by the two robots can be calculated from P_{robot} and P_{ball} as

$$\begin{aligned}\mathcal{P}^a_{ss'} = & P_{\text{robot}}(\Delta[x_1], \Delta[y_1], [\theta_i]' \big| [\theta_i], a^{R1}) \\ & \cdot P_{\text{robot}}(\Delta[x_2], \Delta[y_2], [\theta_2]' \big| [\theta_2], a^{R2}) \\ & \cdot P_{\text{ball}}([r]', [\varphi]' \big| [r], [\varphi], a^{R2}).\end{aligned} \tag{6.16}$$

For the event (iii), $\mathcal{P}^a_{ss'}$ can be represented by

$$\mathcal{P}^a_{ss'} = P_{\text{kick}}([r]', [\varphi]' \big| a^{\text{R2}}) \quad (a^{\text{R2}}\text{: a kicking action}) \tag{6.17}$$

if we assume that the ball never collides with the wall or a robot. This equation means that the state transition depends only on kicking actions. Since we do not consider the distribution of the ball position after a kick, P_{kick} is deterministic in the discrete state space $\mathcal{S}$.

Virtual State Transition

We have decomposed the state transition probabilities into $P_{\text{robot}}, P_{\text{ball}}$, and P_{kick}. On the other hand, we have neglected the existence of the following events that occur in the task.

(1) collision between the ball and the wall around the field

(2) change of Robot1/2 (abbreviation of Robot1 and Robot2)

(3) collision between the ball and a robot

(4) collision between Robot1 and Robot2

(1) and (2) should be especially considered in value iteration. If (1) is neglected, the ball is regarded as being out of the field. In this case, a state-value function obtained by value iteration will contain great errors. When (2) is not considered, the roles of the robots are fixed. If (3) and (4) are neglected in value iteration, the efficiency of the state-action map will decrease.

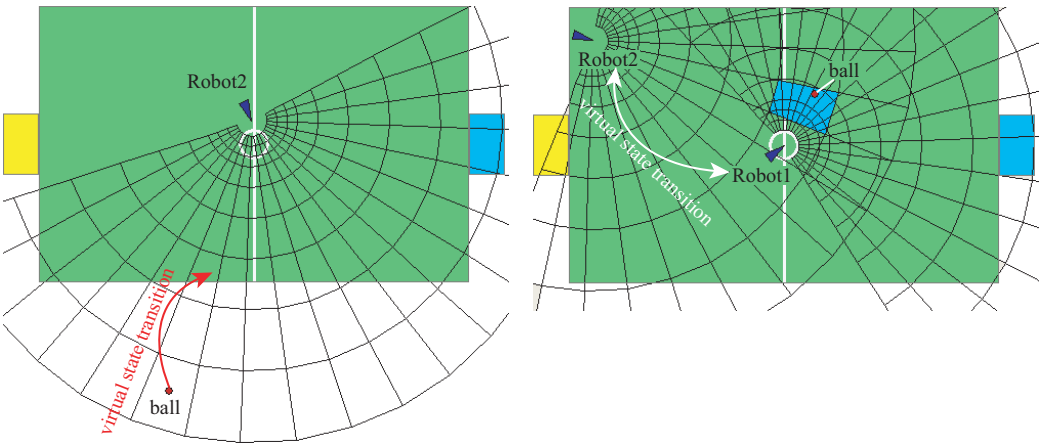
However, considerations of the above events increase the dependency of each state variable. If we consider (1), P_{kick} depends not only on a kicking action, but also the pose of Robot2. If Robot2 and Robot1 should be changed after the kick, the pose of Robot1 should also be considered. P_{robot} and P_{ball} also depend on all of the eight state variables if (2) is considered.

To solve this problem, we introduce *virtual states* and *virtual state transitions*. In the case of (1), we decompose a state transition into the following two processes: 1) the ball goes out of the field and stops, and 2) the ball returns from the outside of the field to the inside instantaneously. A ball out state, as shown in Fig. 6.23(a), is regarded as a virtual state. The exportation of the ball is regarded as a virtual state transition. Since virtual state transitions occur instantaneous, no reward is given. In the actual world, such a pair of state transitions never happens.

In a value iteration algorithm, however, we can consider them if we prepare memory space for recording the value of virtual states and their state-transition probabilities. There are $118,788,390$ ball out states in the set of discrete states defined in Table 6.12 and the memory can be prepared.

In a virtual state transition from ball out state s, interval $[r]$ of state s moves to an inner interval, which contains a part of the field, as shown in Fig. 6.23(a). Since s' is fixed toward a ball out state s, $\mathcal{P}^a_{ss'} = P_{\text{ball out}} = 1$ is the state transition model for each of them. In this case, a is the teleportation. We call it *a virtual action*.

We also regard the event (2) as a virtual state transition. Actions of the robots and a change of Robot1/2 by the actions is decomposed into



(a) a ball out state (b) a reverse state

Fig. 6.23: Virtual States and Virtual State Transitions

a set of two state transitions. We define reverse states, which are virtual states, as the states where Robot1/2 should be switched. An example is shown in Fig. 6.23(b). In a reverse state, the change of Robot1/2 occurs instantaneously. We implement a Monte Carlo algorithm to find a reverse state. In the algorithm, distances of the ball from Robot1 and Robot2 are compared from various continuous state $\boldsymbol{x}$ that belongs to s when both of the robots can observe the ball. If the average distance from Robot1 is smaller than the other, the state s is regarded as a reverse state.

Virtual state transition probabilities P_{rev} from reverse states can be represented as

$$P_{\text{rev}}([r]', [\varphi]' | \Delta[x], \Delta[y], [\theta_1], [\theta_2], [r], [\varphi]). \tag{6.18}$$

P_{rev} can be computed by a Monte Carlo sampling of values $(\Delta x, \Delta y, \theta_1, \theta_2, r, \varphi)$ from the intervals $(\Delta[x], \Delta[y], [\theta_1], [\theta_2], [r], [\varphi])$. $(\Delta x, \Delta y)$ denotes a relative position of Robot1 and Robot2. In our algorithm for computing P_{rev}, the sampling is not at random but fixed. Three values (two boundary values and the mean value) are chosen from $\Delta[x]$ and $\Delta[y]$ and two boundary values are chosen from the other intervals. $45,242,367$ kinds of virtual state transitions are recorded on a look-up table by this implementation.

6.4.5 Reward

The reward is given as $\mathcal{R}^a_{ss'} = -1$[step] when the robots take their actions respectively. When both of the robots cannot observe the ball, $\mathcal{R}^a_{ss'} = -\infty$[step] is given. Therefore, the purpose of decision making is to reach a final state as small number of steps as possible without losing the ball.

6.4.6 Value Iteration

We implement the value iteration algorithm on a computer that has 3.0[GB] RAM, a 300[GB] hard disk drive (HDD) and 3.2[GHz] Pentium D CPU. The algorithm starts from the definition of state space $\mathcal{S}$, set of actions $\mathcal{A}$, and reward $\mathcal{R}^a_{ss'}$. After that, two look-up tables of a state-value function and a state-action map are created on HDD. Each look-up table is divided into $14^2 = 196$ files with respect to a set of $([x_1], [x_2])$.

Each action is represented by a unique number from 0 to 72. 1 byte is required for recording each of these numbers. Before value iteration, zero is filled in the state-action map except the elements of final states, ball out states, and reverse states. They are given unique numbers: 255, 254, and 253 respectively.

In the look-up table for the state-value function, final states are given zero as their value. Nonzero values are given to the other states. A 2-byte unsigned integer is used for representing a value. -1[step] is equivalent to 100 on the integer data.

In the next process, $P_{\text{robot}}, P_{\text{ball}}$, and P_{rev} are computed respectively. ($P_{\text{ball out}}$ and P_{kick} are deterministic.) As shown in Table 6.13, all of the state transition probabilities can be represented by much smaller kinds of probabilities than N^2M.

Table 6.13: Number of State Transitions

probabilities	number of combinations
P_{robot}	$1,024$
P_{ball}	$7,485$
P_{rev}	$45,242,367$

The value iteration algorithm is then executed. Required files are loaded on memory, improved by Eq. (2.11), and rewritten. Since a Pentium D has two cores, computing speed is enhanced by multiprocessing. We divide our value iteration algorithm to Process 1 for $[x_1]_i$ $(i = 0, 1, \ldots, 6)$ and Process 2 for $[x_1]_i$ $(i = 7, 8, \ldots, 13)$ and execute them simultaneously.

The size of the state-action map, which is called the 8D map hereafter, can be calculated by

$$L = N \log_2 \lceil M \rceil = 610,829,100 \cdot \lceil \log_2 76 \rceil = 4,275,803,700[\text{bit}]. \quad (6.19)$$

$M = 76$ in the above calculation contains 73 kinds of action in $\mathcal{A}$, a symbol for representing a final state and two kinds of symbol for the virtual actions.

We have recorded the maximum difference of value $V(s)$ before and after the process Eq. (2.11) in each sweep. The differences are illustrated in Fig. 6.24. The horizontal axis indicates the computation time. The vertical axis is a logarithmic one for the difference of value. Roughly speaking, it takes twice processing time to reduce the difference to a tenth value as shown in this graph. If the value iteration is aborted when the difference is less than one, we can obtain a 8D map with 135 hours.

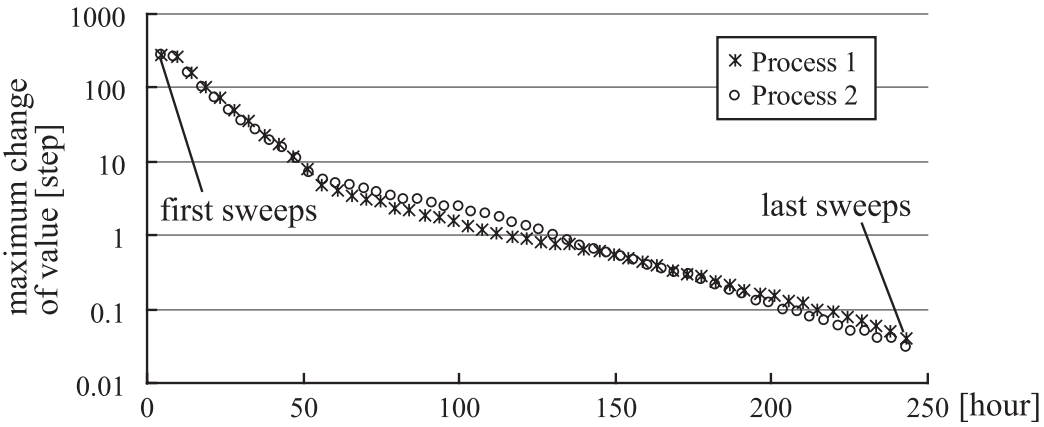


Fig. 6.24: Reduction of Maximum Change of Value

6.4.7 Behavior of Robots with The 8D Map

Emergence of Cooperative Behavior

We show some examples of the robots' behavior obtained by the state-action map. In Fig. 6.25(a), two robots, which were called RobotA and RobotB,

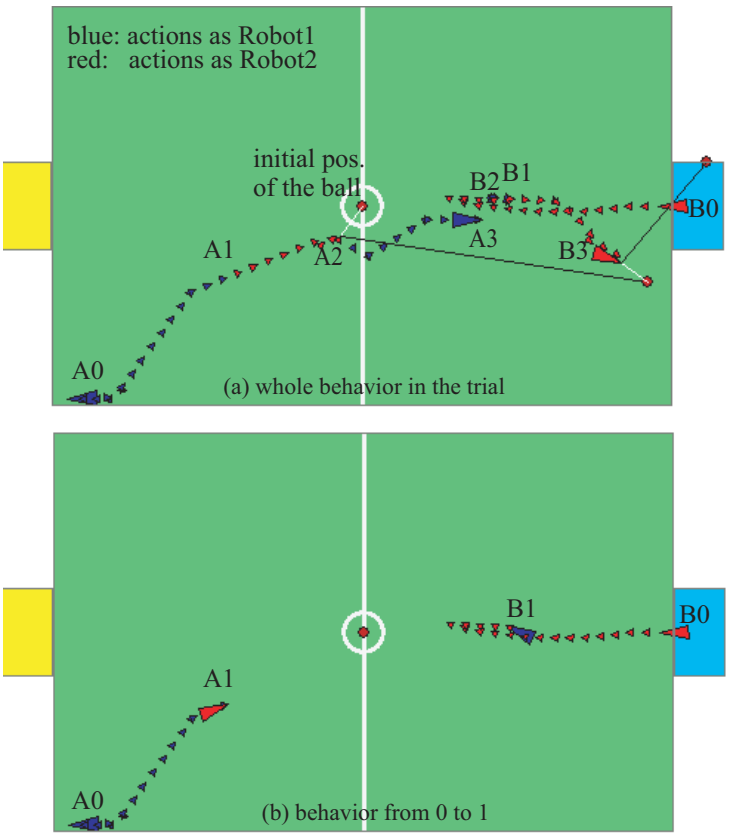


Fig. 6.25: Example of Cooperative Behavior

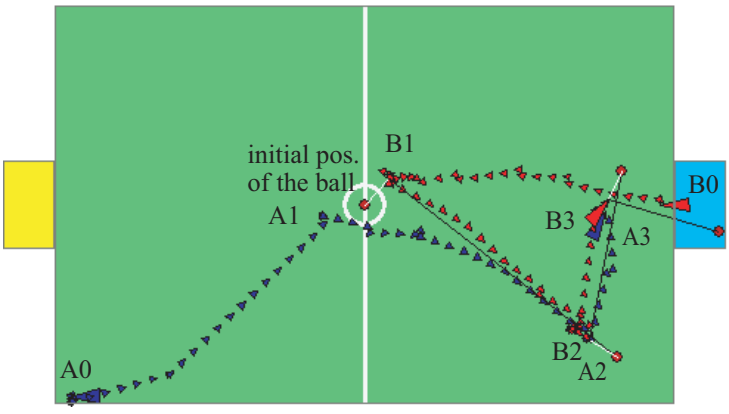


Fig. 6.26: Behavior without Cooperation

started moving from the bottom left corner and in front of the sky-blue goal respectively. The ball was put at the center of the field. In this figure, some important positions of RobotA and RobotB are numbered as Ai and Bi respectively. The numbers are synchronized. When the state became a final state, a suitable kicking action was chosen by the same Monte Carlo algorithm that was used for searching final states.

In Fig. 6.25(b), we pick out the behavior of the robots from (A0,B0) to (A1,B1) from Fig. 6.25(a). In Fig. (b), both of the robots went to the ball at first. RobotB, however, went back the way it had come along the way. RobotB handed over the ball to RobotA at the moment though it had the right of a kick as Robot2. RobotA became Robot2 just at A1. It means that the difference between Robot1 and Robot2 does not fix the task of each robot.

After that, RobotA kicked the ball at A2. RobotB waited for the kick at B2. After the pass, RobotB kicked the ball at B3 and the task was completed. In a precise sense, the pair of RobotA's kick at A2 and RobotB's wait at B2 is only a way to reduce the number of steps. Though it is our subjective judgment, however, we can regard this state-action sequence from (A0,B0) to (A2,B2) as cooperative behavior and regard RobotA's kick as *a pass*.

The evidence of our view will be given by Figure 6.26. In this figure, each robot does not consider the other and both of the robots chase the ball in this figure. These robots used a common state-action map in the space spanned by (x_2, y_2, θ_2) and (r, φ). This five dimensional state-action map will be explained later. When Fig. 6.25 and 6.26 are compared, we can understand that the robots with the eight dimensional state-action map can be regarded as cooperative.

We show another example in Fig. 6.27. In this figure, we can see two passes in one trial. RobotB waited for RobotA's kick at B1 at the first pass. RobotA then waited for RobotB's kick at A2.

Efficiency Loss by Discretization

On the other hand, the cooperative behavior given by the eight dimensional state-action map is not perfect due to the coarse discretization of the state space. As shown in Fig. 6.25, the steps from B2 to B3 could be saved if RobotB waited for the kick of RobotA just at B3. In Fig. 6.27, the positions of the robots that waited for a kick could be closer to the position where the

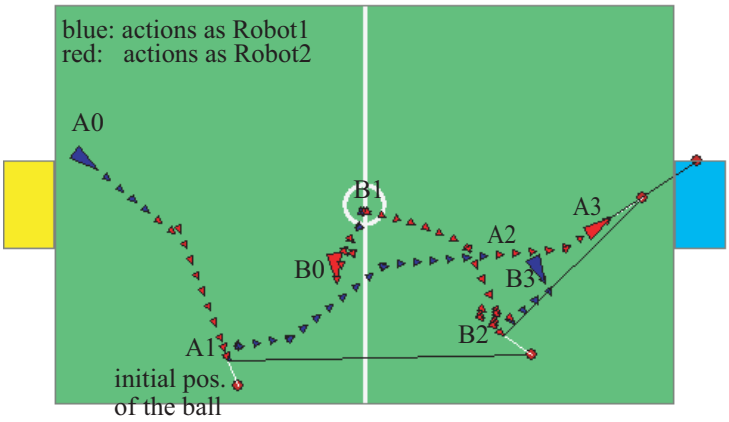


Fig. 6.27: Another Example of Cooperative Behavior

ball came. However, the margin is necessary due to the uncertainty of the poses of the robots and the position of the ball in the discrete state space. Though we assume that the state transition with a kick is definite in the continuous space, the state transition becomes probabilistic in the discrete space as represented by Eq. (6.17). If we want to solve this problem, another definition of state variables will be required.

Another problem is observed in this state-action map. In the case shown in Fig. 6.28, both of the robots stop walking at (A1,B1) due to infinite virtual state transitions between two reverse states. That is because reverse states are contained in the possible states from another reverse state. This kind of incompleteness is inevitable since a part of the value iteration algorithm must choose which robot is near the ball under the discretization.

In the continuous state space, moreover, there are many states where a score is possible, while they are not regarded as possible states for scoring in the discrete state space.

Additional Policy

The finer the discretization is, the ill-effect will be reduced. As a realistic way, on the other hand, we can cover the imperfection of the state-action map with additional control policies that are coded on the continuous state space. However, a tenuous ad-hoc method should not be chosen.

Our proposition toward this problem is as follows. When decision making

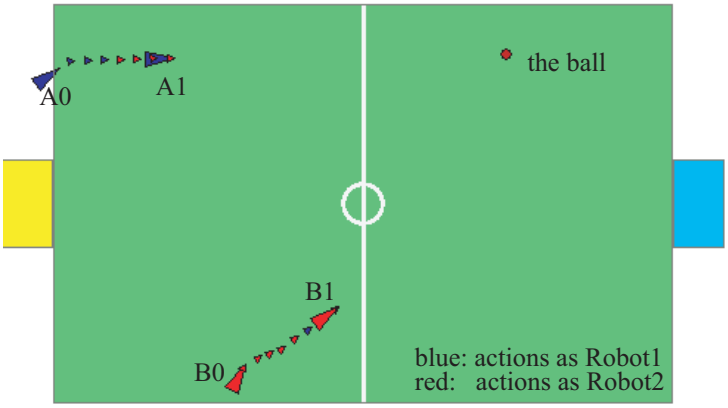


Fig. 6.28: Infinite Loops of Virtual State Transitions

in continuous state space $\mathcal{X}$ has certain advantage over that in discrete space, we should add some algorithms that decide actions in $\mathcal{X}$.

We therefore implement the following additional policies.

(a) **Decision of Shot:** When Robot2 judges whether a score is possible or not by the calculation in $\mathcal{X}$ without the 8D map. When it is possible, Robot2 choose an appropriate kicking action.

(b) **Detection of Reverse State:** When a state is a reverse state in $\mathcal{X}$, a virtual state transition occurs without relation to the 8D map. When a non-reverse state in $\mathcal{X}$ is regarded as a reverse state in the 8D map, there is no information for decision making. In this case, a change of Robot1/2 occurs and new Robot2 chooses *Forward* in order to avoid the deadlock.

