Study on reinforcement learning using Voronoi diagram in continuous state space

連続的な状態空間のボロノイ分割を用いた強化学習に関する研究

KATHY THI AUNG

Graduate School of Science and Engineering

Department of System Information Science

Kagoshima University

March 2013

Study on reinforcement learning using Voronoi diagram in continuous state space

連続的な状態空間のボロノイ分割を用いた強化学習に関する研究

A Thesis submitted to the Graduate School of Science and Engineering, Kagoshima University, Japan, in fulfillment of the requirement for the degree of

Degree of Philosophy in Engineering

Accepted on recommendation of:

1. Dr. Shigeru Nakayama
   Professor
   Dept. of Information Science and Biomedical Engineering
   Graduate School of Science and Engineering
   Kagoshima University, Japan

2. Dr. Kunihiko Mori
   Professor
   Dept. of Information Science and Biomedical Engineering
   Graduate School of Science and Engineering
   Kagoshima University, Japan

3. Dr. Takayasu Fuchida
   Associate Professor
   Dept. of Information Science and Biomedical Engineering
   Graduate School of Science and Engineering
   Kagoshima University, Japan

Contents

List of Figures

List of Tables

Acknowledgements

First of all, I would like to express my profound gratitude to my academic supervisor; associate Professor Dr. Takayasu Fuchida for supervising me, providing guidance, and on time suggestions during all my graduate study. His generous help and affection gave me encouragement during the difficult times and assisted me to carry my research work to completion.

I would also like to express my appreciation to my committee members, Dr. Shigeru Nakayama and Dr. Kunihiko Mori for all useful advices, comments, discussions and correction about my research.

My sincere thanks go to all of my colleagues, teachers and staff members of Information and Computer Science Department for their valuable academic discussions and effective supports in necessary study material.

I am truly and deeply indebted to my (father) and mother, my beloved brothers and sisters for their love, inspiration, encouragement and endless moral support over the years.

Finally, I am sincerely very grateful to all my friends all over the world who always so generously provided and has helped me in different aspects of life. And I am also greatly appreciate to my Japanese host families and the people of Kagoshima for taking care of me, for their very kind hospitality and extremely courteousness, and make me having an unforgettable experience during my stay in Japan.

Abstract

There are several kinds of learning methods however most of the research tell us that reinforcement learning (RL) [1] is the most suitable method in machine learning that deals with the decision to take an action using an agent at discrete time steps, and it is expected that would be useful anywhere in the future. There are several ways to implement the learning process but Q-learning algorithm due to Watkins [2] is a policy for estimating the optimal state-action value (Q-value), and it is one of the most fundamental methods in RL. Q-learning can apply in many practical applications but it works only state and action are both discrete. It is difficult to treat in continuous state space because of the Curse of dimensionality problem.

This dissertation proposes VQE (Voronoi Q-value Element) to be able to apply the Q-learning in continuous state space and to solve the Curse of dimensionality problem by partitioning the state space. As a method of space division, we apply the Voronoi diagram which is a general space division. Nevertheless, Voronoi diagram has a lot of flexibility thus a method of position determination of VQEs becomes a problem. Therefore, we present the addition method of VQEs to decide the position and LBG algorithm is used for adaptive state transition vector grouping. In addition, we propose the integration method of VQEs to reduce the number of states and memory usage and Delaunay tessellation technique is used to find the adjacent VQEs. These proposed methods also aim to show the improvement of a learning efficiency.

In order to examine the efficiency of our proposed methods, we constructed the continuous states and discrete actions experimental model. The experiments are carried out compared with lattice of a previous work. The results indicate that the proposed methods are greatly improved than the previous method.

Chapter 1

Introduction

1.1 Historical Background

Nowadays, intelligent robots are applied in many fields. The intelligent robots have many potential applications in industry, medicine, and even service at home that make their study important. However, highly intelligent tasks are still difficult to be achieved by the robots. When the robots perform the tasks in an uncertain environment, searching the optimal behavior is very important. It is not easy to find the optimal behavior in the changing environment under various situations. There are several kinds of learning methods for the robots, such as supervised learning, unsupervised learning, and reinforcement learning [1].

The supervised learning learns from examples provided by a knowledgeable external supervisor, the unsupervised learning learns without external supervisor, and the reinforcement learning learns from the evaluated feedback information called the reinforcement signal (critic). In order to find the optimal behavior practically, reinforcement learning is the most suitable method. Reinforcement learning is studied in most current research in machine learning, statistical pattern recognition, and artificial neural networks [1]. One method of designing the agents that constitute a single-agent system is called reinforcement learning [1]. Since the distances between the robot (agent) and the obstacles or the target (reward-area) are not discrete values practically, the Q-Learning on continuous state space is applied to intelligent robots in this study.

Reinforcement learning work in statistics, psychology, neuroscience, and computer science, and it has attracted rapidly increasing interest in the machine learning and artificial intelligence communities nowadays. It’s effective that the human being controlling the action of the robot to learn autonomously, and it is the problem faced by an agent that must learn behavior through trial-and-error interactions with an environment.

Some aspects of reinforcement learning are closely related to search and planning issues in artificial intelligence. Learning algorithms based on an evaluative feedback signal are generally referred to as reinforcement learning (RL) algorithms, where the agent solves the given task based on rewards received from the environment. Otherwise, RL is a type of machine learning that deals with the decision to take an action using an agent that can run in certain environments. It’s one of the most important learning methods for intelligent robots working in unknown environments.

In the widely used of RL approaches, it constructs a learning agent using the Q-Learning method, which is a representative technique of RL. RL uses the numerical value by learning that is called Q-value. It’s the value of an action in a certain state. The action value that has the value of each action in a certain state is preserved in the table called Q-Table as shown in Figure 2.2. When the action in a certain state is executed, the expected value of the reward can obtain in the future. If the value of obtaining in the future is large, the amount of a certain reward Q-value is possible a good action. However, it tests various actions, and gradually will look for a good Q-value from the beginning because it doesn't know which action is a good action. The Q-values of the initial states are all initialized to the value zero.

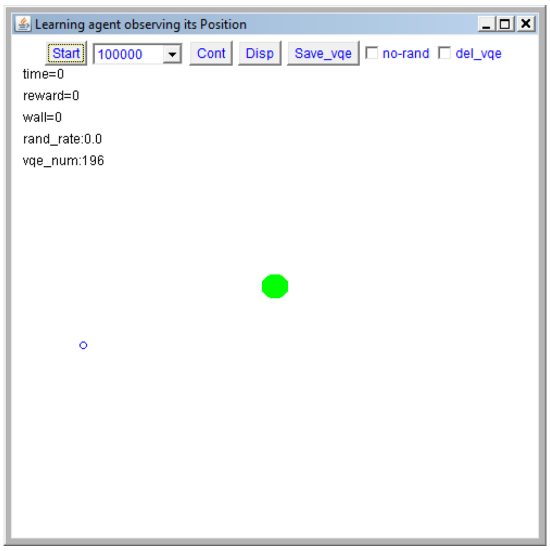
In particular, Q-Learning that is one of the most fundamental methods of reinforcement learning can apply in many practical applications because the optimal policy is guaranteed to be obtained if the learning environment is a discrete Markov decision process. However, it works only for discrete state space. It’s difficult to handle on continuous state space. In the case of Q-Learning is treated on continuous state space, the size of the Q-Table increases rapidly by Curse of dimensionality, and it’s not realistic. It’s one of the most serious problems in function approximation, and reinforcement learning. The curse of dimensionality also exists in a single-agent environment.

Curse of dimensionality problem means as the dimension numbers of state space increases, the state also increases exponentially when the state space is divided into lattice. On the other hand, as the number of state and action variables increases, the size of the Q-Table used to store Q-values grows exponentially. An enlargement of the state space in reinforcement learning causes the learning speed to decrease suddenly or causes an enormous amount of memory to be required. The various approaches to autonomously constructing a feature space have been investigated such as the wire fitting approach, tile coding with hashing, a tree-based algorithm, and a method based on the self-organizing map to solve the curse of dimensionality problem. These methods aim at effective construction of a feature space for function approximation.

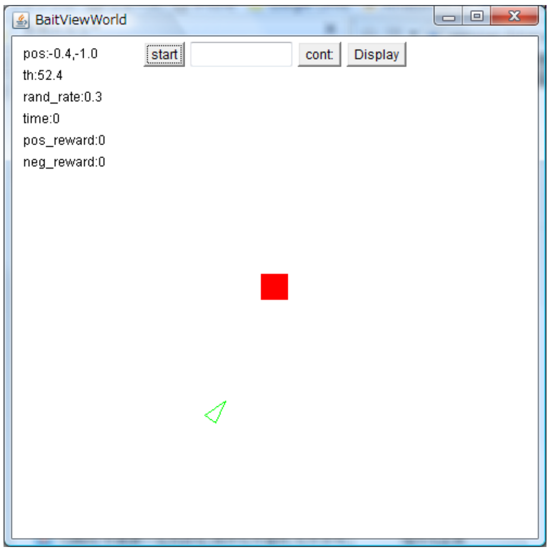
In a single-agent environment, the Actor-Critic method has been proposed for a continuous action space, and a state generalization method using support vector machines (SVM) [8] and state space hierarchy construction [11] have been purposed for a continuous state space.

1.2 Research Objective

The aim of the research is to apply the Q-learning method in continuous state-space using the concept of Voronoi space division. Therefore, we propose Voronoi Q-value Element (VQE) to solve the Curse of dimensionality problem also by partitioning the state space adaptively. As a method of Voronoi space division, Voronoi diagram is used because it is a general space division method. However, Voronoi diagram has a lot of flexibility, i.e., it has a high degree of freedom, thus a position determination method becomes a problem. Therefore, we present an addition method of VQEs to decide the position of VQEs using LBG algorithm for adaptive state transition vector grouping. Moreover, we present the integration method of VQEs to reduce the number of states



time=0
reward=0
wall=0
rand_rate:0.0
vqe_num:196



pos:-0.4,-1.0
th:52.4
rand_rate:0.3
time:0
pos_reward:0
neg_reward:0

Figure 1.1 : BugPos Model   Figure 1.2 : BaitViewWorld Model

and memory usage using the Delaunay tessellation technique for integration of adjacent VQEs. These proposed methods also aim to show the improvement of a learning efficiency based on Q-Learning in continuous state space.

In order to examine the proposed methods, 2-types of experimental model based on continuous states and discrete actions of feeder mouse (Esa-Hiroi Mouse) are constructed. These are non-coincidence of state space and action space model (Model I) we called “Bait View World” shown in Figure1.2, and another one is coincidence of state space and action space model (Model II), we called “Bug Pos” illustrated in Figure 1.1. There is the action space which the agent is learning toward the goal-area, and the state space which is trying to segment the continuous space as a Voronoi space division.

Additionally, we conducted the demonstration experiments using these 2-models and verified the effectiveness of proposed methods. We examined the performance of the following several different methods in a stationary situation of reward-area through the computer simulations. These are 1) lattice also called Q-Table, 2) one of proposed method which is addition method using LBG algorithm, and 3) integration method of VQEs using Delaunay tessellation technique on continuous state space.

Actually, there are many researches that the Q-Learning applied in continuous state space. The two references papers that are most closely related to our research are:

1) Q-Learning in continuous state and action spaces (1999) by Chris Gaskett, David Wettergreen, Alexander Zelinsky [3] but it considers the Q-Learning in both state and action spaces are continuous.

2) An adjustment method of the number of states on Q-learning segmenting state space adaptively (2003) by Tomoki Hamagami, Seiichi Koakutsu, and Hironori Hirata [4]. However, their system is different from ours. Our methods apply the Q-learning in continuous states and discrete actions using the concept of Voronoi space division.

1.3 Dissertation Overview

This thesis consists of 8 chapters. The contents of each chapter are as follows:

Chapter 1 starts with an introduction that explains briefly the historical background of motivation for learning, and discusses the curse of dimensionality. This chapter also describes the aim and objectives of the study, and specifies the dissertation structure.

Chapter 2 gives an overview or the basic concepts and ideas related to this research namely the basic principles for the reinforcement learning such as the standard reinforcement learning model. Section 2.2 defines a technique for people to realize the learning ability, and to automatically perform what kind of action should take by computer machine in order to maximize the expected value of future reward in unknown environment. Section 2.3 describes the problem of reinforcement learning, and defines some classic model-free algorithms for reinforcement learning from delayed reward: Markov decision process, and value function. In section 2.4, the most important aspects of normal Q-Learning algorithm which is a typical technique of reinforcement learning was described. Section 2.5 also describes the action selection method of agent, and section 2.6 describes the Q-Table which divides the state space into lattice and the division method of Q-Table. Moreover, section 2.7 discusses the curse of dimensionality which increases the number of states exponentially in high dimensions. Furthermore, section 2.8 gives and shows the result of Q-Block that execution times takes about twice than Q-Table when we used Q-Block to solve the curse of dimensionality problem.

Chapter 3 describes the VQE (Voronoi Q-value Element) which divides the state space using the concept of Voronoi space division in order to solve the above problem, and also gives the idea of Voronoi diagram, and how Voronoi diagram could be used to partition the state space. Although Q-Table needs to prepare Q-value in all states beforehand, VQE can be added to the state space as required. Section 3.1 describes the Voronoi division which is the division method of space whether arbitrary points being the closest to which mother point with respect to the mother point located on space. Section 3.2 presents the creation method of VQE, a reference method of VQE, method of space division using VQEs, and the advantages of Q-Learning that used VQE are enhanced learning speed and reliability for this task, and the essential characteristics of VQEs in a continuous state space are also described. This chapter also explains several methods of nearest neighbor search.