Since the state-action map does not consider the case where both of the robots cannot observe the ball, we add another policy for ball search. With this policy, RobotA chooses *TurnLeft* and the other chooses *TurnRight* so as to search the ball.

6.4.8 Evaluation

We evaluate the 8D map with a simulator. In the evaluation, 10,000 initial states are chosen. Each trial starts from one of them. When the number of decisions is over 180 times, the trial is regarded as a failure trial. In simulation, the wall reflects the ball with some extent of reflection.

6.4.9 Effectiveness of Cooperation

At first, we evaluate whether the cooperation by the 8D map is effective or not. We create another state-action map, called the 5D map here, under the consideration of $(x_2, y_2, \theta_2, r, \varphi)$. A robot with this map repeats a set of some walking actions and a kicking action so as to score. The discretization interval of each axis is identical with that of the 8D map.

We try the following cases: 1) two robots with the 8D map, 2) two robots with the 5D map, and 3) one robot with the 5D map. We measure their success rates. Only when the robot(s) can score from a common initial state in all of three cases, the sum of rewards is recorded respectively.

We also evaluate another case where the collision of two robots are considered in simulation. When the distance between two robots is shorter than 50[mm], we regard this state as a collision. The trial is then regarded as a failure trial.

Table 6.14: Efficiency of Cooperation

cases	success rate	average of steps	success rate (stop by collision)
8D Map	97.4[%]	36.3[step]	95.9[%]
5D Map (two robots)	93.8[%]	40.9[step]	70.9[%]
5D Map (one robot)	83.2[%]	50.5[step]	—

Table 6.14 shows the results. When the collisions are not simulated, both the success rate and the number of steps by the 8D map are better than those by the 5D map. 4.6[step] were reduced by the pass behavior. The 8D map is more effective when the collision of the robots is considered. It is interesting that the robots with the 8D map tend to move apart from each other though their collision is not considered in the value iteration. The reason is simply because the robots became a passer and a receiver.

6.4.10 Effectiveness of the Additional Algorithms

We evaluate the additional policies: (a) and (b). In Table 6.15, we show the ability of algorithm (a). It can reduce the number of steps surely though

the reduction of steps is only 0.5[step].

The result in Table 6.16 has two faces. The 8D map is incomplete without algorithm (b). The percentage only with the 8D map, 62.7[%], is worse than any result with the 5D map. On the other hand, if the incompleteness is compensated by algorithm (b), the 8D map can output cooperative behavior with the high percentage. When algorithm (b) is used, there is no failure by the deadlock at reverse states in the 10,000 trials. The 2.6[%] of failures at the use of algorithm (b) happen when the ball stops by the wall, or when the ball cannot be found by the ball search algorithm. In the former cases, robots cannot approach to the ball for fear that they collide with the wall. This is the ill-effect of coarse discretization of robots' positions. We show the behavior of the robots with algorithm (b) in Fig. 6.29 from the identical initial state with that of Fig. 6.28. Though the change of Robot1/2 occurs six times, decision making of the robots is reasonable.

Table 6.15: Efficiency of Shot Decision in $\mathcal{X}$

cases	success rate	average of steps
without (a)	97.3[%]	39.3[step]
with (a)	97.4[%]	38.8[step]

Table 6.16: Avoidance of Deadlock

cases (with (a))	success rate
without (b)	62.7[%]
with (b)	97.4[%]

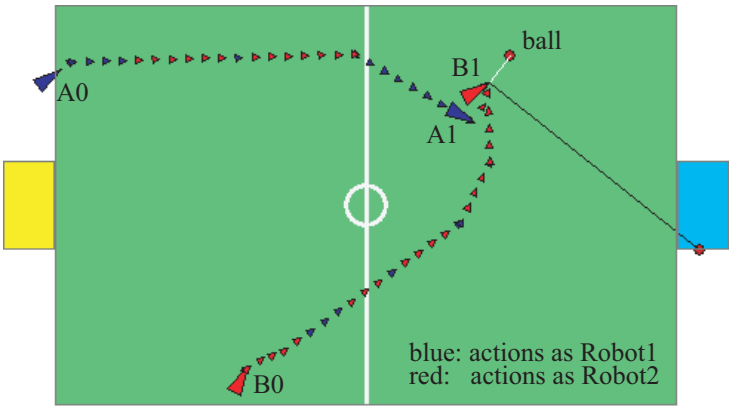


Fig. 6.29: Avoidance of Infinite Loops

6.5 Compression of The 8D Map

The huge state-action map, which contains $610,829,100$ states, is compressed in this section.

6.5.1 Vector Quantization Process

Blocking

At first we cut the state-action map on the axes of x_1, x_2, y_1, and y_2 for creating vectors. In the task of going to ball, efficient VQ maps are obtained when vectors are made by cutting on x-axis and y-axis. We therefore chose that blocking way.

When a state is represented by

$$s_i = ([x_1]_{i_{x1}}, [x_2]_{i_{x2}}, [y_1]_{i_{y1}}, [y_2]_{i_{y2}}, [\theta_1]_{i_{\theta 1}}, [\theta_2]_{i_{\theta 2}}, [r]_{i_r}, [\varphi]_{i_\varphi}),$$

we define that the relation between the state index i of s_i and the indexes of the axes as

$$\begin{aligned} i = {} & 14 \cdot 9^2 \cdot 15^2 \cdot 9 \cdot 19 \cdot i_{x1} \\ & + 9^2 \cdot 15^2 \cdot 9 \cdot 19 \cdot i_{x2} \\ & + 9 \cdot 15^2 \cdot 9 \cdot 19 \cdot i_{y1} \\ & + 15^2 \cdot 9 \cdot 19 \cdot i_{y2} \\ & + 15 \cdot 9 \cdot 19 \cdot i_{\theta 1} \\ & + 9 \cdot 19 \cdot i_{\theta 2} \\ & + 19 \cdot i_r \\ & + i_\varphi \end{aligned}$$

$$\begin{aligned} = {} & 43,630,650 i_{x1} + 3,116,475 i_{x2} + 346,275 i_{y1} \\ & + 38,475 i_{y2} + 2,565 i_{\theta 1} + 171 i_{\theta 2} + 19 i_r + i_\varphi. \end{aligned} \tag{6.20}$$

We then fix a blocking way to

$$\begin{aligned} \begin{pmatrix} i_\nu \\ i_\varepsilon \end{pmatrix} = \begin{pmatrix} \nu(i) \\ \varepsilon(i) \end{pmatrix} &= \begin{pmatrix} \lfloor i/38,475 \rfloor \\ i \% 38,475 \end{pmatrix} \\ &= \begin{pmatrix} 1134 i_{x1} + 81 i_{x2} + 9 i_{y1} + i_{y2} \\ 2565 i_{\theta 1} + 171 i_{\theta 2} + 19 i_r + i_\varphi \end{pmatrix}, \end{aligned} \tag{6.21}$$

where $i_\nu = 0, 1, 2, \ldots, 15,875$ and $i_\varepsilon = 0, 1, 2, \ldots, 38,474$. In this definition, the number of elements in a vector, N_ε, is $38,475$. The number of vectors, N_ν, is then $15,876$.

The size of a VQ map with the above blocking way is calculated by

$$\begin{aligned} L_{\mathrm{VQ}} &= N_\nu \lceil \log_2 N_c \rceil + N_\varepsilon N_c \lceil \log_2 M \rceil \\ &= 15,876 \lceil \log_2 N_c \rceil + 38,475 N_c \lceil \log_2 73 \rceil \end{aligned} \tag{6.22}$$

from Eq. (3.18). We have calculated the size of the state-action map by Eq. (6.19) as 4,275,803,700 [bit]. Since we fix the limit of memory to 10[MB] $= 8 \cdot 10 \cdot 2^{20} = 83,886,080$[bit], the number of representative vectors, N_c, should be less or equal to 310. When $N_c = 310$, $L_{\mathrm{VQ}} = 83,633,634$[bit].

Computation of Distortion Measure

The state-action map contains not only actions in $\mathcal{A}$, but also the virtual actions and a symbol for representing final states. We define the distortion of a change from a virtual action to an action in $\mathcal{A}$ as zero. That is because the virtual actions are not required when the map is used as mentioned in Sec. 6.4.7. The distortion for a change from the symbol of a final state to an action is also defied as zero.

The other distortions are based on Eq. (3.22). Here we have to choose whether all distortions are calculated prior to clustering or not. All of the distortions must be recorded on memory if they are buffered in advance. On the other hand, calculation cost becomes large if each distortion is calculated in the clustering process. The number of combinations of $s \in \mathcal{S}$ and $a \in \mathcal{A}$ is $610,829,100 \cdots 73 = 44,590,524,300$. When each value of distortion is recorded in 2-byte integer, the required size of memory is $89,181,048,600$[byte]≈ 83[GB]. Since this size is small enough to store on a HDD nowadays, we choose to buffer all of the values of distortion measure. The values of distortion are written in $N_\nu = 15,876$ files. Each file relates to a vector and stores all values of distortion $d(s, a)$ whose state s belongs to the vector.

The algorithm for creating the files of distortion can be created with a slight modification of the value iteration algorithm. Every $V(s)$ is calculated on every pair of (s, a) in the value iteration. Therefore, the files can be built by an additional process that computes and records $V^*(s) - V(s)$ for every $a \in \mathcal{A}$.

Clustering Algorithm

We use the Lloyd algorithm for clustering. Though the implemented algorithm is not different from the algorithm in Fig. 3.9, file loading processes should be added to it. At composition of clusters, only one file of a vector is loaded on memory. In the computation of representative vectors, files of vectors in a cluster are loaded for computing a representative vector.

6.5.2 Execution of The VQ Process and Its Result

As mentioned above, $N_c \leq 310$ is a necessary condition for storing a VQ map for scoring task in the flash memory. We choose $N_c = 256$ as the number of representative vectors. In this case, every representative vector index can be represented by an 8[bit] number. This number is convenient for coding. The size of a VQ map with these parameters is

$$\begin{aligned} V_{\mathrm{VQ}} &= N_\nu \lceil \log_2 N_c \rceil + N_\varepsilon N_c \lceil \log_2 M \rceil \\ &= 15,876 \lceil \log_2 256 \rceil + 38,475 \cdot 256 \lceil \log_2 73 \rceil \\ &= 127,008 + 68,947,200 = 69,074,208[\mathrm{bit}] \approx 8.23[\mathrm{MB}] \end{aligned} \tag{6.23}$$

according to Eq. (3.18). This size does not exceed the 10[MB] limit.

The VQ algorithm was executed without any condition of stop. It took 9.4 hours to compute all values of state-value distortion. After that, a VQ map by computation of representative vectors and that by composition of clusters were obtained one after the other. The average time for computation of representative vectors was 45 minutes. That for composition of clusters was then 71 minutes. Therefore, it takes 116 minutes for one pair of iteration.

We recorded every VQ map after the computation of representative vectors and after composition of clusters. Each of them was evaluated by the simulation, and we have obtained the results shown in Fig. 6.30. The data points in this figure start from the simulation result of the VQ map after first computation of representative vectors, and ends on that of the VQ map after 30th computation of representative vectors. We have stopped the computation after the 40th composition of clusters. That is because the VQ maps after 30th computation of representative vectors make the identical result in the simulation.

As shown in this figure, both of success rates and efficiencies were improved until 6th pair of iteration. After that, however, the efficiency

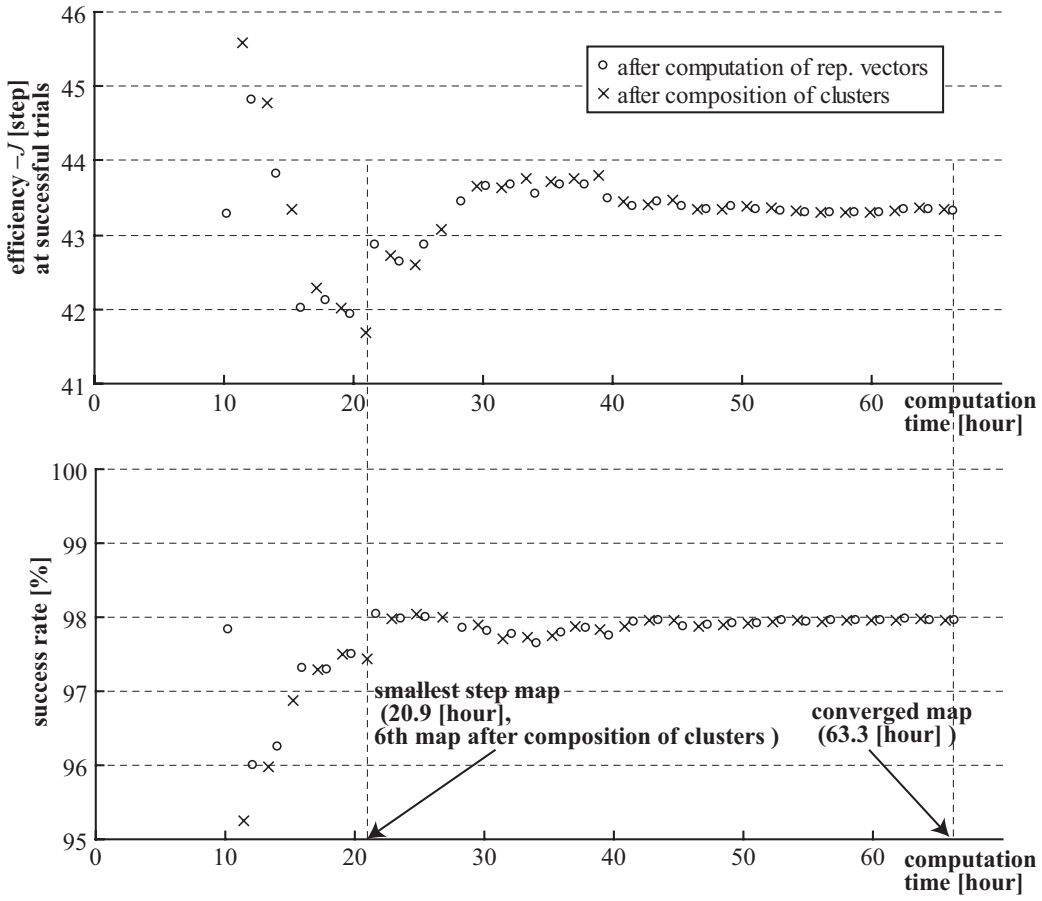


Fig. 6.30: Convergence of VQ Map for Scoring Task

suddenly fell down in exchange for enhancement of the success rate. Finally, the evaluation result converged on 98.0[%] success and $-J = 43.3$[step]. It seems that the enhancement of success rate was more effective to reduce the distortion than the that of efficiency. As a result, the efficiency of the final VQ map was worse than the 6th map after composition of clusters. If the uncompressed map could give assurance of 100[%] success, the graph in this figure would be easier to understand.

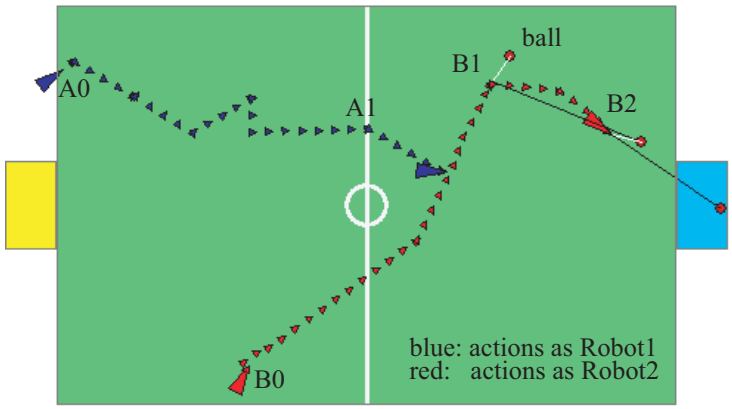
We choose the 6th VQ map after composition of clusters and the VQ map after the convergence as the targets for analysis. They are called the smallest step VQ map and the convergent VQ map hereafter.

Figure 6.31(a) illustrates behavior of the robots with the 6th VQ map. The initial state is identical with that of Fig. 6.28 and Fig. 6.29. The number of kicks increases from one to two because of a subtle change of the policy. We should also pay attention to the bent footstep of RobotA. From this initial state, RobotA does not work in the both cases of the VQ map and the uncompressed map. Since the behavior of RobotA, which is regarded as Robot1, is scarcely related to the task, the part of the state-action map is changed without harm. In Fig. 6.31(b), we give another example. The initial state is the same one with Fig. 6.27. In this example, the number of kicks is reduced from three to two. The VQ map does not always increase the number of kicks.

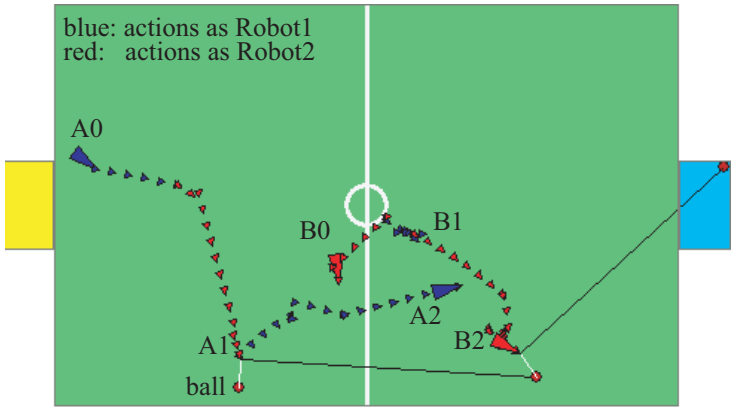
Some parts of the quantization table of the smallest step VQ map is shown in Fig. 6.32 and 6.33. Figure 6.32 represents the distribution map of representative vectors on x_2y_2-space when the pose of Robot1 is fixed. The area where Robot1 exists are marked by a blue frame in each of the eight figures. Roughly speaking[1], these figures illustrate the tendency of motion of Robot2 toward the position of Robot1.

The clustering result in (a) is simple as those of the task of going to ball in Fig. 6.16. It seems that the position of Robot1 is too far from the target goal to cooperate with Robot2 in this case. When Robot1 exists near the goal, on the other hand, the distribution maps of clusters from (b) to (h) are mosaic-like. Not only Robot2's position, but also Robot1's position have important implications to the decision making in those cases.

[1]It is rough statement because each representative vector has information of Robot1's actions.



(a)



(b)

Fig. 6.31: Behavior of Robots with VQ Map

Some parts of the quantization table is shown in Fig. 6.32-6.33. Fig. 6.32 represents the distribution map of representative vectors on x_2y_2-space when the pose of Robot1 is fixed. The area where Robot1 exists are marked by a blue frame in each of the eight figures. Roughly speaking, these figures illustrate the tendency of motion of Robot2 toward the position of Robot1. Figure 6.33 represents the distribution map of representative vectors on x_1y_1-space when the pose of Robot2 is fixed.

The distribution of the representative indexes in Fig. 6.32(a) is simple. The field is divided into some edge parts and some central parts. They are then separated according as the distance from the sky-blue goal. It seems that the position of Robot1 is too far from the target goal to cooperate with Robot2. In this case, Robot2 tries to bring the ball to the goal by itself. The action of Robot2 is based on the relation between the position of the ball and that of the goal. When Robot1 is near the goal, on the other hand, the distribution is mosaic-like as shown in Fig. 6.32(b)-(d). Not only Robot2's position, but also Robot1's position have important meaning to the decision making.

In the case of Fig. 6.33, the space is divided parallel to x-axis. It means that Robot1's position on y-axis, y_1, influences the way of cooperation with Robot2 rather than x_1. It seems that Robot2 changes the direction to approach the ball and to kick it in response to Robot1's position.