Chapter 4 evaluates the effectiveness of proposed Q-Learning technique by using VQEs, and performs the computer simulations as a comparison experiment of Q-Table that described in previous section 2.6 and VQEs. And 2-types of experimental models which are Model I and Model II are explained in Section 4.2 for full details. These 2-models are based on continuous states and discrete actions of feeder mouse (Esa-Hiroi Mouse). After that, we show the better performance using VQEs on continuous states and discrete actions for 4-dimensional spaces by comparing the normal Q-Learning (Q-Table) and Q-Learning with the use of VQEs. In addition, the conclusions are considered.

Chapter 5 examines the learning performance of various strategies using 2-types of experimental model I and model II with reward-area in a stationary situation in single-agent environment and decide how to act in certain state. In order to test our hypotheses, we experimented by rotating the angles of agent’s actions, angles of VQEs by the angle in 5 times interval between 0 degrees and 90 degrees in which VQEs are arranged in a lattice structure. Moreover, a random arrangement of VQEs experiment also conducted to correctly evaluate the optimal Q-values for state and action pairs in order to deal with continuous-valued inputs. As a result of experiments using experimental model II, the learning speed has most increased when the angles of VQEs and angles of actions is just 45 degrees out of alignment in case of 4-actions.

Chapter 6 presents the addition method of VQEs which is a position determination method to decide the position of VQEs in order to realize a Voronoi region since the performance of Q-Learning changes according to the arrangement of VQE. Moreover, the simulation was performed in both experimental models and the learning performance was examined. And also presents block-counting method and a new adaptive segmentation of continuous state space based on vector quantization algorithm such as LBG (Linde-Buzo-Gray) for high-dimensional continuous state spaces. The objective of adaptive state space partitioning is to develop the efficiency of learning reward values with an accumulation of state transition vector (STV) in a single-agent environment. Moreover, the study of the resulting state space partition reveals in a Voronoi tessellation. In addition, the experimental results show that this proposed method can partition the continuous state space appropriately into Voronoi regions according to not only the number of actions, and achieve a good performance of reward based learning tasks compared with other approaches such as square partition lattice on discrete state space.

Chapter 7 describes an algorithm of integration of VQEs to reduce the number of states, the memory usage and the learning time. It also aims to improve the performance of learning efficiency. Then it proceeds and described the topological structures of Delaunay network to find the adjacent VQEs for integration on continuous state space. We add VQEs on state space, and integrate which has the same optimal action selections. A computer simulation has been performed using experimental Model I, and the simulation results are explained compared with 3-methods such as lattice of a previous method which is Q-Table, addition method of VQEs, and integration method of VQEs with the reward-area in a stationary condition only.

Chapter 2

Reinforcement learning and Q-Learning

2.1 Basic Principals

In this chapter we will introduce a review for the basic concepts of machine learning as reinforcement learning and the various representation of the Q-Learning, Markov-decision process, curse of dimensionality, state space division method and the agent's action selections are described.

2.2 Machine Learning and Reinforcement Learning

The performance and computational analysis of machine learning algorithms is a branch of statistics known as computational learning theory. Machine learning is about designing algorithms that allow a computer to learn. Learning is not necessarily involves consciousness but learning is a matter of finding statistical regularities or other patterns in the data. Thus, many machine learning algorithms will barely resemble how human might approach a learning task. However, learning algorithms can give insight into the relative difficulty of learning in different environments.

There is machine learning that is one of the research tasks in the field of Artificial Intelligence to perform the learning ability automatically and the same functions a

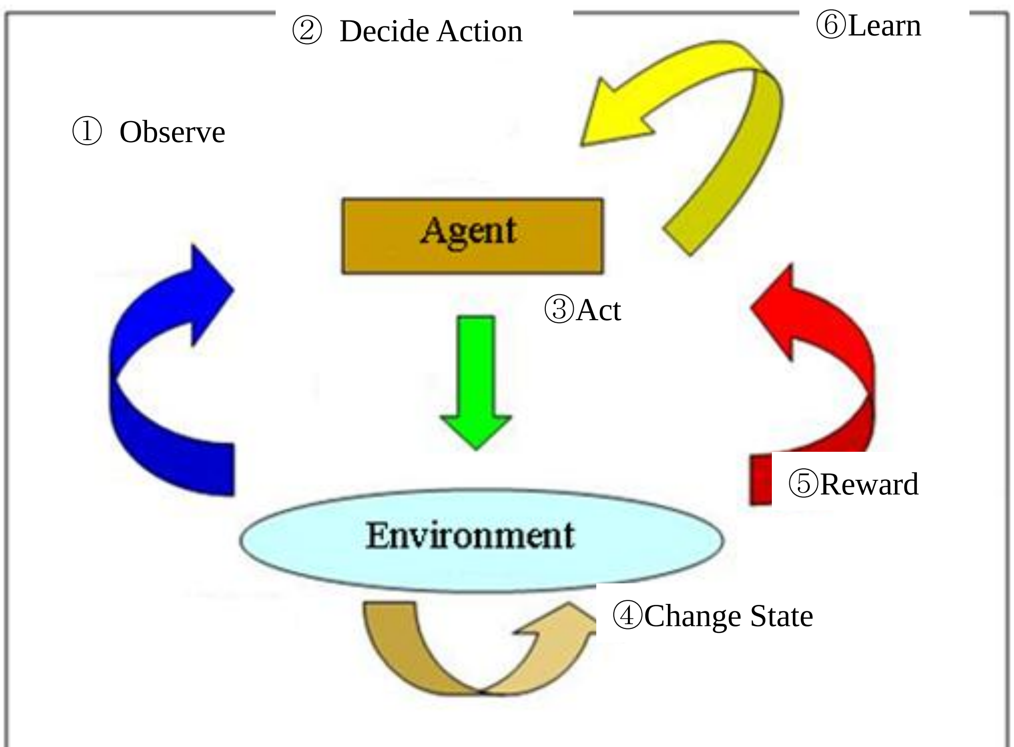


Figure 2.1 : Flowchart of Reinforcement Learning (RL)

reality by computer as technology and a technique. Machine learning is generally classified into *supervised learning* which learns by giving the input so as to policy of back propagation and support vector machine which is a classical multilayer perceptron neural networks, *unsupervised learning* which learns without giving those like clustering, and *reinforcement learning* (RL).