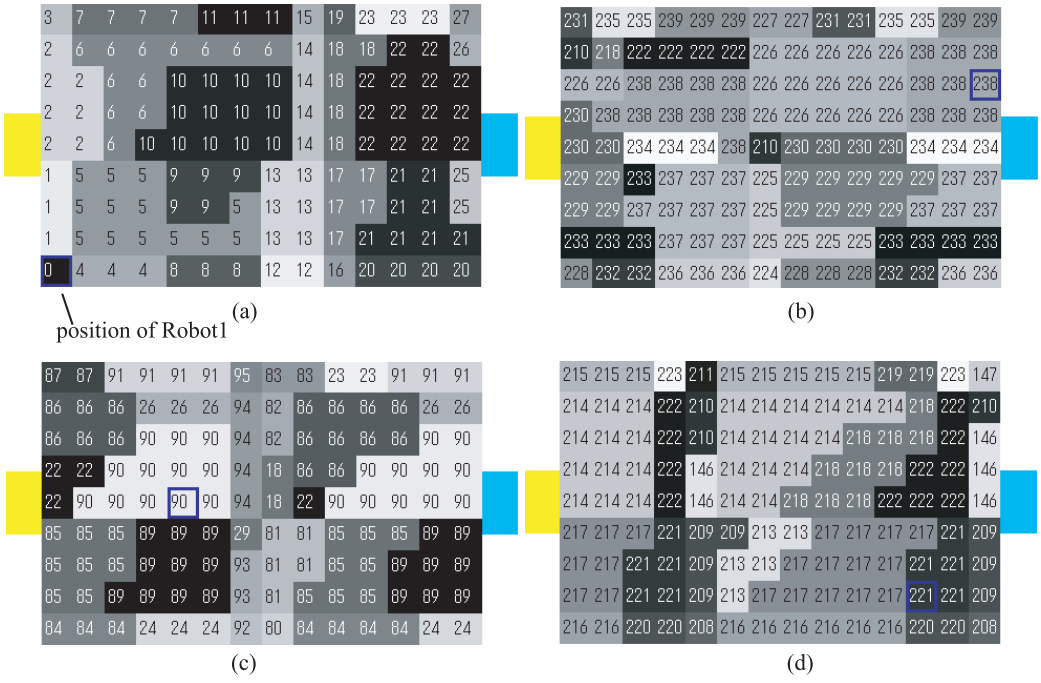


Fig. 6.32: Parts of Quantization Table (Robot1's position is fixed.)

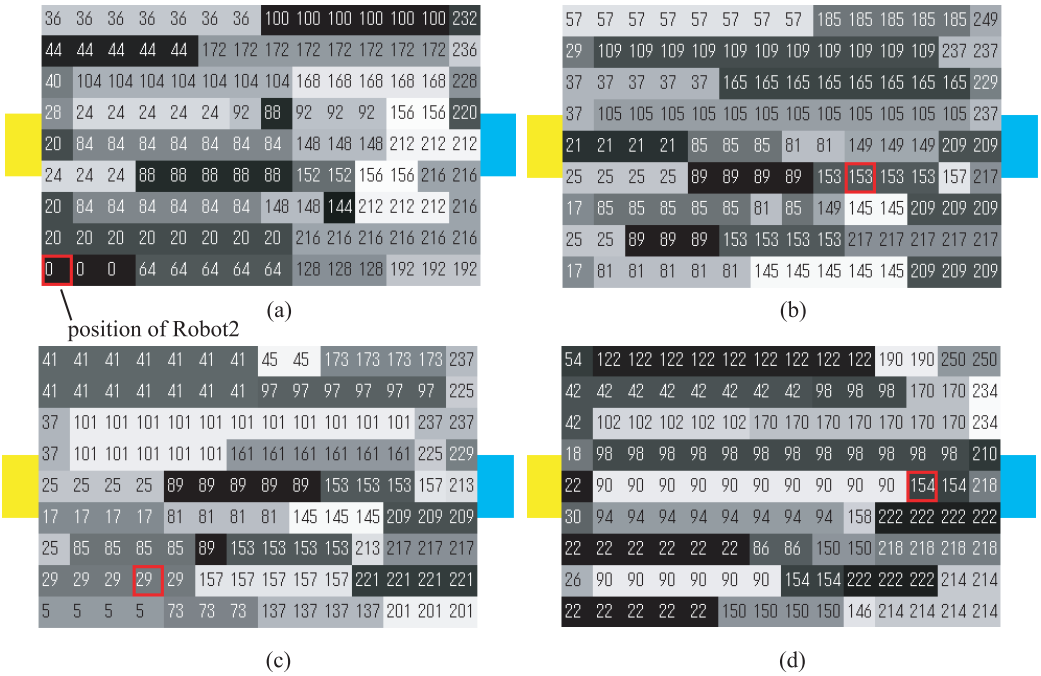


Fig. 6.33: Parts of Quantization Table (Robot2's position is fixed.)

6.6 Comparison of Efficiency

6.6.1 Comparison with the 8D Map and the 5D Map

We evaluate the smallest step VQ map and the convergent VQ map, and compare them to the 8D map and the 5D map. At first, we measure the performance of the VQ maps without consideration of collisions of the robots. The result is compared to the results of the other maps in the left part of Table 6.17. In the case of the smallest step VQ map, performance becomes 2.9[step] worse than that of the 8D map though it is 2.4[step] better than the performance of the 5D map. There is no reduction of the success rate. The success rate of the convergent VQ map is better than the 8D map, while the average of steps increases 4.4[step].

The average of changes in the table denotes the number of the changes of Robot1/2 (not by a map but by the pre-applied decision making in $\mathcal{X}$) per one trial. It indicates the closeness of collaboration between the two robots. As shown in the table, this value of the VQ map is 0.8 smaller than that of the 8D map. Though the robots can make passes with the VQ map, the VQ map tend to make the robots choose solo attack.

The above result quantitatively indicates that the performance decrement occurs through the lossy compression. However, it is significant that the VQ method never makes the map collapse in spite of the high compression ratio (1 : 0.016). The unevenness of the results of the two VQ maps is derived from imperfectness of the 8D map, which does not guarantee 100[%] of success.

Table 6.17: Efficiency of Cooperation with VQ Maps

	success rate	average of steps	average of changes	success rate (stop by collision)
8D Map	97.4[%]	37.5[step]	3.2	95.9[%]
smallest step VQ Map	97.4[%]	40.4[step]	2.4	**95.6**[%]
convergent VQ Map	98.0[%]	41.9[step]	2.4	**96.1**[%]
5D Map (two robots)	93.8[%]	42.8[step]	—	*70.9* [%]

From the right side of Table 6.17, we can verify that a quality of the 8D map can be preserved after the compression. In the right side, the success rates in the case where the collision is considered are illustrated. From these results, we can verify that the VQ map avoids the collision as well as the 8D

map. Though the chance of active cooperation like passes are reduced by the compression, the VQ map can sustain the property of the 8D map.

6.7 Discussion

In this section, the VQ method was applied to state-action maps for two types of task in robot soccer: the going to ball task, and the scoring task.

In the going to ball task, we verified that a VQ map can be used in an actual robots. On the actual robot, the VQ map can achieve equal results with the uncompressed state-action map. As mentioned in Sec. 6.2.6, the noise of measurements makes failure trials increase. Though the VQ map has some broken parts, these faults are hidden. This result is not trivial when we implement policies on actual robots. A fine state-action map is sometimes unnecessary when measurements or motions of robots are not precise. On the other hands, we can obtain an efficient state-action map in theory when the discretization is fine. The VQ method can be a tool for adjustment of the size of a policy under such a practical problem.

The scoring task is the largest scale problem in the tasks examined in this thesis. Though the VQ maps are not evaluated on the actual robots, their size can be reduced under the memory limitation of ERS-210. DP and VQ processes can be finished with 13 days. If there is one more state variable, the computation time will jump to 100 days. By definition, such a basic value iteration cannot deal with an infinite number of state variables. However, we think that the number of state variables will be increased one-by-one. Nowadays, there are two topics, which will accelerate the ability of value iteration, about commercial CPUs.

One of them is parallelization of CPUs. Though it took ten days to obtain the 8D map, this time can be reduced by additional multiprocessing. It would take 20 days if we had not executed the value iteration algorithm on two processes. Some kinds of CPU for personal computers have two cores nowadays and it will increase in future.

The other is the release of 64-bit CPUs for the public. They make coding of DP easy through the explosion of address space. The value iteration algorithm in this paper had to read and write data between RAM and HDD frequently due to the limitation of the memory space. Such a process becomes the cause of slow down and bugs. If we use a 64-bit CPU, the code for value iteration will be simple.

Chapter 7

Total Evaluation and Discussion

In this thesis, the proposed vector quantization method is applied to the following tasks.

- puddle world task
- swinging up of the Acrobot
- decision making of ERS-210 for RoboCup four legged league
  - running to ball task
  - scoring task

In this chapter, the VQ method and some concepts that are used for more than one task are discussed from the viewpoint of their versatility. In Sec. 7.1, space and time costs of our method are evaluated from the results of the above tasks. The ability of the state-value distortion is evaluated in Sec. 7.2.

7.1 Evaluation of Costs of Each Process and VQ Map

7.1.1 Performance of VQ Maps

In Table 7.1, the highest compression ratio that has been obtained in this thesis is illustrated for each task. P_{same} denotes the consistency probability of actions on a pair of adjoining states. We introduce this index in order to show the redundancy of each uncompressed state-action map. It also indicates the inefficiency of discretization when the value is small.

In the case of the puddle world task, we cannot expect a high compression ratio when the discretization is coarse. In that case, the number of bits is cut down before compression. This economization of computing resources at DP, however, reduces the chance of obtaining more efficient policies. In effect, we can obtain more efficient and smaller VQ map, which is compressed from the $N = 20^2$ state-action map, than the $N = 10^2$ state-action map. Moreover, the size of the quantization table is not ignorable when the number of states, N, is small. It is another reason of the low compression ratio. As shown in Table 7.2, the number of vectors are reduced from 20 to 3 by the compression. If only the codebook of the VQ map is considered, the self-compression ratio is 0.15. However, the self-compression ratio of the DLVQ map is just 0.355.

Still, we can verify that the use of memory of VQ is better than the tree structure in Sec. 3.6.2. It is one of the significant results in this thesis.

Table 7.1: Features of Problems and Maps of Applications in This Thesis

			state-action map			VQ map		comp.
task	n	M	N	size [bit]	P_{same}	N	size [bit]	ratio
puddle world	2	4	100	200	0.69	400	142	$1 : 0.71^{\dagger}$
puddle world	2	4	$4.0 \cdot 10^4$	$3.2 \cdot 10^5$	0.96	$1.6 \cdot 10^5$	$1.3 \cdot 10^4$	$1 : 0.16^{\dagger}$
the Acrobot	4	3	$1.5 \cdot 10^7$	$3.0 \cdot 10^7$	0.61	$3.0 \cdot 10^7$	$9.3 \cdot 10^3$	$1 : 0.0031^{\dagger}$
going to ball	5	38	$7.7 \cdot 10^5$	$4.6 \cdot 10^6$	0.39	$7.7 \cdot 10^5$	$6.0 \cdot 10^4$	$1 : 0.013^{\ddagger}$
scoring	8	73	$6.1 \cdot 10^8$	$4.3 \cdot 10^9$	0.34	$6.1 \cdot 10^8$	$6.9 \cdot 10^7$	$1 : 0.016^{\ddagger}$

†: effective compression ratio; ‡: self-compression ratio

n: number of state variables; M: number of actions; N: number of discrete states

On the other hand, much higher compression ratios are obtained in the

Table 7.2: Summary of Discretization

task	N	N_ν	N_ε	N_p	N_c
puddle world (coarse)	400	20	20	—	3
puddle world (fine)	160,000	400	400	—	36
the Acrobot	30,233,088	648	46,656	—	1
going to ball	765,450	630	405	3	8
scoring	610,829,100	15,876	38,475	—	256

tasks of swinging up of the Acrobot and scoring at RoboCup. Though their redundancy index P_{same} are smaller than that of the puddle world task, the obtained compression ratios are better than those of the puddle world task. The similarity between the tasks of the Acrobot and ERS-210 is that

- each problem is defined in higher dimensional state space than that of the puddle world task, and that
- there are some redundant axes.

In the case of the Acrobot, θ_1-axis and θ_2-axis are redundant. In the case of the robot soccer, x-axis and y-axis are redundant. VQ is suitable for reducing this kind of redundancy. Note that the redundancy along an axis does not mean that these axis should be roughly divided on DP. To solve an accurate state-value function, fine discretization of each axis is important. It is not until a state-action map is obtaned that the redundancy on an axis is known.

7.1.2 Computing Complexity for Building

We next summarize the computing complexity of DP and VQ in Table 7.3, and Table 7.4. In the case of the swinging up task of the Acrobot and the scoring task of ERS-210, the amount of memory is calculated from the size of look-up tables for recording the map, the state-value function, probabilities of state-transition, and values of distortion measure. In the other cases, the actual consumption of memory measured by the operating system is written.

When the PNN algorithm is used, its computation amount is as same order as that of DP. The Lloyd algorithm can finish compression within tens percent of the time for DP.

On the other hands, required amount of memory for VQ is ten times or one hundred times as large as that of DP in ever task. The huge amount

Table 7.3: Summary of Costs for DP

task	time (frequency of CPU)	memory
puddle world ($N = 400^2$)	21[min](1.5 GHz)	$4 \cdot 10^6$[bit]
the Acrobot	17[hours] (1.5 GHz)	$2 \cdot 10^9$[bit]
going to ball	25[min] (3.6 GHz)	$3 \cdot 10^6$[bit]
scoring	10[day] (3.2 GHz, 2 process)	$2 \cdot 10^9$[bit]

Table 7.4: Summary of Algorithms and Their Cost for VQ

task	used algorithms	time (frequency of CPU)	memory
puddle world	PNN, Lloyd, VI, DLVQ	33[min]	$9 \cdot 10^6$[bit]
the Acrobot	Lloyd	7[min]-38[min] (1.5GHz)	$8 \cdot 10^9$[bit]
going to ball	PNN with partitioning	4.5[min]-163[min] (3.6GHz)	$1 \cdot 10^8$[bit]
scoring	Lloyd	20.9[hour]-63.3[hour] (3.2GHz)	$9 \cdot 10^{10}$[bit]

VI: value iteration after vector quantization

of memory is used for storing the values of distortion. Though this use of memory is effective for reducing the computing time of VQ, it will be a difficulty for compressing huger state-action maps. We should find a way to balance the amount of memory and the time for computing.

7.1.3 Double Layered Vector Quantization

We have introduced vector quantization of VQ maps and this method has applied to the puddle world task and the going to goal task. In the case of the puddle world task, the sizes of VQ maps can be reduced to 89[%]–44[%] sizes. The larger the original state-action map is, the larger compression ratio can be obtained. On the other hand, enhancement of the compression ratio between the VQ map and the DLVQ map is only $1 : 0.87$ in the going to ball task. Though the number of states at the going to ball task are larger than the numbers of states for the puddle world task, this enhancement is not large. The number of actions for the going to ball task is 38, while that of the puddle world task is 4. It seems that the existence of many actions prevents the DLVQ algorithm for compressing the VQ map with high compression ratio since the DLVQ algorithm is a lossless compression method.

7.1.4 Entropy Function

It seems that evaluation by the entropy function toward every axis helps to find good ways of blocking. As shown in Table 4.1 and Table 6.5,

the entropy function chooses suitable directions to cut the state-action maps.

Not only the evaluation toward each axis, but also ways of blocking are evaluated by the entropy function. The results are shown in Fig. 4.11, Fig. 6.14, and Table 6.6-6.7. In the puddle world task, the evaluation is not clearer than the evaluation toward each axis. We can marginally expect a high compression ratio from some pairs of (ν, ε) with small $\mathcal{H}$ and large N_ε. On the other hand, in the case of the going to ball task, we can estimate suitable ways of blocking with the entropy function.

We think that there is still room for improvement on the use of the entropy function. Moreover, if we want to find suitable blocking ways with higher accuracy, use of the state-value distortion should be also tried. However, it will be difficult to find a better way than the entropy evaluation due to enlargement of computing time. As mentioned before, moreover, the acutual appropriateness of blocking is unknown until a VQ map is obtained.

7.2 Evaluation of State-Value Distortion

The state-value distortion is proposed in this thesis and it can be applied to all of the tasks in this thesis. However, there are some possible definitions of distortion measure. Here we evaluate the state-value distortion with some definitions.

7.2.1 Other Distortion Measures

Difference Count Distortion

One of them counts the number of different actions between two maps. The distortion in a state is defined as

$$d'^{\pi}(s,a) = \begin{cases} 0 & (\text{if } \pi(s) = a) \\ 1 & (\text{otherwise}) \end{cases}, \tag{7.1}$$

when the action at $s \in \mathcal{S}$ is changed from $\pi(s)$ to $a \in \mathcal{A}$. The distortion measure between a vector and a representative vector can be written as

$$D'^{\pi}_{\text{vector}}(\boldsymbol{v}, \boldsymbol{c}) = \sum_{i=0}^{N_\varepsilon - 1} \begin{cases} 0 & (v_i = c_i) \\ 1 & (v_i \neq c_i) \end{cases}, \tag{7.2}$$

where $\boldsymbol{v} = (v_0, v_1, \ldots, v_{N_\varepsilon - 1})$ and $\boldsymbol{c} = (c_0, c_1, \ldots, c_{N_\varepsilon - 1})$. D'^{π}_{vector} can be called a difference count distortion.

Control Input Distortion

The other is defined in consideration of displacement of actions. When the displacement of action $a \in \mathcal{A}$ on xy-plane is represented by $\boldsymbol{x}_a$, the distortion at $s \in \mathcal{S}$ is defined as

$$d''^{\pi}(s,a) = |\boldsymbol{x}_{\pi(s)} - \boldsymbol{x}_a|. \tag{7.3}$$

The distortion between a vector and a representative vector is then defined as

$$D''^{\pi}_{\text{vector}}(\boldsymbol{v}, \boldsymbol{c}) = \sum_{i=0}^{N_\varepsilon - 1} |\boldsymbol{x}_{v_i} - \boldsymbol{x}_{c_i}|. \tag{7.4}$$

$D''^{\pi}_{\text{vector}}$ can be called a control input distortion because it is measured in parameter space of control input.

7.2.2 Comparison on The Puddle World Task

We compare the state-value distortion, the difference count distortion, and the control input distortion on the puddle world task. VQ maps are created with every distortion measure. The PNN algorithm, the Lloyd algorithm, and the value iteration algorithm for VQ maps are applied to creation of each VQ map. After that, each VQ map is evaluated by the simulation.

Figure 7.1 shows the result. The state-action maps with $N = 10^2, 20^2, 40^2, 100^2, 200^2, 400^2$ are compressed and their sizes and efficiencies are illustrated from Fig. (a) to (f) respectively. The numbers of clusters are chosen from $N_c = 1, 2, \ldots, 10, 2^4, 2^5, 2^6, \ldots$ Each horizontal axis and vertical axis indicates size and efficiency respectively.

As shown in this figure, the advantage of the state-value distortion is clear. In Fig. (b)-(f), the state-value distortion endures higher ratio compression than the others. The efficiency of VQ maps created with the state-value distortion are then equal or better than the other maps in Fig. (a) and (c)-(f). The difference of ability is not prominent between the difference count distortion and the control input distortion.

Figure 7.2 illustrates an example of the cases where the state-value distortion is advantageous. The map in (a) is compressed to a VQ map in (b) with the state-value distortion. The VQ map obtained by the other definitions of distortion is coincidentally identical and it is shown in (c). The clustering results of the VQ maps in (b) and (c) are illustrated in (d) and (e) respectively.

The major difference between (d) and (e) is the clustering result of $\mathcal{K}_1$. Even though the alignment of actions in the vectors from $\boldsymbol{v}_2$ to $\boldsymbol{v}_{17}$ is opposite of the alignment in the vectors from $\boldsymbol{v}_{74}$ to $\boldsymbol{v}_{89}$, the state-value distortion D^{π}_{vector} makes them belong to the same cluster $\mathcal{K}_1$. Since D^{π}_{vector} can consider the equality of a_{right} and a_{up} in this area, they can belong to the same cluster. On the other hands, the clustering result in (e) by the other definitions of distortion is more superficial than that of the state-value distortion.

Since $\mathcal{K}_1$ obtained by the state-value distortion covers large areas of the puddle world, the other clusters can be composed finely. As a result, the efficiency of the VQ map in (b) is better than the VQ map in (c), which is obtained by the other definitions.

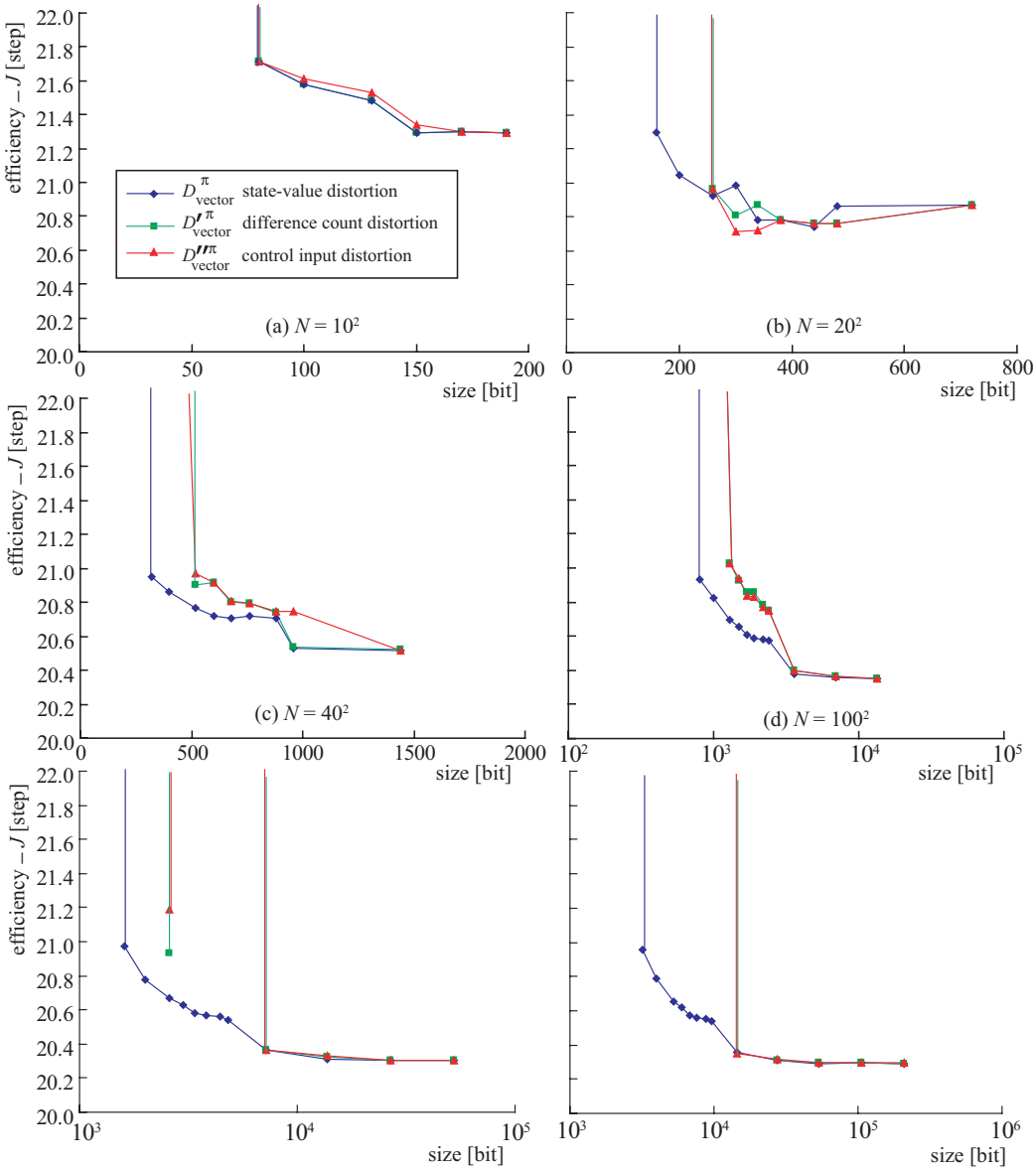


Fig. 7.1: Evaluation of the State-Value Distortion

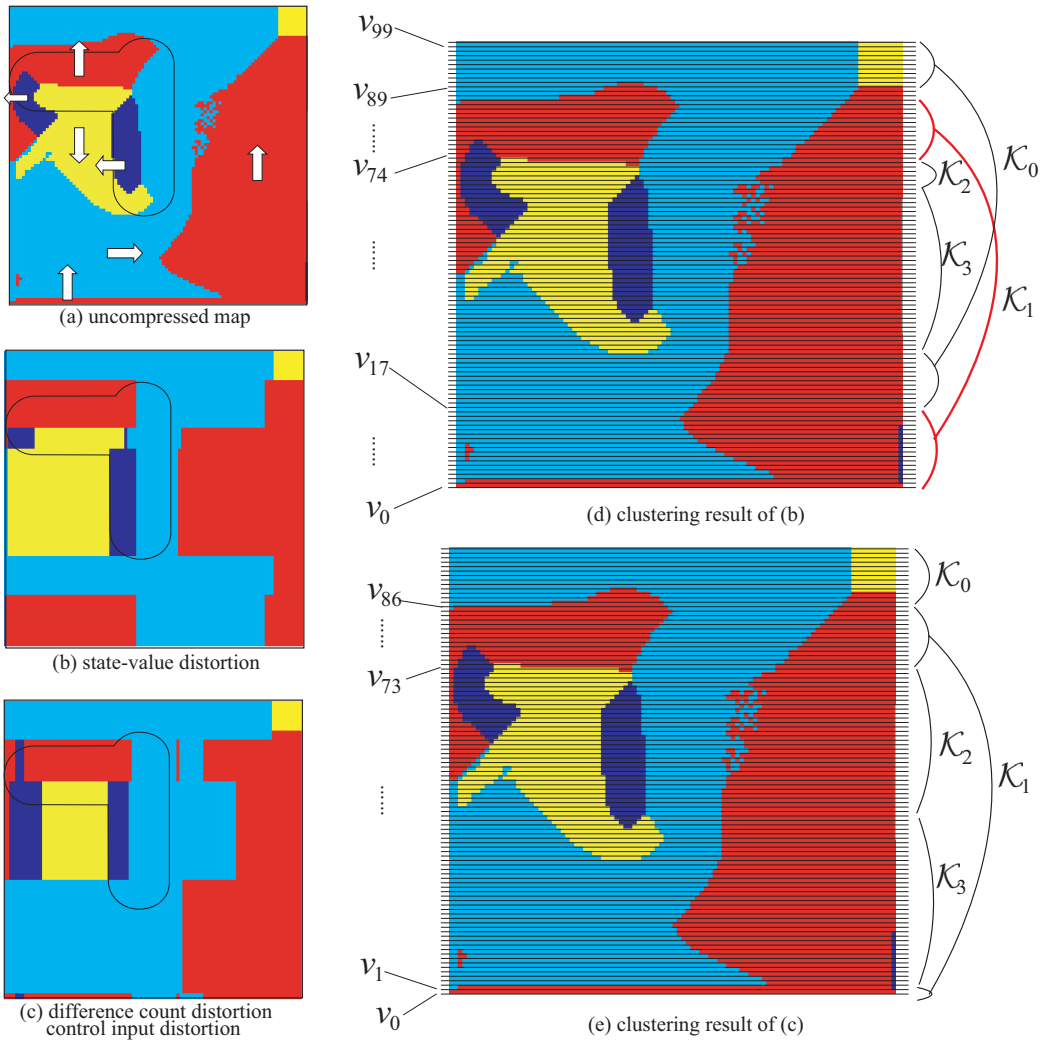


Fig. 7.2: VQ Maps Obtained by the Three Kinds of Distortion Measure ($N = 100^2, N_c = 4$)

7.2.3 Comparison on The Acrobot

We also create VQ maps using the difference count distortion and the control input distortion. In the case of this task, Eq. (7.1) and Eq. (7.3) should be rewritten as

$$d'^{\pi}(s,\tau) = \begin{cases} 0 \text{ (if } \pi(s) = \tau) \\ 1 \text{ (otherwise)} \end{cases} \quad , \text{ and} \tag{7.5}$$

$$d''^{\pi}(s,\tau) = |\pi(s) - \tau| \tag{7.6}$$

respectively. In the both cases, the distortion in every state whose value is worse than -100[s] is regarded as zero toward any change of torque. This condition is identical with that on the creating process of the VQ map in Fig. 5.12.

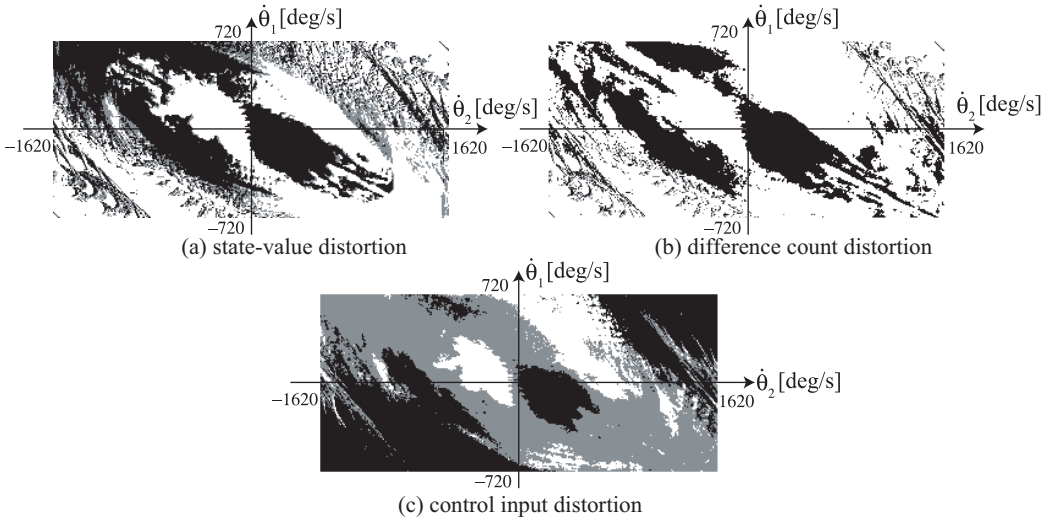


Fig. 7.3: VQ Map with $N_c = 1$ Obtained by Three Definitions of Distortion Measure

The obtained VQ maps with $N_c = 1$ are shown in Fig. 7.3. The difference between (a) and (c) is eye-catching. In the case of the control input distortion, $\tau = 0$ tends to be the average of torques at states that are related to an element of a representative vector. Therefore, the gray part is much larger than the other VQ maps. In the case of the VQ map in (b), on the contrary, $\tau = 0$ is hardly chosen as elements of representative vectors by the difference count distortion.

Not only these three VQ maps, but also VQ maps with other N_c are evaluated by the simulation. The results are shown in Table 7.5. The average time is illustated in Fig. 7.4. As shown in the table and the graph, the VQ maps with the state-value distortion mark the best average time in every N_c except $N_c = 4$. The difference count distortion can give the best result in the average of $N_c = 32, 4$ and in the worst time of some VQ maps. However, the averages in $N_c = 256$ and $N_c = 128$ are inscrutably worse than those of the state-value distortion. The control input distortion seems to be unsuitable for this task. Since representative vectors have many elements of $\tau = 0$, the control policies provided by the maps seem to be indistinct.

Table 7.5: Comparison of VQ Maps Obtained by The Distortion Measures

	average [s]			worst [s]		
	state-	difference	control	state-	difference	control
N_c	value	count	input	value	count	input
256	**11.4**	12.4	12.4	**45.6**	48.7	60.3
128	**11.3**	12.3	12.4	60.4	**42.2**	52.3
64	**11.8**	11.9	12.2	**51.6**	51.9	56.3
32	**12.4**	**12.4**	12.9	63.5	59.5	**59.2**
16	**12.8**	13.0	13.8	78.7	**56.6**	57.5
8	**13.4**	13.5	14.3	**55.5**	59.8	61.8
4	13.7	**13.6**	15.1	71.3	**54.5**	106.2
2	**13.7**	14.0	16.0	**62.1**	75.3	117.6
1	**13.7**	14.1	17.9	85.0	**59.7**	failure

7.2.4 Comparison on The Scoring Task

In the case of the scoring task, we cannot define the control input distortion because the state variables $(x_1, y_1, \theta_1, x_2, y_2, \theta_2, r, \varphi)$ have different characteristics. If we define a generalized distance, definition of a distortion measure can be possible. However, there is no basis of the definition of distance.

Therefore, VQ maps only with the difference count distortion are created and compared here. In the same way with Fig. 6.30, the efficiency and success rate of each VQ map in the Lloyd iteration is evaluated. The evaluation result is shown in Fig. 7.5. Convergence was recognized after the 44th computation of representative vectors. The total time for convergence was 43.7[hour].

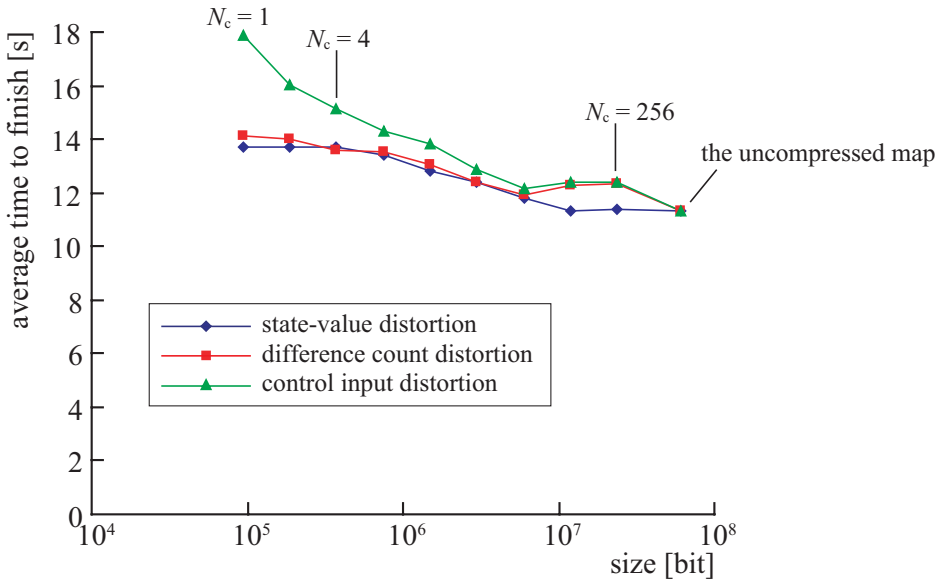


Fig. 7.4: Comparison of VQ Maps Obtained by The Distortion Measures (graph)

From the comparison between Fig. 6.30 and Fig. 7.5, we notice that the converged map by the difference count is obviously more efficient than the converged map by the state-value distortion. This result is not expected one for us. The reason is discussed later.

The smallest step map was obtained after 6th composition of clusters. This smallest step map and the converged map obtained by the difference count distortion are compared with those obtained by the state-value distortion. The simulation result is shown in Table 7.6. As shown in the table, each map has different drawbacks and advantages. We cannot make a sweeping judgment about whether each map is good or bad.

However, we notice that the VQ maps with the difference count distortion are inherited the property of the 8D map. That is because the average numbers of changes of the robots are larger than those of the VQ maps that are obtained by the state-value function. The VQ maps with the difference count distortion make the robots attempt passes more frequently than the other VQ maps.

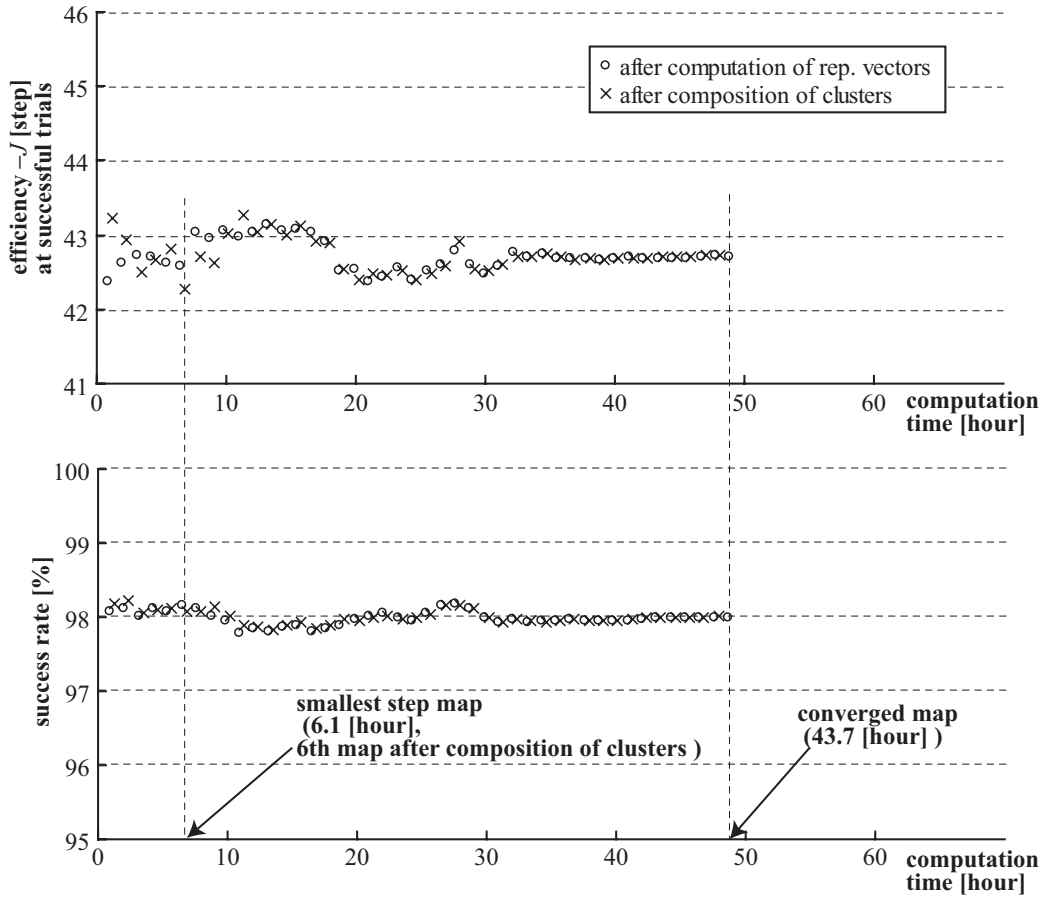


Fig. 7.5: Convergence of VQ Map for Scoring Task by Difference Count Distortion

Table 7.6: Efficiency of VQ Maps Obtained by the Definitions of Distortion Measure

distortion measure and maps		success rate	avg. of steps	avg. of changes	success rate (stop by collision)
state-value	smallest step VQ map	97.4[%]	**41.5**[step]	2.4	95.6[%]
	convergent VQ map	98.0[%]	43.0[step]	2.4	**96.1**[%]
difference count	smallest step VQ map	**98.1**[%]	42.0[step]	**3.0**	95.7[%]
	convergent VQ map	98.0[%]	42.4[step]	**2.9**	95.5[%]

To put it the other way around, the state-value distortion changed the 8D map more radically than the difference count distortion. This phenomenon is also shown in Fig. 7.6. To reduce the number of steps, the state-value distortion chose to reduce the frequency of passes. Though it was not effective on this task, the numbers of changes in the table suggest that VQ with the state-value distortion can cause the qualitative change of a state-action map in accordance with required compression ratios.