Reinforcement Learning (RL) is a type of Machine Learning, and thereby also a branch of Artificial Intelligence. It is the most suitable method in machine learning that deals with the decision to take an action using an agent at discrete time steps and it is expected that would be useful anywhere in the future [1]. RL methods attempt to improve the agent's decision-making policy over the time. The agent's goal is to get as much reward as it can over the long run. Moreover it allows machines and software agents to automatically determine the ideal behavior within a specific context, in order to maximize its performance. The goal of RL is to figure out how to choose actions in response to states so that reinforcement is maximized. That is, the agent is learning a policy, a mapping from states to actions. The agent's policy is divided into two components, how good the agent thinks an action is for a given state and how the agent uses what it knows to choose an action for a given state. Simple reward feedback is required for the agent to learn its behavior this is known as the reinforcement signal. In the general case of RL problem, the agent's actions determine not only its immediate reward, but also the next state of the environment. The agent will have to be able to learn from delayed reinforcement. In the problem, an agent is supposed decide the best action on the current state. When this step is repeated, the problem is known as a Markov Decision Process.

In the standard RL model, an agent is connected to its environment via perception and action, as depicted in Figure 2.1. RL deal with the issue of the agent within a certain environment observes the present state, and decides the next action to be taken. It is performed by the interaction of agent and environment, and learning progresses by repeating a series of flows from the state observation to the learning. TD (Temporal Difference) Learning, Q-Learning which is online learning of control strategies when next state function is unknown learning, etc. are known as a typical technique of RL.

Figure 2.1 shows the interaction of agent and environment at discrete time steps ***t*** in learning process. A system with RL comprises two elements: an agent and an environment. The agent repeatedly observes the state variable of its environment, and chooses an action. The environment changes the state by the agent's behavior, and returns the reward. The agent takes the reward from the environment, and updates a policy accordingly, and outputs the action to the environment. After that, the state changes from the current state to the next state, and the agent learns based on the reward by its own action policy.

1. State Observation (Observe)

   The agent first observes the environment and obtains the state.

2. Action Decision (Decide Action)

   The agent then decides the action based on the state that was obtained by state observation.

3. Act

   The agent performs the decided action with respect to the environment.

4. State Transition (Change State)

   The environment influences by acting the agent, and a state is changed.

5. Reward

   The environment returns the reward to the agent that corresponds to the current state after changing the state.

6. Learn

   Finally, the agent learns by receiving the reward.

RL learns the optimal action by repeating this process from 1~6, and the agent accumulates the value of Q during the learning period. The Q-value is expected value of the returns, which is discounted sum of the rewards that agent received.

2.3 Problem of RL

2.3.1 Proceeding

The reinforcement learning aims at maximizing the final accumulation of remuneration obtained from environment. This accumulation of remuneration is given by the following formulas.

$$R_t = \sum_{k=0}^{T} \gamma^k r_{t+k+1}$$

Here, T is the last time, γ is a numerical value (reward) of whether to carry out the consideration of the remuneration that can obtained in future however.

2.3.2 Markov Decision Processes

In RL study, an environmental state is needed for an agent in the case of action determination and learning. This state is generally modeled by the Markov decision processes, also known problems with delayed reinforcement are well modeled as *Markov decision processes* (MDPs). An MDP consist of a set of states, a set of actions, a reward function, and a state transition function. The agent observes state and chooses action then receives reward, and state changes to next state. A state transition function probabilistically specifies the next of the environment as a function of its current state and the agent's action. The reward function specifies expected instantaneous reward as a function of the current state and action. The model is *Markov* if the state transitions are independent of any previous environment states or agent actions. There are many good references to MDP models (Bellman 1957; Bertsekas 1987; Howard 1960; Puterman 1994). The general MDPs may have infinite (even uncountable) state and action spaces.

The probability that the phenomenon in the future will happen is decided only from the present state and the character in which it does not depend is call the Markov nature to the previous state. Generally it is the action that taken by the time t. Since all the phenomena which happened in the past are related, the response to set $t+1$ is defined as follows. On the other hand, it will be time t f a state signal has the Markov nature. A response is time t, since it is depend only on the state where state and action pair set defines as follows.

$$\Pr\{s_{t+1} = s', r_{t+1} = r \mid s_t, a_t\}$$

Here, if the environment has the Markov nature, the present state and the state of action to the next, and action can be predicted. Furthermore, all the future states and actions can be predicted form the present by repeating it. The RL study which fills the Markov nature is called the Markov decision process. If the space of a state and action is limited, it will be called a limited Markov decision process. When the arbitrary states and action are given, it will be a next state possible and the probability is given by the following formula.

$$P_{ss'}^{a} = \Pr\{s_{t+1} = s' \mid s_t = s, a_t = a\}$$

Moreover, when the present state and action is given the expected value of reward is given by the following formula.

$$R_{ss'}^{a} = E\left\{r_{t+1} \mid s_t = s, a_t = a, s_{t+1} = s'\right\}$$

2.3.3 Value Function

A reward which carries out the target of RL evaluating remuneration and maximizing the expected value of reward which defines a value function in order to measure whether it is worthy of the present state being how much in that case. Firstly, the action a takes the state by the policy π is assumed as $\pi(s,a)$.

$$V^{\pi}(s) = E_{\pi}\left\{R_t \mid s_{t=s} = s\right\} = E_{\pi}\left\{\sum_{k=0}^{\infty}\gamma^{k} r_{t+k+1} \mid s_t = s\right\}$$

It carries out the basis value by this policy π and can be expressed as follows.

$$Q^{\pi}(s,a) = E_{\pi}\left\{R_t \mid s_{t=s} = s, a_t = a\right\} = E_{\pi}\left\{\sum_{k=0}^{\infty}\gamma^{k} r_{t+k+1} \mid s_t = s, a_t = a\right\}$$

The value function has the basic property of recursive relations.

$$\begin{aligned}
V^{\pi}(s) &= E_{\pi}\left\{R_t \mid s_{t=s} = s\right\} \\
&= E_{\pi}\left\{\sum_{k=0}^{\infty}\gamma^{k} r_{t+k+1} \mid s_t = s\right\} \\
&= E_{\pi}\left\{r_{t+1} + \gamma\sum_{k=0}^{\infty}\gamma^{k} r_{t+k+2} \mid s_t = s\right\} \\
&= \sum_{a}\pi(s,a)\sum_{s'}P_{ss'}^{a}\left[R_{ss'}^{a} + \gamma E_{\pi}\left\{r_{t+1} + \gamma\sum_{k=0}^{\infty}\gamma^{k} r_{t+k+2} \mid s_{t+1} = s'\right\}\right] \\
&= \sum_{a}\pi(s,a)\sum_{s'}P_{ss'}^{a}\left[R_{ss'}^{a} + \gamma V^{\pi}(s')\right]
\end{aligned}$$

It is called the Bellman equation and this equation is a state s worth of the remuneration in all succession discounted. It is carried out by occurrence probability to all the actions.