7.2.5 Discussion

The result of evaluation in each task can be summarized as Table 7.7. The state-value distortion and the difference count distortion can be used for the three tasks. On the other hand, the control input distortion has some problems. We think that this difference occurs because the control input distortion considers not the property of tasks but that of the motion of robots.

Table 7.7: Summary of Comparison Results

	state-value	difference count	control input
puddle world	better	worse	worse
the Acrobot	better	even	worse
scoring	even	even	—

From this point of view, the property of each task can by taken into the account by the difference count distortion through the contents of each state-action map. Therefore, the compression with this distortion measure is possible in each task. However, this distortion measure cannot evaluate the influence of changes of actions, while the state-value distortion evaluates them. This difference causes the difference of efficiency especially in the case of the puddle world task and the task of the Acrobot.

Chapter 8

Conclusion and Future Work

8.1 Conclusion

In this thesis, we have introduced a concept of compression of policies for decision making problems and control problems so as to implement reflexive policies on robots with small amount of memory and high efficiency.

The vector quantization (VQ) method is used for the compression. In the compression, dynamic programming (DP) result is reused as the state-value distortion. Therefore, our method belongs not only to information processing, but also to control or decision making problems. Our method is evaluated by the puddle world tasks, the swinging up task of the Acrobot, and the tasks of robot soccer.

We have obtained the following knowledge through this study. From them, we conclude that our methodology is applicable for the above tasks, and outperforms current competitive and popular methods.

• Versatility for Different Types of Tasks

The VQ method can be applied to the puddle world task, the swinging up task of the Acrobot, and the tasks in RoboCup successfully. In the puddle world task and the swinging up task, we have verified that a coarse state-action map is larger and less efficient than some VQ maps created from a fine state-action map. The highest effective compression ratios are $1 : 0.16$ and $1 : 0.0031$ in the puddle world task and the swinging up task respectively. In the going to ball task, a VQ map whose self-compression ratio is $1 : 0.015$ is obtained. It has been as efficient as the uncompressed state-action map on the actual ERS-210. In the scoring task, the self-compression ratio is

$1 : 0.016$, and the 8D map, whose size is $4.3 \cdot 10^9$[bit], can be compressed to a $6.9 \cdot 10^7$[bit] VQ map. The size is smaller than the size of ERS-210's flash memory.

• Superior Memory Economization Ability to Other Methods

As the other methods, we have implemented a tile coding algorithm, an interpolation algorithm and a tree-structure algorithm. They are compared to our method on the puddle world task. As mentioned in Sec. 2.5, these techniques are not effective when DP can create a state-action map within a given amount of memory.

On the other hand, we can obtain some small and effective policies by the tree-structure algorithm. A binary-tree policy only with 1913 discrete states (leaves) is superior (12,240[bit], 20.31[step]) to a state-action map with $N = 100^2 = 10,000$. However, we can also obtain a smaller DLVQ map (10.568[bit], 20.31[step]) than the binary-tree policy. Though a binary-tree can represent a policy with small number of discrete states, a large amount of memory is used for representing its tree-structure.

Moreover, the size of binary-tree policies that are superior to state-action maps are limited in the range from 2.7k[bit] to 12.0k[bit], while those of VQ maps are distributed from 150[bit] to 100k[bit]. We think that this difference is caused by the versatility of vector quantization as a universal compression method. In this stage, we should conclude that application of VQ is appropriate for the policy implementation problem.

8.2 Future Work

8.2.1 Reduction of Cost in Creating Process of Policies

In this thesis, we have chosen the most secure approach, which is value iteration with a look-up table, on the off-line process. As we have mentioned repeatedly, it is difficult to develop more efficient DP methods than the value iteration algorithm. However, if the VQ method is applied inversely to creating a policy, there is a possibility that we can reduce time and space complexity of dynamic programming. The word *inversely* means that a method starts from a state in which a temporary VQ map exists. The VQ map can be enhanced by changes of its codebook and quantization table with efficiency evaluation of the VQ map. In this case, use of a function approximation method should be looked into for reducing computing complexity. In this case, however, we should expect that such a method does not have versatility.

For reinforcement learning, use of the VQ map is more realistic. There are reinforcement learning methods that utilize *actor-critic* structure [Kimura, 1998]. In an actor-critic structure, a state-value function and a policy are separated and they are enhanced with their interactions and experiences of an agent. A VQ map can be a possible format for the policy.

8.2.2 for Augmented Decision Making Problems

In this thesis, a decision making problem is handled in a limited number of state variables. On the other hand, the number of state variables exist infinitely when we parameterize various environments in which robots labor on tasks. Though it is impossible to deal with the infinite number of state variables, we can connect our study to some typical problems in robotics.

Consideration of Parametric Errors

Then, there will be some cases where parameters should be regarded as state variables. For example, the parameters of the Acrobot shown in Table 5.2 cannot be always measured precisely. Though the VQ maps for the swinging up task are robust against errors of parameters as mentioned in Sec. A.3, that robustness cannot be always expected in all control problems.

There are two ways to deal with this problem. One of them is to consider the disturbance of state-transitions by some margins of parametric errors

when the state-transition probabilities are calculated. The other one is to build a state-action map for each part in an area of parameter space. If the calculations can be finished in feasible time, we can obtain a policy that can be applied to all of the set of parameters in the area. In this case, the parameters should be learned at on-line since one suitable state-action map must be chosen.

We have interest in the latter though time complexity for creating state-action maps is large. That is because the VQ method can compress the state-action maps not respectively but jointly. When a part of a policy barely changes within a range of parameter change, the parts of the state-action maps obtained within the range of parameters can be represented by one representative vector. Therefore, we can expect efficient memory use for representing a policy for various sets of parameters.

Uncertainty of State Variables

Autonomous robots must know state variables by themselves. Then the obtained state variables contain some extent of uncertainty. Barraquand *et al.* and LaValle have proposed a method for dealing with the uncertainty with identical formulation of Markov decision processes [Barraquand, 1995; LaValle, 2000]. In this method, the state space is augmented by additional state variables that represent how the state variables are uncertain. If the state-action map can be obtained in the augmented state-space, the VQ method can be applied to it without any modification.

Roy *et al.* have proposed *the coastal navigation method* in which a state-action map for navigation of a mobile robot is solved by value iteration with an additional state variable [Roy, 1999]. The additional state variable represents the uncertainty of the position and orientation of the robot. In this study, range sensors were used to measure the distance from walls in an indoor environment and the results were used for self-localization. Since the state-transitions of uncertainty reduction by the range sensors could be estimated, the value iteration could build an effective state-action map. This map makes a robot move along walls so that the robot can localize itself easily. We have also studied this attempt with the going to ball task in early stage of this study [Fukase, 2003]. However, the effectiveness of uncertainty consideration was unclear because it is difficult to estimate state-transitions of the uncertainty by the observations with the camera of ERS-210.

We have also studied another kind of decision making methods under

the uncertainty [Ueda, 2003; Ueda, 2005]. In the method, DP is executed without consideration of uncertainty. On the other hand, the uncertainty is represented by a probability distribution in the state-space. The decision making method chooses an action that is expected to maximize the value from the probability distribution and the state-value function. The state-action map is also used for the calculations. To use this method, we have to compress not only the state-action map, but also the state-value function. Alternatively, we need to propose a method that can choose an appropriate action from a probability distribution and a VQ map.

Consideration of Other Movable/Moving Objects

The state-action maps created in this thesis have a common problem: they do not deal with a movable object or a moving object whose state variables are not considered. The movable objects mean chairs, boxes, or other static obstacles for instance. The moving objects will be robots or persons in the environment. Though they will be eliminated in the case of the Acrobot, other robots always exist in the games of RoboCup. Moreover, four of them are opponents.

However, we have used state-action maps in actual games simply because they are useful. In the case of the going to ball task, for example, a robot in the task never avoids the other robots; otherwise an opponent robot will keep the ball. It is a case where the number of state variables is extremely limited.

On the contrary, a robot never collides with a human being or another robot when it is working in an office environment. Since this is an important function for mobile robots, various collision avoidance methods have been proposed. In this case, we should not create a state-action map in state-space that is augmented by some state variables for representing poses of others. Since the motion of the others is unpredictable, an obtained state-action map will not be efficient for its size.

Though the above cases are utterly different to each other, state-action maps that do not consider the other objects are important in the both cases. In the latter case, a robot can restart the task with a state-action map after any other method is used for collision avoidance. Since the state-action map can regard any state as an initial state, the collision avoidance method can concentrate the robot on going to the other side of an obstacle. On that point, state-action maps in partial state space of the environment have high affinity with other decision making policies.

As a matter of course, combinations of state-action maps are a possible way to enable robots to engage in various tasks. In this case, the VQ method is also useful for storing various state-action maps as mentioned in Sec. 1.3.4.

8.2.3 Minimum Description Length Principle for Decision Making

To formulate the relation between time complexity, space complexity, and the efficiency of decision making policies and control policies will be the ultimate objective as an extension of this thesis. In fact, we have limited the target for the study to reflexive policies in this thesis.

This formulation will be a version of the minimum description length (MDL) principle [Rissanen, 1999] for control and decision making problems. In the MDP principle, the trade-off between the size of data and its discrimination capability is discussed. If the response time of a policy is added to the trade-off analysis, we can compare not only reflexive policies but also outputs of search methods.

Appendix A

Further Note

A.1 Coding of Eq. (2.17)

We had to compute Eq. (2.17) as accurate as possible so that we evaluate the methods accurately. When s and s' are following rectangles:

$$s = \{(x, y)|x_1 \leq x < x_2, y_1 \leq y < y_2\}, \text{and} \tag{A.1}$$
$$s' = \{(x', y')|x'_1 \leq x' < x'_2, y'_1 \leq y' < y'_2\}, \tag{A.2}$$

Eq. (2.17) is modified as follows.

$$\begin{aligned}
\mathcal{P}^a_{ss'} &= \frac{\int_{\boldsymbol{x}\in s}\int_{\boldsymbol{x}'\in s'} \frac{5000}{\pi}\exp\left(-5000|\boldsymbol{x}' - \boldsymbol{x} - \boldsymbol{u}|^2\right) d\boldsymbol{x}' d\boldsymbol{x}}{\int_{\boldsymbol{x}\in s} d\boldsymbol{x}} \\
&= \frac{5000}{\pi(x_2 - x_1)(y_2 - y_1)}\cdot \\
&\quad \int_{x_1}^{x_2}\int_{y_1}^{y_2}\int_{x'_1}^{x'_2}\int_{y'_1}^{y'_2} e^{-5000(x'-x-u_x)^2} e^{-5000(y'-y-u_y)^2} dy'dx'dydx \\
&= \frac{5000}{\pi(x_2 - x_1)(y_2 - y_1)}\cdot \\
&\quad \left(\int_{x_1}^{x_2}\int_{x'_1}^{x'_2} e^{-5000(x'-x-u_x)^2} dx'dx\right)\left(\int_{y_1}^{y_2}\int_{y'_1}^{y'_2} e^{-5000(y'-y-u_y)^2} dy'dy\right)
\end{aligned}$$

$$
\begin{aligned}
&= \frac{1}{20000\pi(x_2 - x_1)(y_2 - y_1)} \Big\{ \\
&- e^{-5000(x_1 - x_1' + u_x)^2} - 50\sqrt{2\pi}(x_1 - x_1' + u_x) f_e[50\sqrt{2}(x_1 - x_1' + u_x)] \\
&+ e^{-5000(x_2 - x_1' + u_x)^2} + 50\sqrt{2\pi}(x_2 - x_1' + u_x) f_e[50\sqrt{2}(x_2 - x_1' + u_x)] \\
&+ e^{-5000(x_1 - x_2' + u_x)^2} + 50\sqrt{2\pi}(x_1 - x_2' + u_x) f_e[50\sqrt{2}(x_1 - x_2' + u_x)] \\
&- e^{-5000(x_2 - x_2' + u_x)^2} - 50\sqrt{2\pi}(x_2 - x_2' + u_x) f_e[50\sqrt{2}(x_2 - x_2' + u_x)] \\
\Big\} \Big\{ & \\
&- e^{-5000(y_1 - y_1' + u_y)^2} - 50\sqrt{2\pi}(y_1 - y_1' + u_y) f_e[50\sqrt{2}(y_1 - y_1' + u_y)] \\
&+ e^{-5000(y_2 - y_1' + u_y)^2} + 50\sqrt{2\pi}(y_2 - y_1' + u_y) f_e[50\sqrt{2}(y_2 - y_1' + u_y)] \\
&+ e^{-5000(y_1 - y_2' + u_y)^2} + 50\sqrt{2\pi}(y_1 - y_2' + u_y) f_e[50\sqrt{2}(y_1 - y_2' + u_y)] \\
&- e^{-5000(y_2 - y_2' + u_y)^2} - 50\sqrt{2\pi}(y_2 - y_2' + u_y) f_e[50\sqrt{2}(y_2 - y_2' + u_y)] \\
\Big\}, &
\end{aligned}
\tag{A.3}
$$

$$
\text{where } f_e(z) = \frac{2}{\sqrt{\pi}} \int_0^z e^{-z^2} dz. \tag{A.4}
$$

f_e is called the error function, shown in books of statistics as a look-up table. We implement the look-up table in the code that calculate $\mathcal{P}^a_{ss'}$ directly from Eq. A.3.

A.2 Consideration of Collision

To consider the collisions, we implement an algorithm that allocates $\mathcal{P}^a_{ss'}$ to other states when s' is out of the world.

1. Obtain the intersecting point $\boldsymbol{p}_0$ of a wall and a line segment whose end-points are the center points of s and s'.

2. Assume another line segment. Its length is $1/\sqrt{N}$, its median point is $\boldsymbol{p}_0$, and it is parallel with the wall. $1/\sqrt{N}$ is the width of an edge of a discrete state. Its end-points are named $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$.

3. A state that is the nearest to $\boldsymbol{p}_i$ in $\mathcal{S}$ is named s'_i $(i = 1, 2)$.

4. If $s'_1 \neq s'_2$, there is a point that divides s'_1 and s'_2 on the wall. This point is named $\boldsymbol{q}_{12}$.

5. The probability $\mathcal{P}^a_{ss'}$ is shared by s'_1 and s'_2 based on the following equations:

$$\mathcal{P}^a_{ss'_1\text{collide}} = \sqrt{N}|\boldsymbol{p}_1 - \boldsymbol{q}_{12}|\mathcal{P}^a_{ss'}, \text{ and} \tag{A.5}$$

$$\mathcal{P}^a_{ss'_2\text{collide}} = \sqrt{N}|\boldsymbol{p}_2 - \boldsymbol{q}_{12}|\mathcal{P}^a_{ss'}. \tag{A.6}$$

These probabilities are added to $\mathcal{P}^a_{ss'_1}$ and $\mathcal{P}^a_{ss'_2}$ that are computed without collisions. Figure A.1 illustrates an example of the relation between the symbols.

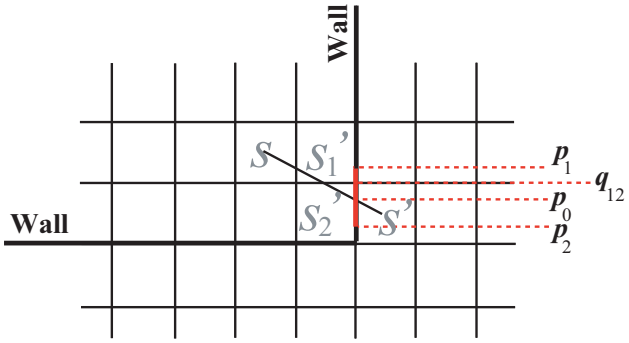


Fig. A.1: Probability share

A.3 Robustness of Maps for the Acrobot toward Errors of Parameters

Here we evaluate the robustness of the state-action maps and the VQ maps in Chapter 5. We change values of the parameters of the Acrobot and use the maps created in Chapter 5. Since the values are different from those when the maps are created, the performance will be lost.

Table A.1: Performance toward Parameter Error ($m_1 = 1.1$[kg])

N_c	average [s]	loss[s]	worst [s]	loss[s]
normal	11.4	0.1	41.8	5.7
256	11.5	0.1	40.9	-4.7
128	11.3	0.0	44.6	-15.8
64	11.7	-0.1	49.0	-2.6
32	12.4	0.0	65.3	1.8
16	12.9	0.1	55.5	-23.2
8	13.5	0.1	71.3	15.8
4	13.8	0.1	81.2	9.9
2	13.7	0.0	72.9	10.8
1	13.9	0.2	68.1	-16.9
minimum map	17.1	0.1	failure	—

Table A.2: Performance toward Parameter Error ($m_2 = 1.1$[kg])

N_c	average [s]	loss[s]	worst [s]	loss[s]
normal	12.1	0.7	40.5	1.1
256	12.0	0.6	47.7	7.9
128	11.7	0.4	44.1	-1.8
64	12.2	0.4	60.0	-6.2
32	12.8	0.3	52.1	-7.1
16	13.2	0.3	56.5	-11.2
8	13.9	0.4	82.1	23.6
4	14.2	0.6	69.0	13.7
2	14.1	0.4	75.1	-14.4
1	14.4	0.7	68.7	-23.1
minimum map	17.7	0.9	failure	—

Table A.3: Performance toward Parameter Error ($(\ell_1, \ell_2) = (0.9, 1.1)$[m])

N_c	average [s]	loss[s]	worst [s]	loss[s]
normal	11.6	0.3	51.1	14.9
256	11.6	0.2	42.6	8.5
128	11.4	0.1	50.3	10.9
64	11.7	-0.1	56.7	16.8
32	12.3	-0.2	53.9	8.0
16	12.6	-0.2	58.9	-7.4
8	13.2	-0.3	failure	—
4	13.3	-0.4	64.7	-3.0
2	13.2	-0.5	66.7	8.2
1	13.5	-0.3	79.4	24.2
minimum map	16.2	-0.6	failure	—

A.4 Difference between Actual Environment and Simulator for Scoring Task

We used a simulator that has the following restrictions for evaluation of maps and policies for the scoring task, and for observation of behavior of the robots. This simulator is relatively simple. Experiments with actual robots, or superb simulators shown in [Asanuma, 2004] and `http://araibo.mech.chuo-u.ac.jp/haribote` should be used for in future.

Absence of Corner Walls In the simulator, the triangle wall in every corner is not considered as well as the value iteration algorithm.

Simulation of Collision Collisions between the wall and the ball, and between the wall and the robot are considered in the simulator. On the other hands, the collisions of the two robots, and those of a robot and the ball are not considered.

Simulated Rebound of the Ball

In the simulation, the collision of the ball and the wall around the field is modeled as follows. When the ball collides with the wall, the following difference of distances is calculated.

- the default distance r_{after} of the ball by the kicking action
- the distance from the robot to the collision point

The difference is decomposed into the vertical component d_{v} and the horizontal component d_{h} toward the wall. After the collision, the ball stops $d_{\text{v}}c_{\text{ref}}$ apart from the wall, while the ball travels d_{h} from the point along the wall. In the simulation, the coefficient c_{ref} is set to 0.3.

The dynamics of the rebound is much complex than the above model. It is ideal if the value iteration algorithm and the simulator can use the model for calculation. However, the dynamics easily changes the difference of the material of carpets that are used as the ground of the soccer field in this league. We choose the simpler model than any elaborate one in the simulator. In the case of value iteration, we do not care the rebound due to the problem of computational complexity.

Appendix B

Self-Localization

Here we explain the self-localization method used in the experiments in the RoboCup. The evaluation of this method is shown in [Ueda, 2004].

B.1 Monte Carlo localization with Resetting

We used Monte Carlo localization (MCL). MCL is the application of particle filters for mobile robot localization by Fox, Dellaert *et al.* [Fox, 1999; Dellaert, 1999].

A Bayes filter is the mathematical model that is represented by MCL. It can be applied to mobile robot localization when the pose changes with Markov process, and when information about the pose is stochastically obtained based on the pose. An estimation result, which is represented by a probability distribution, is always maintained in a Bayes filter. It is renewed when a robot moves or obtains sensor information. Since those assumptions are generally suitable for the case of mobile robot localization, MCL and other methods based on Bayes filters work successfully.