2.4 Q-Learning

There are several ways to implement the learning process. However, Q-Learning is one of the policy types of TD learning proposed by Watkins in 1989 [1]. It's based on the idea that the expected discounted sum of future reinforcements can be estimated by a function of each action in each state, which gives the value of Q to the pair of a state and action, and can be used to define an optimal policy. It's one of the most fundamental methods, and the most popular. It has been proposed for the intelligent robots to find the optimal behavior. It seems to be the most effective model-free algorithm for learning from delayed reinforcement, and can apply in many practical applications. However, Q-Learning is generally considered in the case that states and actions are both discrete. It's difficult to handle on continuous state space because of Curse of Dimensionality problem that explain in section 2.7.1. It needs to discretize the state space into a lot of smaller discrete regions when the case of continuous state space is treated.

In general Q-Learning, every action and state pair have their own Q-value. Q-values store in a table is called Q-Table and it looks like a square lattice in 2-dimensions. These values are initialized to small random numbers, and gradually change toward the optimal values through learning. Q-values are used to predict the discounted cumulative reinforcement for each state-action pair because of the agent learns a mapping form states and actions to their Q-values.

In the simplest case, the Q-value for a state-action pair is the sum of all reinforcement signals, and the Q-Learning function is the function that maps from state-action pairs to values. But the sum of all future reinforcements may be infinite when there is no terminal state. In Q-Learning, the selected action a of Q-value in time t is updated by following equation.

＜Q-Learning Equation＞

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]$$

2.4.1 Q-Learning Algorithm

A state-action value (Q-value) which is denoted by $Q(s_t, a_t)$ is updated just by taking the one with the maximum Q-value for the current state. For each state-action pair,

1. Observe the current state s_t.
2. Select an action a_t using ε-greedy action selection method and execute it.
3. Receive an immediate reward from the environment.
4. Observe a next state s_{t+1}.
5. Update the Q-value for $Q(s_t, a_t)$ state-action pair using above Q-Learning equation.
6. Increase the time t to t+1 and go back to step (1) and repeat the process.

In this equation, a state in time $\boldsymbol{t}$ is $\boldsymbol{s_t}$, Q-value of action a_t in the state $\boldsymbol{s_t}$ is $Q(s_t, a_t)$, it tell us the immediate reward for making a good an action is given a certain state. Moreover, α is the learning rate, γ is the discount rate, r_t is a reward. Furthermore, $\max_a Q(s_{t+1}, a)$ shows the maximum Q-value of next state in time $t+1$. This equation means if the Q-value of next state is greater than the current Q-value, it increases the current Q-value. Conversely, if the Q-value of next state is lesser than current Q-value, it decreases the current Q-value.

2.5 Action Selection Method

Since all state and action pairs must be chosen for the sufficient number of times as what kind of action selection method may be used for convergence of Q-learning. However, in order to gain more reward the following techniques are used in many cases.

2.5.1 ε-Greedy Method

This method chooses the random action in small probability ε . If it does not take the random action, it selects the maximum Q-value by taking the highest action. In other words, it chooses the action a that will become $\max_a Q_t(a)$ at time t .

Probability of random action	:	ε(ε＜1)
Probability of maximum Q-value action	:	1-ε

2.5.2 Roulette Selection Method

It is a selection method of individual i to choose the probability when it assumes as p_i

$$p_i = \frac{f_i}{\sum_{k=1}^{N} f_k}$$

A substance of adaptive value is expressed as $f_{i,}$ and it is the requisite that adaptive value does not take a negative value. Since the probability that a substance or individual organism with high adaptive value will be chosen when the value which is carried out scaling of the adaptive value in fact is used in many cases.

2.5.3 Softmax Technique

This technique is given the high selection probability in good action, and as for other, it is given to the high order selection probability according to the presumed value. The Boltzmann distribution is used as the function. A selection probability $\pi(s,a)$ of an action a in state s is expressed as

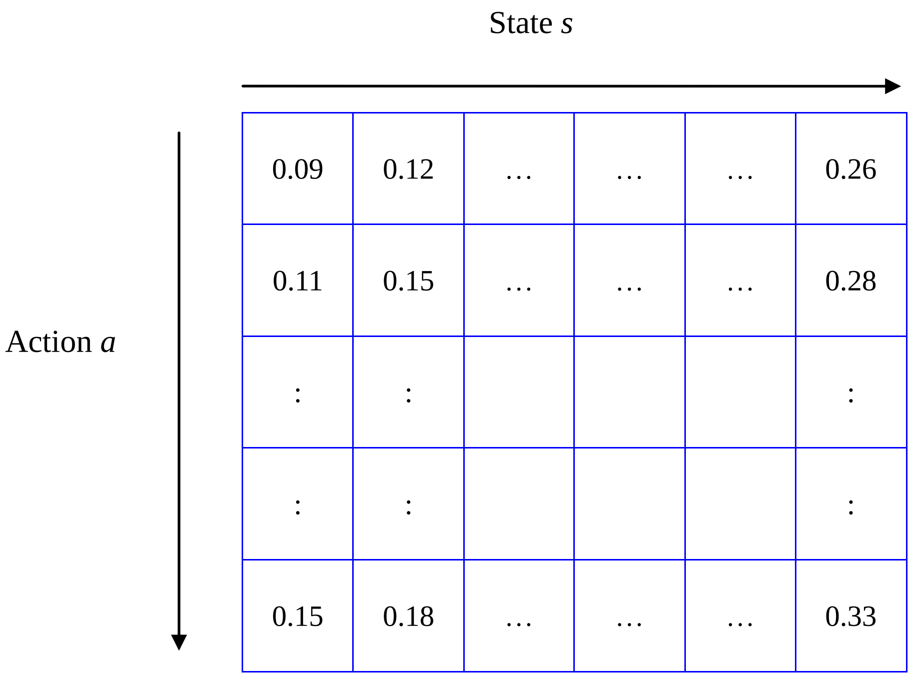
$$\pi(s,a) = \frac{\exp\left(Q(s,a)/T\right)}{\sum_{p\in A}\exp\left(Q(s,p)/T\right)}$$

Here, T is a positive constant, A is a set of the possible action in state.

2.6 General state space partition method in Q-Learning

2.6.1 Q-Table

The expected value (Q-value) of each action in each state is stored in a table that used in Q-Learning is called a Q-Table, which looks like a square lattice in two dimensions. Q-value is the discounted sum of the rewards agent that receives for a state and action pair. But if the state space is too large then it would be impossible to store all the Q-values. An example of Q-Table is shown in figure 2.2, and figure 2.3 also shows an example of Q-value selection in Q-Table.