B.1.1 Bayes Filters

The pose of a mobile robot on a flat environment is usually represented by the set of x, y and θ. As shown in Fig.B.1(a), (x, y) denote the position of the robot, and θ denotes its orientation based on a coordinate system. We define ℓ as a point in the $xy\theta$-space and ℓ^* as the actual pose of the robot. A space $\mathcal{X}$ that contains all possible ℓ^* is also defined. Moreover, the following symbols:

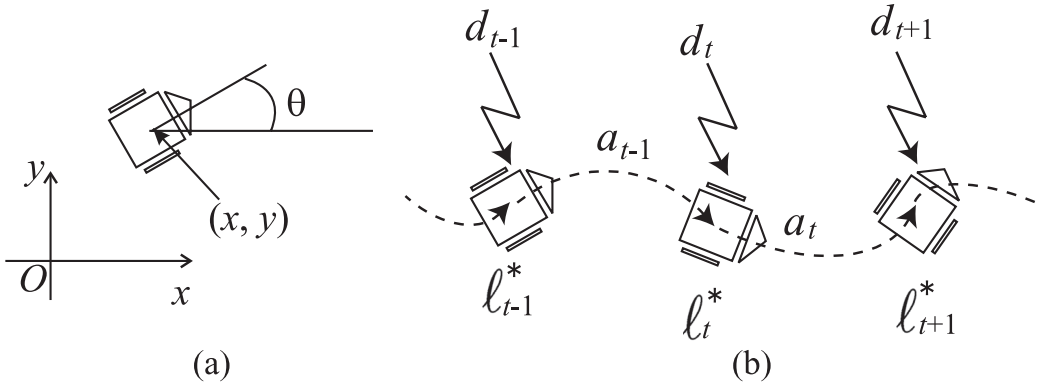


Fig. B.1: Definition of Symbols. (a): parameters for pose ℓ^*_t. (b): relation between discrete time t, information d, action a, and pose ℓ^*.

- discrete time: $T = 0, 1, 2, \ldots, t-1, t, t+1, \ldots,$
- available information at $T = t$: d_t, and
- the robot's action while $t \leq T < t+1$: a_t

are given for formulation. They are shown in Fig.B.1(b). A self-localization problem is formulated with these symbols. The problem is to solve the following probability:

$$B_t(X) = \int_X b_t(\ell) d\ell = \Pr\{\ell^*_t \in X | d_t, a_{t-1}, d_{t-1}, a_{t-2}, d_{t-2}, a_{t-3}, \ldots, d_1, a_0, b_0\}, \quad \text{(B.1)}$$

where X denotes any region in $\mathcal{X}$ and $b_t(\ell)$ is the probability density of $\ell = \ell^*$. Initial probability density function (pdf) b_0 affects the character of a localization problem. If b_0 converges around actual pose ℓ^*, it is called *position tracking*. If ℓ^* is unknown and b_0 is set as an uniform distribution, it is called a *global localization problem*.

When the set of all possible d and the set of all possible action a are represented by $\mathcal{D}$ and $\mathcal{A}$ respectively, Bayes filters require that the following probability densities can be measured for $\forall \ell \in \mathcal{X}, \forall \ell' \in \mathcal{X}, \forall a \in \mathcal{A}$, and $\forall d \in \mathcal{D}$ at any time step.

- $\ell | \ell', a \sim p(\ell | \ell', a)$: the probability density that $\ell = \ell^*_{t+1}$ on condition $\ell' = \ell^*_t$ and $a = a_t$ (Markov property)

- $d|\ell \sim p(d|\ell)$: the probability density that $d = d_t$ on condition $\ell = \ell_t^*$. If d is a discrete quantity, probability $P(d|\ell)$ is required as its substitution.

A Bayes filter can be formulated as

$$\hat{b}_t(\ell) = \int_{\mathcal{X}} p(\ell|\ell', a_{t-1}) b_{t-1}(\ell') d\ell', \text{and} \tag{B.2}$$

$$b_t(\ell) = \frac{p(d_t|\ell)\hat{b}_t(\ell)}{\int_{\mathcal{X}} p(d_t|\ell')\hat{b}_t(\ell')d\ell'}. \tag{B.3}$$

Eq.(B.2) and Eq.(B.3) denote a Markov process and the Bayes theorem respectively. Basically, they are calculated alternately.

B.1.2 Monte Carlo localization

A Bayes filter is implemented with various methods according to conditions. For global localization problems, particle filters [Fox, 1999; Dellaert, 1999; Kwok, 2003] and multi hypothesis trackings (MHT) [Jensfelt, 2001; Kristensen, 2003] are frequently used. A MHT executes many Kalman filters at once. The center pose of each Kalman filter is a hypothesis of the robot's pose. An algorithm that generates and eliminates hypotheses is required for a MHT. On the other hand, particle filters approximate Bayes filters more directly than MHTs. They are sometimes compared from various standpoints [Gutmann, 2002; Kristensen, 2003]. We think that the combination of a Bayes filter and a resetting method can be used in various conditions while its behavior is easy to understand.

Particle filters for mobile robot localization have been named Monte Carlo localization (MCL). This method utilizes particles $s_t^{(i)}$ $(i = 1, 2, \ldots, N)$ that drift in the $xy\theta$-space for approximation of $\hat{b}_t$ and b_t. The particles share weight whose total is one. The weighted distribution of particles approximates the probability distributions.

Figure B.2 shows the algorithm of MCL. $s_t^{(i)} = \langle \ell_t^{(i)}(x_t^{(i)}, y_t^{(i)}, \theta_t^{(i)}), w_t^{(i)} \rangle$, which is the set of its pose in the $xy\theta$-space and its weight, denotes a particle. These processes approximate Eq.(B.2) and Eq.(B.3) respectively. The larger the number of particles N is, the more the approximation is expected. In general, many particles are required when b_t is uncertain. Decision of a proper number of particles is studied by Fox [Fox, 2003].

- Process for a_{t-1} $(i = 1, 2 \ldots, N)$:

  $\ell' \longleftarrow \ell \sim b_{t-1}(\ell)$

  $\ell'' \longleftarrow \ell \sim p(\ell|\ell', a_{t-1})$

  generate $s_t^{(i)} = \langle \ell_t^{(i)}, w_t^{(i)} \rangle = \langle \ell'', 1/N \rangle$.

- Process for d_t $(i = 1, 2 \ldots, N)$:

  $w_t^{(i)} \longleftarrow p(d_t|\ell_t^{(i)})/\alpha$, where $\alpha = \sum_{i=1}^{N} p(d_t|\ell_t^{(i)})$.

Fig. B.2: Algorithm of MCL

It is the fatal state for MCL when there is no particle around actual pose ℓ^*. MCL cannot approximate the probability distribution at the region where no particle exists.

This fatal case often occurs due to not only the lack of particles, but also any accident that goes against the suppositions of a Bayes filter. These accidents cannot be enumerated without omission. However, we can write the worst result by such an accident with only one expression:

$$B_t(Y) = \int_Y b_t(\ell) d\ell \approx 1 \quad (\ell^* \notin Y \subset \mathcal{X}). \tag{B.4}$$

Therefore, a recovery method from the state of this equation can enable an MCL method to be robust.

Incidentally, *kidnapped robot problems* frequently become subjects for research as one of the fatal cases. The *kidnap* refers to the situation that a robot that knows its pose well is moved anywhere. This case can be also represented by Eq. (B.4).

B.2 Resetting Methods

Note that any Bayes estimation does not work well with wrong prior knowledge. Therefore, we must handle the problem beyond the limits of Bayes filters.

The following process: 1) stop a Bayes filter when the state like Eq.(B.4) is detected, 2) start the Bayes filter with new b_0, is a possible way for this

problem. This method is called a resetting method and is used in MCLs. Though Thrun *et al.* have pointed out that these kinds of method does not have theoretical validity in [Thrun, 2001], it does not contradict with Bayes filters. That is because Bayes filters do not restrict the condition of b_0 as long as it is not wrong.

B.2.1 Sensor Resetting

The sensor resetting uses α, which is shown in Fig.B.2, for the trigger of resetting [Lenser, 2000]. α is explained by

$$\alpha = \int_{\mathcal{X}} p(d_t|\ell')\hat{b}_t(\ell')d\ell' \tag{B.5}$$

in a Bayes filter. When the following value

$$\beta = 1 - \alpha/\alpha_{\text{th}} \quad (\alpha_{\text{th}} : \text{a positive threshold}) \tag{B.6}$$

becomes positive, particles are placed based on the pdf:

$$b_{0\text{new}}(\ell) = \frac{1}{1+\beta}b_t(\ell) + \frac{\beta}{1+\beta}p(\ell|d_t). \tag{B.7}$$

B.2.2 Expansion Resetting

We have used another resetting method since RoboCup 2002 in our localization method (Uniform Monte Carlo localization, [Ueda, 2002]). This method is called the *expansion resetting*. We generalize it and propose its use for MCL. It initializes the pdf as

$$b_{0\text{new}} = f[\hat{b}_t] \tag{B.8}$$

when $\beta > 0$. This equation only explains that only $\hat{b}_t$ is used for allocation of particles. A more important thing than Eq.(B.8) is that mapping f acts as $b_{0\text{new}}$ becomes vaguer than $\hat{b}_t$. After that, $b_{0\text{new}} \longleftarrow f[b_{0\text{new}}]$ is repeated every input of new information d until β of Eq.(B.6) becomes zero or less.

In the case of MCL, mapping f expands the region where particles exist as shown in Fig.B.3. The extent of expansion on each axis should be roughly in proportion to the interval in which particles exist. This resetting method has the following property if there is no perceptual aliasing.

- If $\hat{b}_t$ is not wrong, the expansion stops soon.

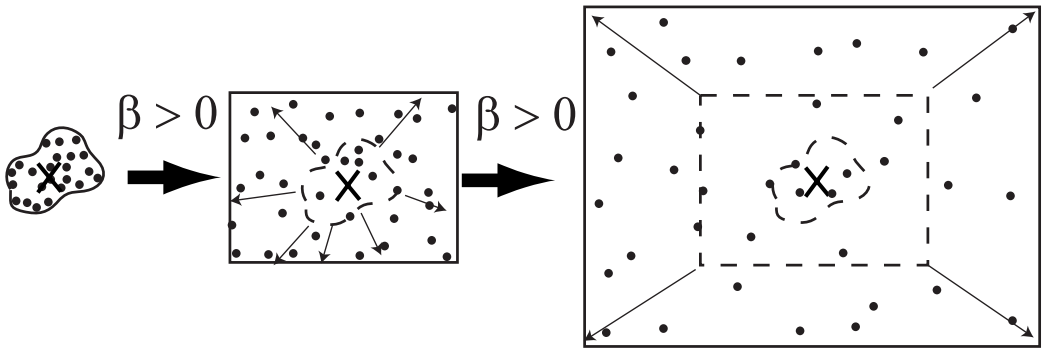


Fig. B.3: Expansion Resettings (an example in a 2-d space)

- If $\hat{b}_t$ is completely wrong, the expansion usually supplies particles near correct pose ℓ^*.

Therefore, this method avoids the discard of accurate b_t when wrong information is singly obtained.

B.2.3 Blending of Resetting Methods

For solving the perceptual aliasing problems or other special problems, the above methods can be blended with some ways. For example, the following choice is possible when $\beta > 0$ with Eq.(B.6).

- if particles exist in a small region: expansion resettings
- otherwise: sensor resettings

This procedure regards $\hat{b}_t$ as important if it converges. Otherwise the procedure attaches importance to sensor information.

B.2.4 Implementation

Common Setting

An MCL, the three resetting methods and an image processing algorithm were implemented on a virtual robot. The image processing algorithm identifies each landmark at first. After that, it returns h, the width of the identified landmark (the width of the arrow in Fig.6.3(b)), and $\hat{\varphi}$, the relative direction of the identified landmark from the robot. Hence $d = \langle h, \hat{\varphi} \rangle$. When

the actual direction of the landmark and its distance from the robot are represented $\varphi^*(\ell)$ and $r(\ell)$ respectively, the probability model for information is computed as:

$$p(d|\ell) = p(h, \hat{\varphi}|\ell) = p(\hat{\varphi} - \varphi_i^*(\ell))p(h|r_i(\ell)), \tag{B.9}$$

where subscription i means the id of the landmark. The independence of $\hat{\varphi}$ and h are assumed for this formulation.

$p(d|\ell)$ was previously measured with random walk of a robot in the simulator. 71,920 measurement results were obtained. $\hat{\varphi} - \varphi^*(\ell)$, r, and h were discretized into 1[deg], 100[mm], and 3[pixel] respectively for creating frequency tables. Figure B.4 illustrates the contents of the tables. There was 10 mistakes of landmark identification and the results at the mistakes were removed from the tables. Though a landmark was recognized only once when $r < 300$[mm], this result was also removed. These tables were implemented on the robot for the probability models in MCL. Probability distribution $p(\ell|d)$ was also calculated and implemented for the sensor resettings.

The number of particles was fixed as $N = 1000$. This number of particles is sufficient for MCL to be stably executed in the field of the four legged robot league. Though it is preferable that the ability of each resetting method for compensation for the shortage of particles is investigated, this paper does not refer to it.

Rule for Expansion Resetting

At an expansion resetting, all particles are randomly placed in a box whose each edge is parallel to x-axis, y-axis, or θ-axis. The center of the box is placed on the mean position of particles: $(\bar{x}_t, \bar{y}_t, \bar{\theta}_t)$, where $\bar{x}_t = \sum_{i=1}^{N} x_t^{(i)} w_t^{(i)}$, $\bar{y}_t = \sum_{i=1}^{N} y_t^{(i)} w_t^{(i)}$. They are calculated based on their state just before the information that causes a resetting is input. When the size of the box is represented by the length of each edge: ω_x[mm], ω_y[mm], and ω_θ[deg], they are calculated based on weighted standard deviations of the distribution: σ_x[mm], σ_y[mm], and σ_θ[deg]. The lengths are basically calculated as $(\omega_x, \omega_y, \omega_\theta) = (6\sigma_x, 6\sigma_y, 6\sigma_\theta)$. However, we set the lower limits of ω_x, ω_y, and ω_θ to 300[mm], 300[mm], and 60[deg] respectively for fear that expansions become slow when particles are converging. If one of the edges is shorter than the limit length for the edge, its length is extended to the limit.

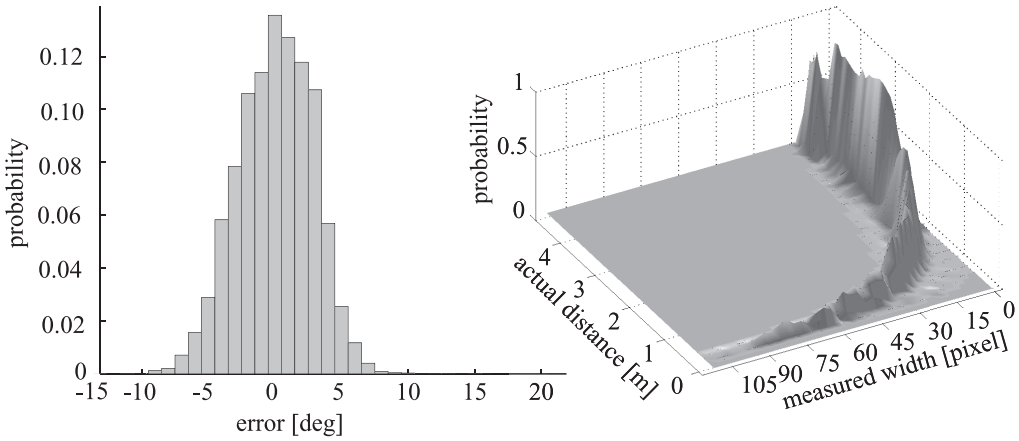


Fig. B.4: Dispersion of Sensor Information. (a) $P(\hat{\varphi} - \varphi_i^*(\ell))$, (b) $P(h|r)$.

Implementation of a blending method

We implemented a blending method of sensor resettings and expansion resettings. When $\beta > 0$ with Eq.(B.6), this method chooses a sensor resetting if $\sigma_x > 300$[mm], $\sigma_y > 300$[mm] or $\omega_\theta > 60$[deg]. Otherwise an expansion resetting is chosen.

Appendix C

Use of State-Action Maps in RoboCup

C.1 Goalkeeper Task

We have utilized a state-action map for our goalkeeper in RoboCup. A goalkeeper should frequently change its pattern of behavior based on its intention. For example, when a goalkeeper goes to the position of the ball so as to grab it, there is a suitable pattern of behavior. When the goalkeeper fends off a shot, another pattern of behavior is required. In this case, we sometimes think that a policy is required for every pattern of behavior. However, we have used only one state-action map for whole behavior of a goalkeeper. The state-action map can be created by the following manner.

State, Action, and State Transition

We use identical state variables with the ball approaching task. The domain and the way to discretize each parameter are shown in Table C.1(a). We then add 14 by 27 by 18 cells in $xy\theta$-space for occasions when the robot is not observing the ball. We call these additional cells *ball invisible states*. The total number of cells is $2,980,152$. Though dynamics of the ball is not taken into account, the planned result can be used for competitions as shown in Fig.C.1.

In this task, six kinds of actions are added to the set of actions as shown in Table C.3. The goalkeeper robot can use kinds of action. State transition probabilities are computed by the identical method with the ball approaching task.

Table C.1: Parameters for Value Iteration

(a) State Space

state variable	domain	width of a cell	# of cells
x	$[1000, 2400)$[mm]	100[mm]	14
y	$[-1350, 1350)$[mm]	100[mm]	27
θ	$[-180, 180)$[deg]	20[deg]	18
r	$[120, 2020)$[mm]	100[mm]	19
φ	$[-92, 92]$[deg]	8[deg]	23

(b) Final States

case	condition	value[step]
(i)	*ball invisible* & $\lvert x - 2200\rvert < 100$ & $\lvert y\rvert < 100$ & $\lvert\theta\rvert > 150$	-15
(ii)	$2100 < x < 2200$ & $\lvert\theta\rvert > 150$ & $\{\lvert\varphi\rvert < 12$ or $(\varphi < 0$ & $y > 50)$ or $(\varphi > 0$ & $y < -50)\}$	-10
(iii)	$\varphi < 45$ & $r < 200$ & $\{\lvert\theta\rvert > 135$ or $\lvert\varphi_{\mathrm{G}}\rvert > 135\}$	0

(c) Penalty $\mathcal{R}^a_{ss'}$

case	penalty[step]
(1) Action a is executed.	-1
(2) $\lvert\varphi\rvert \leq 92$ and $\lvert\varphi'\rvert > 92$	-5
(3) $\lvert\varphi'\rvert > 92$[deg], $\lvert\varphi'_{\mathrm{G}}\rvert > 30$ and $x' < 2100$	-5
(4) $s' \notin \mathcal{S}$	-250

Table C.2: Computation time (with a 3.6GHz Pentium IV CPU)

procedure	time
Monte Carlo integration for P_{robot} (10^5 samples)	[s]
Monte Carlo integration for P_{ball} (10^4 samples)	[s]
value iteration (one iteration)	40.7[s]
total time (71 sweeps)	2.9×10^3[s]

Final states

The task is to guide the goalkeeper to appropriate pose (x, y, θ) according to (r, φ) without an own-goal. The appropriate pose restlessly changes when other robots move the ball. There are some candidates for appropriate poses as shown in Fig.C.3 and on Table C.1(b). Final states are set based on these poses. When the center of a cell satisfies one of the conditions, the cell is one of final states. φ_{G} is the direction of point $(2100, 0)$ (center of the goal) on Σ_{r}.