State *s*
Action *a*

0.09	0.12	…	…	…	0.26
0.11	0.15	…	…	…	0.28
:	:				:
:	:				:
0.15	0.18	…	…	…	0.33

Figure 2.2 : An example of Q-Table

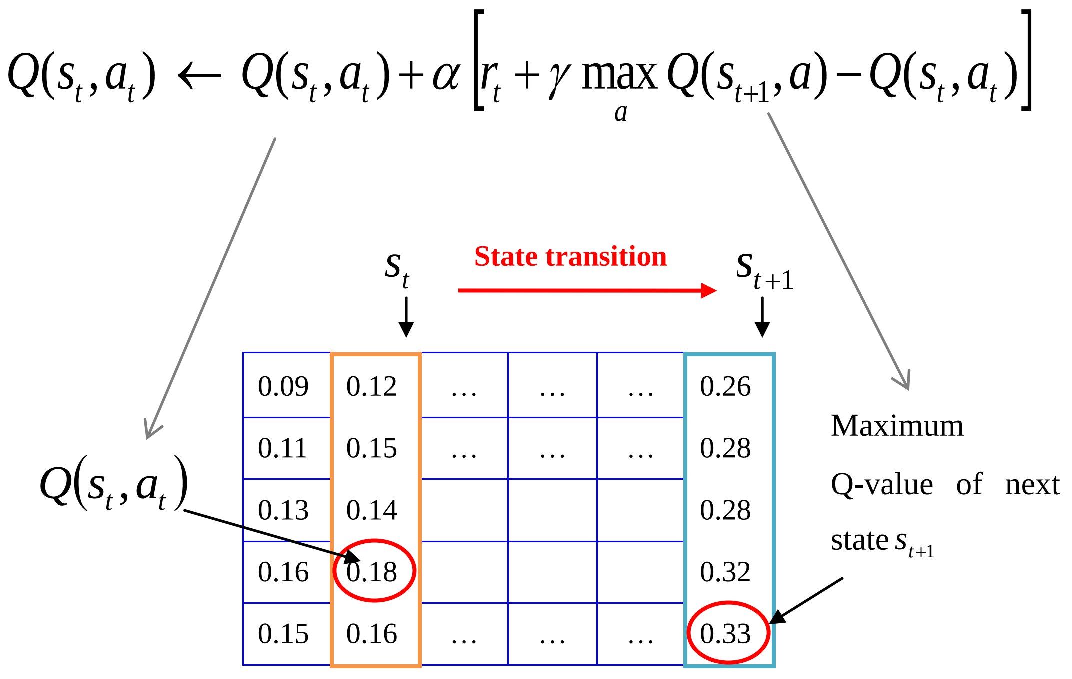

$Q(s_t,a_t) \leftarrow Q(s_t,a_t)+\alpha\left[r_t+\gamma \max_a Q(s_{t+1},a)-Q(s_t,a_t)\right]$
s_t
State transition
s_{t+1}
$Q(s_t,a_t)$
Maximum Q-value of next state s_{t+1}
0.09 0.12 … … … 0.26
0.11 0.15 … … … 0.28
0.13 0.14 0.28
0.16 0.18 0.32
0.15 0.16 … … … 0.33

Figure 2.3 : An example of Q-value selection in Q-Table

2.6.2 Space-Division method of Q-Table

A segmentation of space using normal Q-Learning is shown in Figure 2.4. Q-Table divides the entire all state space into the shape of square discrete regions like a “lattice” to treat the case of normal Q-learning on continuous state space. It finely divides the state space into very small tiny discrete regions thus the size of all state are equal. Dividing the state space into many smaller discrete regions like a lattice, the values of Q are evenly arranged all over the space surface. Therefore this subdivision method can possibly generate the waste space because it needs to divide the all space surface even the position which is not used Q-value. It means Q-Table use the Q-value in all states. Thus, Q-Table takes a long time to learn to get a reward and it is hard to partition the space partially. Furthermore, it has Curse of dimensionality problem explained in section 2.7.1.

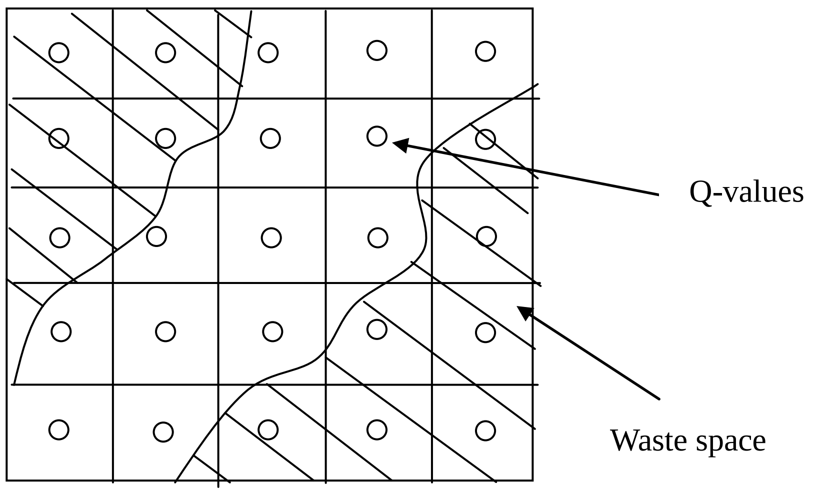


Figure 2.4 : Example of space-division method of Q-Table

2.7 Q-Learning Problem

2.7.1 Curse of Dimensionality

The curse of dimensionality also exists in a single-agent environment. When reinforcement learning is applied in an actual environment, the action outputs are continuous values, especially for robot control, and the state space is given according to continuous values. With a method that performs learning by discretizing the continuous values, the size of the action space and state space will explode. In an environment where the curse of dimensionality occurs, a great deal of time will be required for an agent to search the learning space, and the learning speed will end up decreasing significantly.

Even in a single-agent environment or 2-dimensional state space, when reinforcement learning is used in an actual environment, the decrease in learning speed is problematical. Therefore, multi-dimensional state space requires even more learning time than learning in a 2-dimensional state space. In addition, it also has the important problem that the increase in the number of dimensions or number of agents will cause the curse of dimensionality. This is because the state space is constructed with the input state or all agents as part of the environment, and as the number of dimensions or as the number of agents increases, the state space grows exponentially, which results in the size of state space exploding.

Curse of dimensionality also means, as the number of dimensions or agents increases, the size of the Q-Table to store Q-values grows exponentially and it's not realistic. An enlargement of the state space in reinforcement learning causes the learning speed to decrease suddenly or causes an enormous amount of memory to be required.