Penalty

The reward, which should be called penalty in this case, is given every one action. That value is -1 regardless of the kind of action. We set other conditions that should be penalized as shown in Table C.1(c). (2) is given because it is disadvantage if the goalkeeper loses the ball. (3) prevents own-goals. In case (4), the robot collides with a wall around the field or the ball. The former upsets self-localization since the orientation of the robot is changed unexpectedly. The latter is connected directly with an own-goal.

Value Iteration and the Result

The state-value function is obtained by value iteration with the above conditions. V^* is obtained within 49 minutes with a 3.6 GHz CPU. The graph in Fig.C.5 is a part of the state-value function. Each value on the graph is obtained when θ is fixed to the direction to the ball. The state-value function has discontinuity and local maxima. Figure C.6 then shows some examples of the behavior computed from π^*.

Table C.3: Added Actions for Goalkeeper Task

name	δ_x[mm]	δ_y[mm]	δ_θ[deg]
Backward	0.0	-87.0	0.0
ShortBackward	0.0	-46.0	0.0
RightForwardTurnLeft	30.0	50.0	17.0
LeftForwardTurnRight	-25.0	70.0	-15.0
RightForwardTurnRight	92.0	80.0	-15.0
LeftForwardTurnLeft	-100.0	80.0	15.0

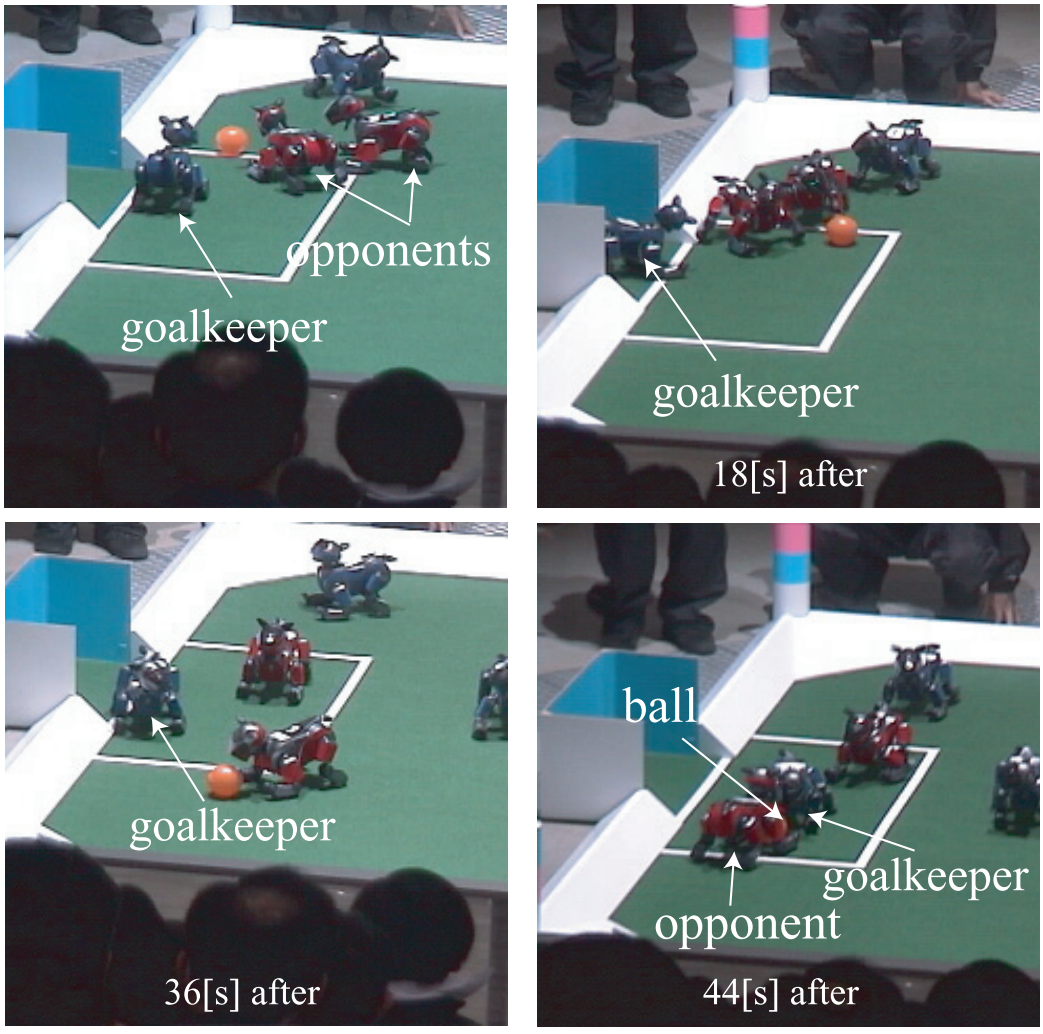


Fig. C.1: Scenes in a game (Team ARAIBO 0-2 Kyushu Institute of Technology)

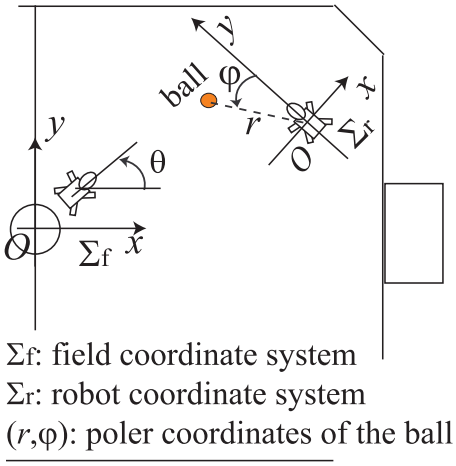


Fig. C.2: Coordinate Systems

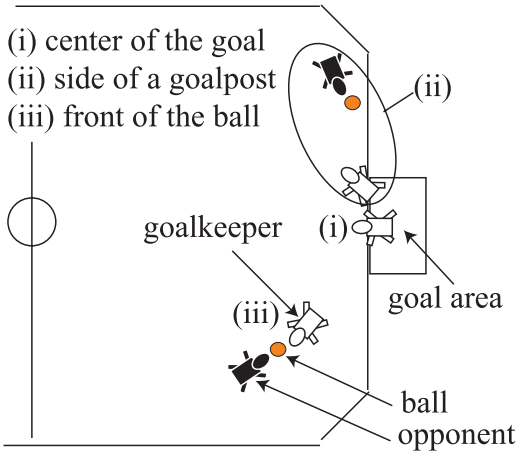


Fig. C.3: Appropriate Poses

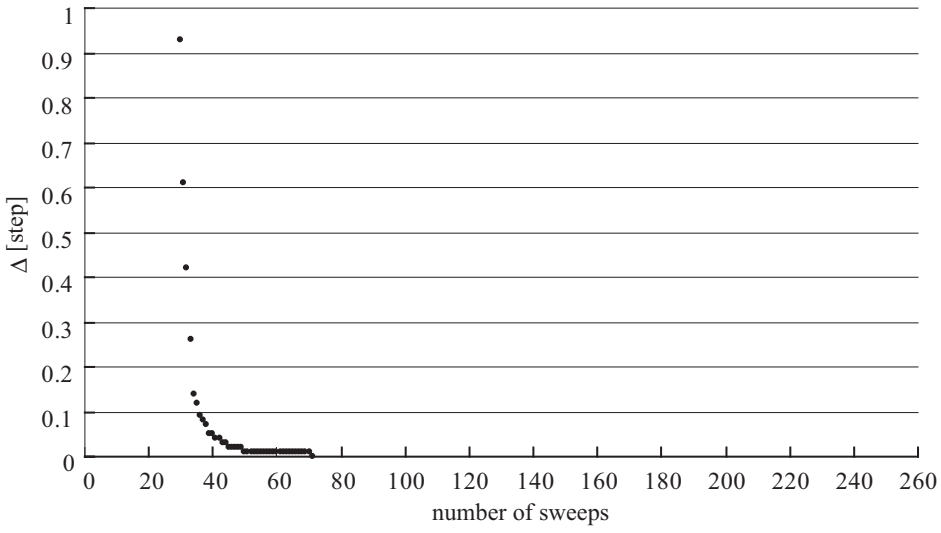


Fig. C.4: Convergence of state-value function

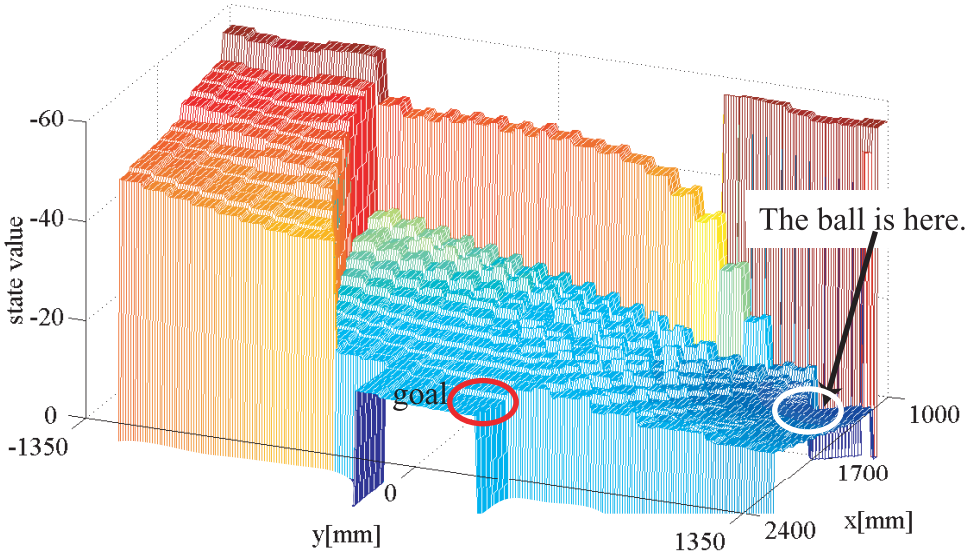


Fig. C.5: A Part of The State-Value Function

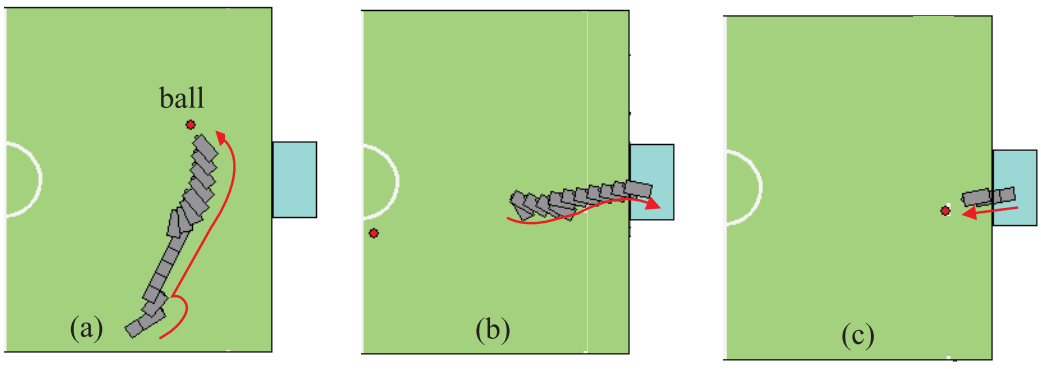


Fig. C.6: Behavior of The Robot with π^*

References

[Al-Ansari, 1998] Mohammad A. Al-Ansari and Ronald J. Williams: "Robust, Efficient, Globally-Optimized Reinforcement Learning with the Parti-Game Algorithm," In *NIPS*, pp. 961–967, 1998.

[Albus, 1975a] James S. Albus: "A New Approach to Manipulator Control: The Cerebellar Model Articulation Controller (CMAC)," *Journal of Dynamic Systems, Measurement and Control*, 97(3), pp. 220–227, 1975.

[Albus, 1975b] James S. Albus: "Data Storage in the Cerebellar Model Articulation Controller (CMAC)," *Journal of Dynamic Systems, Measurement and Control*, 97(3), pp. 228–233, 1975.

[Asada, 1999] Minoru Asada, Hiroaki Kitano, Itsuki Noda, and Manuela Veloso: "RoboCup: Today and Tomorrow – What we have learned," *Artificial Intelligence*, 110pp. 193–214, 1999.

[Asanuma, 2004] Kazunori Asanuma, Kazunori Umeda, Ryuichi Ueda, and Tamio Arai: "Development of a Simulator of Environment and Measurement for Autonomous Mobile Robots Considering Camera Characteristics," In *D. Polani et al. (Eds.): RoboCup 2003: Robot Soccer World Cup VII*, pp. 446–457, 2004.

[Banavar, 2003] Ravi N. Banavar and Arun D. mahindrakar: "Energy based swing-up of the Acrobot and Time-optimal Motion," In *Proc. of IEEE International Conference on Control Applications*, pp. 706–711, 2003.

[Barraquand, 1995] Jérôme Barraquand and Pierre Ferbach: "Motion Planning with Uncertainty: The Information Space Approach," In *Proc. of IEEE International Conference on Robotics and Automation*, pp. 1341–1348, 1995.

[Barron, 1998] Andrew Barron, Jorma Rissanen, and Bin Yu: "The Minimum Description Length Principle in Coding and Modeling," *IEEE Trans. on Information Theory*, 44(6), pp. 2743–2760, 1998.

[Baum, 2000] David Baum: *Dave Baum's Definitive Guide to LEGO MINDSTORMS*, Springer, 2000.

[Bellman, 1957] Richard Bellman: *Dynamic Programming*, Princeton University Press, Princeton, NJ, 1957.

[Boone, 1997] Gary Boone: "Minimum-time Control of the Acrobot," In *Proc. of IEEE ICRA*, pp. 3281–3287, 1997.

[Bouzy, 2001] Bruno Bouzy and Tristan Cazenave: "Computer Go: An AI oriented survey," *Artificial Intelligence*, 132(1), pp. 29–103, 2001.

[Boyan, 1995] Justin A. Boyan and Andrew W. Moore: "Generalization in Reinforcement Learning: Safely Approximating the Value Function," In *Advances in Neural Information Processing Systems 7*, pp. 369–376, 1995.

[Broomhead, 1988] D.S. Broomhead and David Lowe: "Multivariable Functional Interpolation and Adaptive Networks," *Complex Systems*, 2(3), pp. 321–355, 1988.

[Buck, 2002] Sebastian Buck, Michael Beetz, and Thorsten Schmitt: "Approximating the Value Function for Continuous Space Reinforcement Learning in Robot Control," In *Proc. of the IEEE/RSJ IROS*, pp. 1062–1067, 2002.

[Campbell, 2002] Murray Campbell, A. Joseph Hoane Jr., and Feng hsiung Hsu: "Deep Blue," *Artificial Intelligence*, 134( ), pp. 57–83, 2002.

[Choset, 2005] Howie Choset, Kevin M. Lynch, Seth Hutchinson, George Kantor, Wolfram Burgard, Lydia E. Kavraki, and Sebastian Thrun: *Principles of Robot Motion*, MIT Press, Cambridge, MA, 2005.

[Delft, 1994] Christian van Delft: "Approximate Solutions for Large-Scale Piecewise Deterministric Control Systems Arising in Manufacturing Flow Control Models," *IEEE Trans. on Robotics and Automation*, 10(2), pp. 142–152, 1994.

[Dellaert, 1999] Frank Dellaert, Dieter Fox, Wolfram Burgard, and Sebastian Thrun: "Monte Carlo Localization for Mobile Robots," In *Proc. of IEEE International Conference on Robotics and Automation (ICRA99)*, pp. 1322–1328, 1999.

[Equitz, 1989] W. H. Equitz: "A new vector quantization clustering algorithm," *IEEE Trans. Acoust. Speech Signal Process*, 37(10), pp. 1568–1575, 1989.

[Ewing, 1985] George M. Ewing: *Calculus of Variations with Application*, Dover publications, Inc., Mineorla, NY, 1985.

[Fekri, 2000] Faramarz Fekri, Russell M. Mersereau, and Ronald W. Schafer: "A Generalized Interpolative Vector Quantization Method for Jointly Optimal Quantization, Interpolation, and Binarization of Text Images," *IEEE Trans. on Image Processing*, 9(7), pp. 1272–1281, Jul. 2000.

[Ferbach, 1998] Pierre Ferbach: "A Method of Progressive Constraints for Nonholonomic Motion Planning," *IEEE Trans. on Robotics and Automation*, 14(1), pp. 172–179, Feb. 1998.

[Fox, 2003] Dieter Fox: "Adapting the Sample Size in Particle Filters Through KLD-Sampling," *International Journal of Robotics Research*, 22(12), pp. 985–1004, 2003.

[Fox, 1999] Dieter Fox, Wolfram Burgard, Frank Dellaert, and Sebastian Thrun: "Monte Carlo Localization: Efficient Position Estimation for Mobile Robots," In *Proc. of AAAI-99*, pp. 343–349, 1999.

[Fraser, 2005] Gordon Fraser and Franz Wotawa: "Cooperative Planning and Plan Execution in Partially Observable Dynamic Domains," In D. Nardi et al., editors, *RoboCup 2004: Robot Soccer World Cup VIII*, pp. 524–531, 2005.

[Fujii, 2004] Hikari Fujii, Daiki Sakai, and Kazuo Yoshida: "Cooperative Control Method Using Evaluation Information on Objective Achievement," In *Proc. of DARS*, pp. 201–210, 2004.

[Fujita, 2003] Masahiro Fujita: "Sony Four Legged Robot League at RoboCup 2002," *Gal A. Kaminka, et al. (Eds.) RoboCup 2002: Robot Soccer World Cup VI*, pp. 469–476, 2003.

[Fukase, 2002] Takeshi Fukase, Masahiro Yokoi, Yuichi Kobayashi, Hideo Yuasa, and Tamio Arai: "Quadruped Robot Navigation Considering the Observation Cost," In A. Birk, S. Coradeschi, and S.Tadokoro, editors, *RoboCup 2001: Robot Soccer World Cup V*, pp. 350–355, 2002.

[Fukase, 2003] Takeshi Fukase, Yuichi Kobayashi, Ryuichi Ueda, Takanobu Kawabe, and Tamio Arai: "Real-time Decision Making under Uncertainty of Self-Localization Results," *Gal A. Kaminka, et al. (Eds.) RoboCup 2002: Robot Soccer World Cup VI*, pp. 375–383, 2003.

[Fukazawa, 2003] Yusuke Fukazawa, Chomchana Trevai, Jun Ota, and Tamio Arai: "Controlling a Mobile Robot That Searches for and Rearranges Objects with Unknown Locations and Shapes," In *Proc. of IEEE/RSJ IROS*, pp. 1721–1726, 2003.

[Ge, 2000] S.S. Ge and Y.J. Cui: "New Potential Functions for Mobile Robot Path Planning," *IEEE Trans. on Robotics and Automation*, 16(5), pp. 615–620, 2000.

[Gersho, 1992] A. Gersho and R. M. Gray: *Vector Quantization and Signal Compression*, Kluwer Academic Publishers, Boston, MA, 1992.

[Greer, 2000] Kieran Greer: "Computer chess move-ordering schemes using move influence," *Artificial Intelligence*, 120(2), pp. 235–250, 2000.

[Gutmann, 2002] J. Gutmann and Dieter Fox: "An Experimental Comparison of Localization Methods Continued," In *Proc. of IROS*, pp. 454–459, 2002.

[Hu, 1997] Huosheng Hu and Michael Brady: "Dynamic Global Path Planning with Uncertainty for Mobile Robots in Manufacturing," *IEEE Trans. on Robotics and Automation*, 13(5), pp. 760–767, Oct. 1997.

[Huang, 2002] Yuan-Hao Huang and Tzi-Dar Chiueh: "A New Audio Coding Scheme Using a Forward Masking Model and Perceptually Weighted Vector Quantization," *IEEE Trans. on Speech and Audio Processing*, 10(5), pp. 325–335, Jul. 2002.

[Iida, 2002] Hiroyuki Iida, Makoto Sakuta, and Jeff Rollason: "Computer shogi," *Artificial Intelligence*, 134(1-2), pp. 121–144, 2002.