2.8 Previous Work

2.8.1 Q-Block

There are so many state where it is not used in division of Q-Table exists in many cases although the learning progresses as described in above section 2.6.2. And if we use Q-Table in high dimension, the number of states increases exponential to create all states at first and may occur the curse of dimensionality, and a memory may be

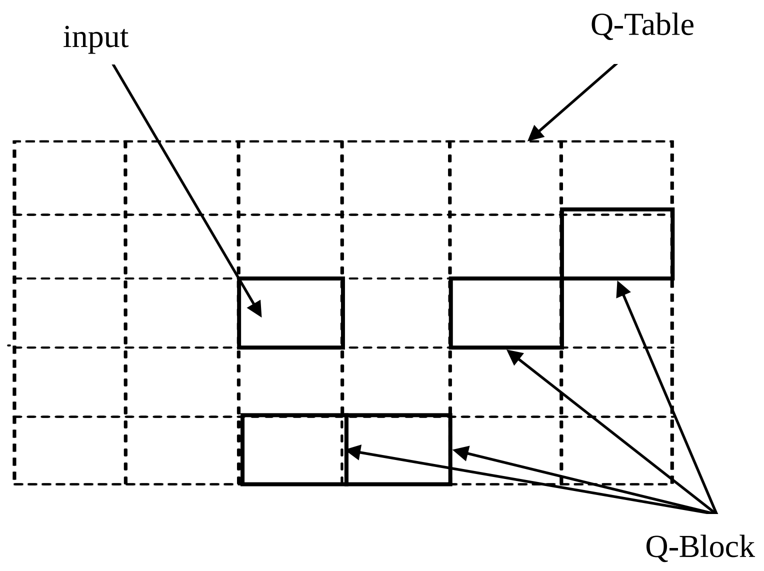


Figure 2.5 : An example of Q-Block

insufficient. Then a technique of creating the element of Q-Table that only require for learning is proposed. Q-Block is generated as one element of a table when Q-value is needed. First, an agent generates the element of Q-Table corresponding to the state where it needed in state observation and action determination, and stores in a hash table. Next, it is referred to when the state where it is referred to exists in a hash table. It enables it for a state to reduce the number of states generated by the memory into a high dimension, and to use a lattice-like separation technique.

However, according to the results of our experiments in Q-Learning using Q-Block, the number of obtained rewards is equivalent to the previous study which used Q-Table. Although, Q-Table cannot perform execution over 8-dimensions because of insufficient memory and the number of states could be reduced, Q-Block can apply to 40-dimensions were checked. However, it resulted the Q-Learning using Q-Block took a time nearly twice compared with Q-Table.

Since we seen decrease in performance as the dimension grows up in execution time, we need to consider another method that raises the learning efficiency for space division such as a required place which used Q-value divides finely but the place which is not required is not used.

Chapter 3

A proposition of VQE (Voronoi Q-value Element)

3.1 Voronoi Diagram

Q-Learning is an effective learning method regarding the discrete state space. Nevertheless, it needs to discretize the state space in the case of Q-Learning treats on continuous state space but it has the problem that is not realistic in normal Q-Learning. And, it is difficult to adjust the number of state in Q-Table depending on the necessary adjustments of addition and deletion of an element because the normal Q-Learning separates the state equality.

Therefore, we proposed VQE (Voronoi Q-value Element) to treat on continuous state space using the general idea of Voronoi space division. As a method of Voronoi space division, Voronoi diagram is used in our approach to partitions the space into cells or a number of convex regions, where each region consists of all points that are closer to one site than to any other. It has nearest-neighbor searches property but it cannot able to use over 2-dimensions. As a simple illustration of Voronoi diagram is described in figure 3.1.

Voronoi diagram records information about what is close to what. It is a partition or a subdivision of the plane into ***n*** cells or convex regions in terms of a given discrete

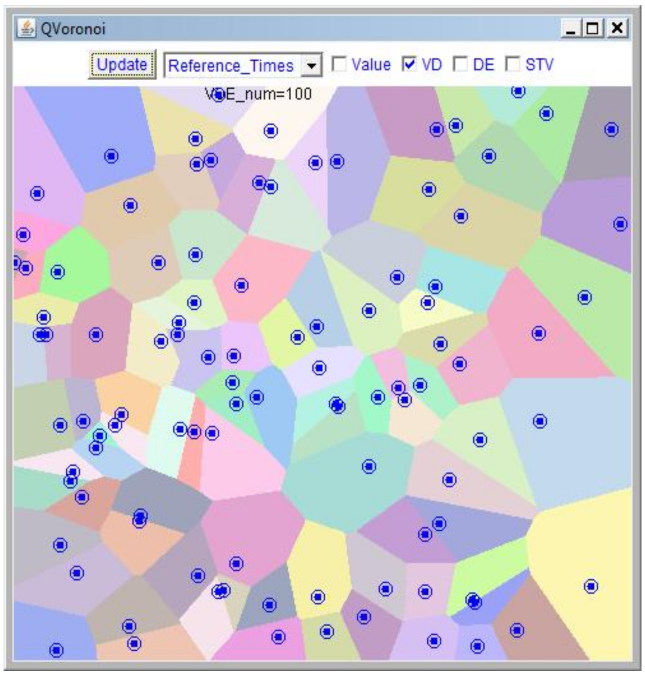


Figure 3.1 : Illustration of a Voronoi Diagram in a random set of points

set of points. It's a segmented shape divided by the mother point or an arbitrary point, however VQE is used here in our method. The partition of the plane formed by the closest point Voronoi regions. In other words, if ***S*** is a given set of points in the plane, and each point of the plane is associated with the nearest point of ***S***, then the plane is divided into convex polygons, or cells containing exactly one member of ***S*** such that for each point. Such a partition is called a Voronoi tessellation, also called Voronoi Diagram. If **S** is generated randomly or start from a random set of points, the result is a random Voronoi Diagram.

Voronoi cells can also be defined by measuring distances to objects that are not points. Voronoi diagram are often not feasible for over 2-dimensions due to its exponentially increasing size but it has the property of solving the waste of spaces and nearest neighbor search problem because one of the most important data structures problems in computational geometry is solving nearest neighbor queries.

Figure 3.1 shows an example of java applet animation of Voronoi diagram in a random set of points. A set of point is given as input, and output is a partition of the plane into regions of equal nearest neighbors. VQEs are marked with a black point, and the colored areas are the Voronoi region that belongs to the black point. Voronoi regions (cells) are bounded by line segments and Vorornoi region of a point is unbounded if and only if the point is a vertex of the convex hull of the point set. Voronoi edge is a bisector of two sites whose regions are adjacent. It can see that Voronoi Diagram is closely related to Delaunay Triangulation as well as to the convex hull of its projection onto some paraboloid. In both cases the connection is made with the use of duality. We will discuss it later when we talk about the dual structure called a Delaunay triangulation.