[Ito, 2002] K. Ito and F. Mathuno: "A Study of Reinforcement Learning for the Robot with Many Degrees of Freedom -Acquisition of Locomotion Patterns for Multi Legged Robot-," In *Proc. of ICRA-2002*, pp. 3392–3396, 2002.

[Jensfelt, 2001] P. Jensfelt and S. Kristensen: "Active Global Localization for a Mobile Robot Using Multiple Hypothesis Tracking," *IEEE Trans. on Robotics and Automation*, 17(5), pp. 748–760, 2001.

[Kavraki, 1996] Lydia E. Kavraki, Petr Švestka, Jean-Claude Latombe, and Mark H. Overmars: "Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces," *IEEE Transaction on Robotics and Automation*, 12(4), pp. 566–580, 1996.

[Khatib, 1986] Oussama Khatib: "Real-Time Obstacle Avoidance for Manipulators and Mobile Robots," *International Journal of Robotics Research*, 5(1), pp. 90–98, 1986.

[Khojastech, 2004] Mohammad Reza Khojastech et al.: "Using Learning Automata in Cooperation among Agents in a Team," In *Proc. of International RoboCup Symposium*, pp. CD–ROM, 2004.

[Kimura, 1998] Hajime Kimura and Shigenobu Kobayashi: "An Analysis of Actor/Critic Algorithms using Eligibility Traces: Reinforcement Learning with Imperfect Value Function," In *15th International Conference on Machine Learning*, pp. 278–286, 1998.

[Kitano, 1997] Hiroaki Kitano, Minoru Asada, Yasuo Kuniyoshi, Itsuki Noda, and Eiichi Osawa: "RoboCup: The Robot World Cup Initiative," In *Proc of The First International Conference on Autonomous Agent*, pp. 340–347, 1997.

[Kleiner, 2003] Alexander Kleiner, Markus Dietl, and Bernhard Nebel: "Towards a Life-Long Learning Soccer Agent," *Gal A. Kaminka, et al. (Eds.) RoboCup 2002: Robot Soccer World Cup VI*, pp. 126–134, 2003.

[Kobayashi, 2002] Yuichi Kobayashi, Hideo Yuasa, and Tamio Arai: "Function Approximation for Reinforcement Learning Based on Reaction-Diffusion Equation on a Graph," In *Proc. of SICE Annual Conference 2002*, pp. 916–921, 2002.

[Koren, 1991] Y. Koren and J. Borenstein: "Potential Field Methods and Their Inherent Limitations for Mobile Robot Navigation," In *Proc. of IEEE ICRA*, pp. 1398–1404, 1991.

[Kristensen, 2003] Steen Kristensen and Patric Jensfelt: "An Experimental Comparison of Localisation Methods, the MHL Sessions," In *Proc. of IROS*, pp. 992–997, 2003.

[Kwok, 2003] Kody Kwok, Dieter Fox, and Marina Meila: "Adaptive Real-time Particle Filters for Robot Localization," In *Proc. of IEEE ICRA*, pp. 2836–2841, 2003.

[Latombe, 1991] Jean-Claude Latombe: *Robot Motion Planning*, Kluwer Academic Publishers, Boston, MA, 1991.

[Laue, 2004] Tim Laue and Thomas Röfer: "A Behavior Architecture for Autonomous Mobile Robots Based on Potential Fields," In *Proc. of RoboCup 2004 symposium*, pp. CD–ROM, 2004.

[Laumond, 1994] Jean-Paul Laumond, Paul E. Jacobs, Michel Taix, and Richard M. Murray: "A Motion Planner for Nonholonomic Mobile Robots," *IEEE Trans. on Robotics and Automation*, 10(5), pp. 577–593, 1994.

[LaValle, 1999] S. M. LaValle and J. J. Kuffner: "Randomized Kinodynamic Planning," In *Proc. of IEEE International Conference on Robotics and Automation*, pp. 473–479, 1999.

[LaValle, 2000] Steven M. LaValle: "Robot Motion Planning: A Game–Theoretic Foundation," *Algorithmica*, 26pp. 430–465, 2000.

[Lenser, 2000] Scott Lenser and Manuela Veloso: "Sensor resetting localization for poorly modelled robots," In *Proc. of IEEE ICRA*, pp. 1225–1232, 2000.

[Likhachev, 2002] Maxim Likhachev and Sven Koenig: "Speeding up the Parti-Game Algorithm," In *NIPS*, pp. 1563–1570, 2002.

[Miyazawa, 2005] Kiyokazu Miyazawa, Yusuke Maeda, and Tamio Arai: "Planning of Graspless Manipulation based on Rapidly-Exploring Random Trees," In *Proc. of 6th IEEE Int. Symp. on Assembly and Task Planning (ISATP 2005)*, pp. ITP–3, 2005.

[Moody, 1989] John Moody and Christian J. Darken: "Fast Learning in Networks of Locally-Tuned Processing Units," *Neural Computation*, 1(2), pp. 281–294, 1989.

[Moore, 1995] Andrew W. Moore and Christopher G. Atkeson: "The Partigame Alogirthm for Variable Resolution Reinforcement Learning in Multidimensional State-spaces," *Machine Learning*, 21, 1995.

[Müller, 2002] Martin Müller: "Computer Go," *Artificial Intelligence*, 134(1-2), pp. 145–179, 2002.

[Munos, 1998] Rémi Munos and Andrew Moore: "Barycentric Interpolator for Continuous Space and Time Reinforcement Learning," *Neural Information Processing Systems 11*, pp. 1024–1030, 1998.

[Munos, 2002] Rémi Munos and Andrew Moore: "Variable Resolution Discretization in Optimal Control," *Machine Learning*, 49(2-3), pp. 291–323, 2002.

[Nitschke, 2006] Geoff Nitschke: "Emergent cooperation in robocup: A review," In *RoboCup 2005: Robot Soccer World Cup IX*, pp. 512–520, 2006.

[Ota, 2004] Jun Ota: "Rearrangement of Multiple Movable Objects," In *Proc. of IEEE ICRA*, pp. 1962–1967, 2004.

[Pierre, 1986] Donald A. Pierre: *Optimization Theory with Applications*, Dover Publications, Inc., Mineola, NY, 1986.

[Rissanen, 1999] J. Rissanen: "Hypothesis Selection and Testing by the MDL Principle," *The Computer Journal*, 42(4), pp. 260–269, 1999.

[Roy, 1999] Nicholas Roy, Wolfram Burgard, Dieter Fox, and Sebastian Thrun: "Coastal Navigation - Mobile Robot Navigation with Uncertainty in Dynamic Environments," In *Proc. of IEEE ICRA*, pp. 35–40, 1999.

[Samejima, 1999] K. Samejima and T. Omori: "Adaptive internal state space construction method for reinforcement learning," *Neural Networks*, 12(7-8), pp. 1143–1155, 1999.

[Spong, 1994] Mark W. Spong: "Swing Up Control of the Acrobot," In *Proc. of IEEE ICRA*, pp. 2356–2361, 1994.

[Spong, 1995] Mark W. Spong: "The swing up control problem for the Acrobot ," *IEEE Control Systems Magazine*, 15(1), pp. 49–55, 1995.

[Sutton, 1988] Richard S. Sutton: "Learning to Predict by the Methods of Temporal Differences," *Machine Learning*, 3, pp. 9–44, 1988.

[Sutton, 1996] Richard S. Sutton: "Generalization in Reinforcement Learning: Successful Examples Using Space Coarse Coding," In *Neural Information Processing Systems*, pp. 1038–1044, 1996.

[Sutton, 1998] Richard S. Sutton and Andrew G. Barto: *Reinforcement Learning: An Introduction*, The MIT Press, Cambridge, MA, 1998.

[Takahashi, 1999] Yasutake Takahashi, Masanori Takeda, and Minoru Asada: "Continuous Valued Q-learning for Vision-Guided Behavior Acquisition," In *Proc. of the 1999 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems*, pp. 255–260, 1999.

[Takahashi, 2001] Yasutake Takahashi, Masanori Takeda, and Minoru Asada: "Improvement Continuous Valued Q-learning and Its Application to Vision Guided Behavior Acquisition," *RoboCup 2000: Robot Soccer. World Cup IV*, pp. 385–390, 2001.

[Tesauro, 1995] Gerald Tesauro: "Temporal Difference Learning and TD-Gammon," *Communications of the ACM*, 38(3), pp. 58–68, 1995.

[Thrun, 2001] S. Thrun et al.: "Robust Monte Carlo Localization for Mobile Robots," *Artificial Intelligence Journal*, 128(1-2), pp. 99–141, 2001.

[Tsai, 2000] Jyi-Chang Tsai, Chaur-Heh Hsieh, and Te-Cheng Hsu: "A New Dynamic Finite-State Vector Quantization Algorithm for Image Compression," *IEEE Trans. on Image Processing*, 9(11), pp. 1825–1836, Nov. 2000.

[Tuyls, 2003] Karl Tuyls, Sam Maes, and Bernard Manderick: "Reinforcement Learning in Large State Spaces – Simulated Robotic Soccer as a Testbed," *Gal A. Kaminka, et al. (Eds.) RoboCup 2002: Robot Soccer World Cup VI*, pp. 319–326, 2003.

[Ueda, 2002] Ryuichi Ueda, Takeshi Fukase, Yuichi Kobayashi, Tamio Arai, Hideo Yuasa, and Jun Ota: "Uniform Monte Carlo Localization – Fast and Robust Self-localization Method for Mobile Robots," In *Proc. of ICRA*, pp. 1353–1358, 2002.

[Ueda, 2003] Ryuichi Ueda, Tamio Arai, Kazunori Asanuma, Shogo Kamiya, and Kazunori Umeda: "Mobile Robot Navigation based on Expected State Value under Uncertainty of Self-localization," In *Proc. of IROS*, pp. 473–478, 2003.

[Ueda, 2004] Ryuichi Ueda, Tamio Arai, Kohei Sakamoto, Toshifumi Kikuchi, and Shogo Kamiya: "Expansion Resetting for Recovery from Fatal Error in Monte Carlo Localization – Comparison with Sensor Resetting Methods," In *Proc. of IROS*, pp. 2481–2486, 2004.

[Ueda, 2005] Ryuichi Ueda, Tamio Arai, Kohei Sakamoto, Yoshiaki Jitsukawa, Kazunori Umeda, Hisashi Osumi, Toshifumi Kikuchi, and Masaki Komura: "Real-Time Decision Making with State-Value Function under Uncertainty of State Estimation," In *Proc. of ICRA*, 2005.

[Veloso, 1998] Manuela Veloso, William Uther, Masahiro Fujita, Minoru Asada, and Hiroaki Kitano: "Playing Soccer with Legged Robots," In *Proc. of IEEE/RSJ IROS-98*, 1998.

[Watkins, 1992] Christopher J.C.H. Watkins and Peter Dayan: "Q-Learning," *Machine Learning*, 8(3-4), pp. 279–292, 1992.

[Xin, 2002] Xin Xin and Masahiro Kaneda: "The Swing up Control for the Acrobot based on Energy Control Approach," In *Proc. of IEEE Conf. on Decision and Control*, pp. 3261–3266, 2002.

[Xin, 2004] Xin Xin and Masahiro Kaneda: "New Analytical Results of the Energy Based Swinging up Control of the Acrobot," In *Proc. of IEEE Conf. on Decision and Control*, pp. 704–709, 2004.

[Yoshimoto, 2005] J. Yoshimoto, M. Nishimura, Y. Tokita, and S. Ishii: "Acrobot control by learning the switching of multiple controllers," *Journal of Artificial Life and Robotics*, 9(2), pp. 67–71, 2005.

Publication and Award List

Journal Papers

1. Yuichi Kobayashi, Takeshi Fukase, Ryuichi Ueda, Hideo Yuasa, Tamio Arai: "Design of Quadruped Robot Soccer Behavior Considering Observational Cost," Journal of the Robotics Society of Japan, Vol. 21, No.7, 802-810 (in Japanese), 2003.

2. Ryuichi Ueda, Takeshi Fukase, Yuichi Kobayashi, Tamio Arai and Shogo Kamiya: "Lossy Compression of Deterministic Policy Map with Vector Quantization," Journal of the Robotics Society of Japan, Vol.23, No.1, pp.104-112 (in Japanese), 2005.

3. Ryuichi Ueda, Tamio Arai, Kazunori Asanuma, Kazunori Umeda, and Hisashi Osumi: "Recovery Methods for Fatal Estimation Errors on Monte Carlo Localization," Journal of the Robotics Society of Japan, Vol.23, No.4, pp.84-91 (in Japanese), 2005.

4. Kazunori Umeda, Kazunori Asanuma, Toshifumi Kikuchi, Ryuichi Ueda, Hisashi Osumi, and Tamio Arai: "Development of a Simulator of Environment and Measurement for Multiple Autonomous Mobile Robots Considering Camera Characteristics," Journal of the Robotics Society of Japan, Vol.23, No.7, pp.878-885 (in Japanese), 2005.

5. Ryuichi Ueda and Tamio Arai: "Real-Time Decision Making of Autonomous Robot under Uncertainty of State Estimation by Using Particle Filter and Q-MDP Value Method," Journal of the Robotics Society of Japan, Vol.25, No.1, pp.103-112 (in Japanese), 2007.

Conference Papers (reviewed)

1. Takeshi Fukase, Masahiro Yokoi, Yuichi Kobayashi, Ryuichi Ueda, Hideo Yuasa and Tamio Arai: "Quadruped Robot Navigation Considering the Observational Cost," Andreas Birk, Silvia Coradeschi and

Satoshi Tadokoro (Eds.), RoboCup 2001: Robot Soccer World Cup V, pp. 350-355, Springer, 2002.

2. Ryuichi Ueda, Takeshi Fukase, Yuichi Kobayashi, Tamio Arai, Hideo Yuasa, and Jun Ota: "Uniform Monte Carlo Localization –Fast and Robust Self-localization Method for Mobile Robots.," IEEE International Conference on Robotics and Automation (ICRA), pp. 1353-1358, 2002.

3. Ryuichi Ueda, Takeshi Fukase, Yuichi Kobayashi and Tamio Arai: "Vector Quantization for State-Action Map Compression," IEEE International Conference on Robotics and Automation (ICRA), pp. 2356-2361, 2003.

4. Takeshi Fukase, Yuichi Kobayashi, Ryuichi Ueda, Takanobu Kawabe and Tamio Arai: "Real-time Decision Making under Uncertainty of Self-Localization Results," Gal A. Kaminka, et al. (Eds.) RoboCup 2002: Robot Soccer World Cup VI, pp. 375-383, 2003.

5. Ryuichi Ueda and Tamio Arai: "Value Iteration Under the Constraint of Vector Quantization for Improving Compressed State-Action Maps," Proc. of IEEE International Conference on Robotics and Automation (ICRA), pp. 4771-4776, 2004.

6. Kazunori Asanuma, Kazunori Umeda, Ryuichi Ueda and Tamio Arai: "Development of a Simulator of Environment and Measurement for Autonomous Mobile Robots Considering Camera Characteristics," Daniel Polani, et al. (Eds.), RoboCup 2003: Robot Soccer World Cup VII, pp. 446-457, 2004.

7. Chomchana Trevai, Ryuichi Ueda, Toshio Moriya and Tamio Arai: "Mobile Robot System for Composition of Seamless and High Resolution Images –Mobile Robot Localization and Map Building," Proc. of IEEE Int. Conf. on Systems, Man and Cybernetics, pp. 1822-1827, 2003.

8. Ryuichi Ueda, Tamio Arai, Kazunori Asanuma, Shogo Kamiya, Toshifumi Kikuchi, Kazunori Umeda: "Mobile Robot Navigation based on Expected State Value under Uncertainty of Self-localization," Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 473-478, 2003.

9. Ryuichi Ueda, Toshio Moriya, Trevai Chomchana, and Tamio Arai: "Mobile Robot Control for Composition of Seamless and High-

resolution Images in Library," IS&T/SPIE's 16th Annual Symposium on Electronic Imaging, San Jose, CA, USA, 2004.

10. Chomchana Trevai, Ryuichi Ueda, Toshio Moriya, and Tamio Arai: "Integration of Monte Carlo Localization Method for Mobile Robot with Sonar Array," Proc. of The 8th Conference on Intelligent Autonomous Systems (IAS-8), Amsterdam, Netherlands, 2004.

11. Ryuichi Ueda, Tamio Arai and Kohei Sakamoto, Toshifumi Kikuchi and Shogo Kamiya: "Expansion Resetting for Recovery from Fatal Error in Monte Carlo Localization –Comparison with Sensor Resetting Methods," Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 2481-2486, 2004.

12. Ryuichi Ueda, Tamio Arai, Kohei Sakamoto, Yoshiaki Jitsukawa, Kazunori Umeda, Hisashi Osumi, Toshifumi Kikuchi and Masaki Komura: "Real-Time Decision Making with State-Value Function under Uncertainty of State Estimation –Evaluation with Local Maxima and Discontinuity," Proc. of IEEE International Conference on Robotics and Automation (ICRA), 3475-3480, 2005.

13. Ryuichi Ueda, Tamio Arai, and Kazutaka Takeshita: "Vector Quantization for State-Action Map Compression – Comparison with Coarse Discretization Techniques and Efficiency Enhancement," Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 166-171, 2005.

14. Yoshiaki Jitsukawa, Ryuichi Ueda, Tamio Arai, Toshifumi Kikuchi, and Kazunori Umeda: "An Object Recognition method using Fuzzy Color Classification," 36th International Symposium on Robotics (ISR), CD-ROM, 2005.

15. Kazutaka Takeshita, Ryuichi Ueda, and Tamio Arai: "Fast Vector Quantization for State-Action Map Compression," Proc. of The 9th International Conference on Intelligent Autonomous Systems (IAS-9), 694-701, 2006.

16. Hisashi Osumi, Shogo Kamiya, Hirokazu Kato, Kazunori Umeda, Ryuichi Ueda, and Tamio Arai: "Time Optimal Control for Quadruped Walking Robots," IEEE International Conference on Robotics and Automation (ICRA), pp. 1102-1108, 2006.

17. Toshifumi Kikuchi, Kazunori Umeda, Ryuichi Ueda, Yoshiaki Jitsukawa, Hisashi Osumi, and Tamio Arai: "Improvement of Color

Recognition Using Colored Objects," A. Bredenfeld et al. (Eds.): RoboCup 2005, pp. 537-544, 2006.

18. Feng DUAN, Yoshiaki JITSUKAWA, Ryuichi UEDA, and Tamio ARAI: "Pushing Motions of Quadruped Robot Generated by Genetic Algorithm," Proc. of The 6th International Workshop on Emergent Synthesis (IWES), pp. 277-282, 2006.

19. Natsuki Yamanobe, Tamio Arai, and Ryuichi Ueda: "Robot Motion Planning by Reusing Multiple Knowledge under Uncertain Conditions," Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 2232-2237, 2006.

20. Ryuichi Ueda, Tamio Arai, and Kojiro Matsushita: "Creation and Compression of Global Control Policy for Swinging up Control of the Acrobot," Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 2557-2562, 2006.

21. Ryuichi Ueda, Kohei Sakamoto, Kazutaka Takeshita and Tamio Arai: "Dynamic Programming for Creating Cooperative Behavior of Two Soccer Robots —Part 1: Computation of State-Action Map," Proc. of IEEE International Conference on Robotics and Automation (ICRA), 2007. (to appear)

Awards

in Academic Activity

1. Young Investigator Excellence Award, Robotics Society of Japan

in RoboCup Activity (as a member of Team ARAIBO)

1. RoboCup JapanOpen 2000 Robotics Society of Japan Award
2. 3rd place, Technical Challenge of Four Legged Robot League in RoboCup 2003 Padua
3. 2nd place, Technical Challenge of Four Legged Robot League in RoboCup 2004 Lisboa
4. 3rd place, Technical Challenge of Four Legged Robot League in RoboCup 2005 Osaka