3.2 A Proposition of VQE in Continuous State Space

3.2.1 VQE (Voronoi Q-value Element)

Voronoi Q-value element (VQE) is a point that corresponds to the region or area. It has Q-value with regards to each discrete action. In the Q-Learning with the use of VQE, it was just creates the VQE on each agent's action. Figure 3.2 shows 100 random numbers of VQEs are arranged on the state space. The blue points belong to the point of VQE.

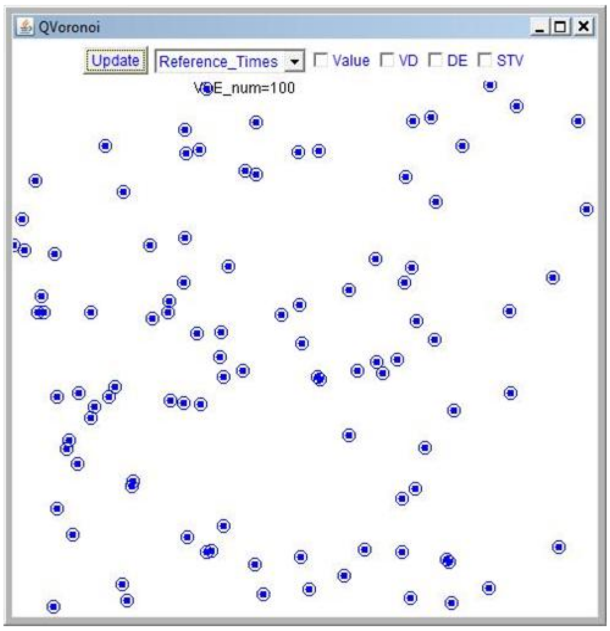


Figure 3.2 : Example of 100 random VQEs on state space

3.2.2 Creation Method of VQE

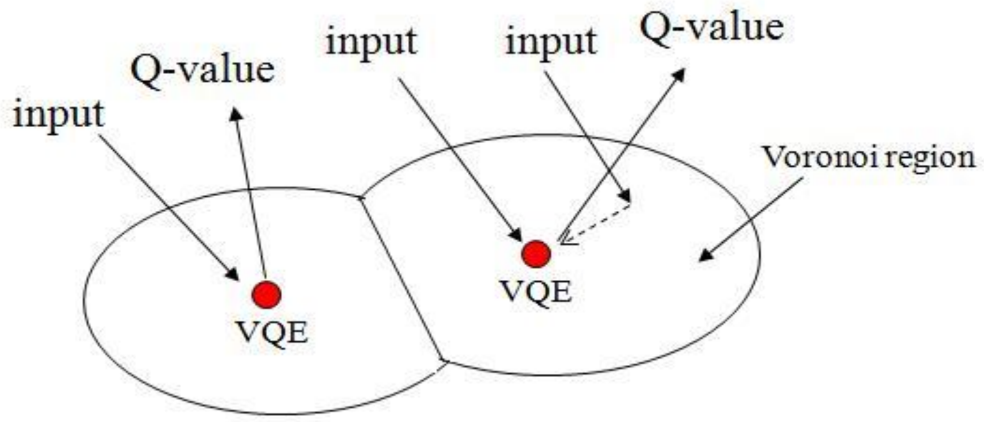


Figure 3.3 :　Creation method of VQE

An example of creation of VQE is when the input enters on the state space, VQE is created and it returns the Q-value. Moreover, in the case of more than one inputs enter within the area of VQE, it searches the shortest distance from VQE, and Q-value of nearest VQEs from that point is returned. The image of creation method of VQE is shown in Figure 3.3 and 3.4.

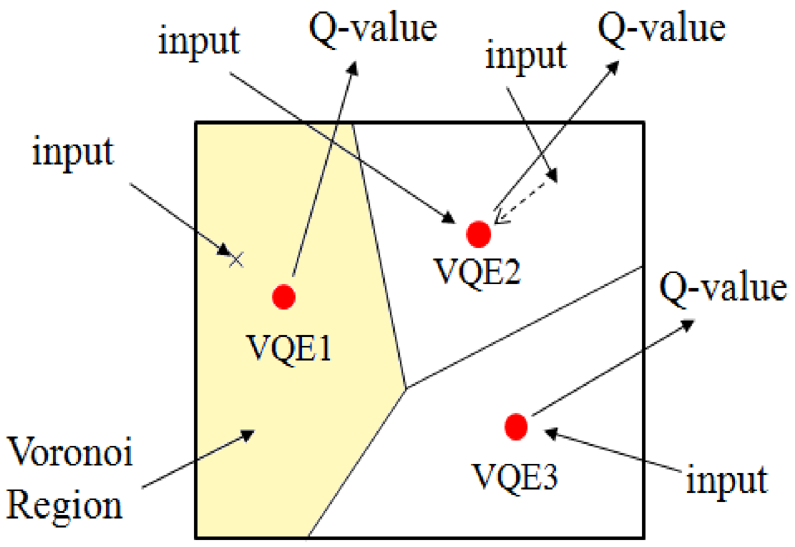


Figure 3.4 :　Example of state space division using VQEs

3.2.3 Method of Space-Division using VQEs

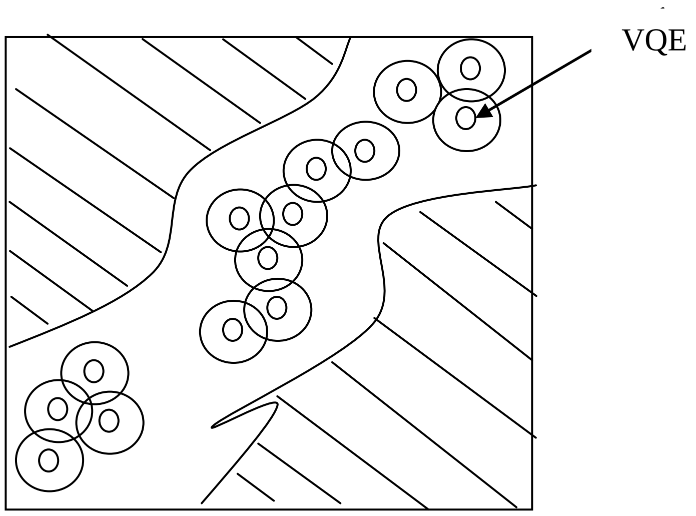


Figure 3.5 : Example of space-division of VQEs

The state space is divided into Voronoi region which is only required to divide in space division method of VQEs, and it just only uses the Q-value where there are lots of inputs as shown in Figure 3.5. Although Q-Table needs to prepare Q-value in all states beforehand, VQE can be added to the state space as required.

3.2.4 Advantages of VQEs

There are some advantages of VQEs. These are described as follows:

1. VQE can reduce the waste of spaces.
2. It does not have overlapping area within the region of VQE when partitioning the state space.
3. If data are deleted, other data can supplement the space in the erased part when data are deleted.
4. It is easy to integrate.

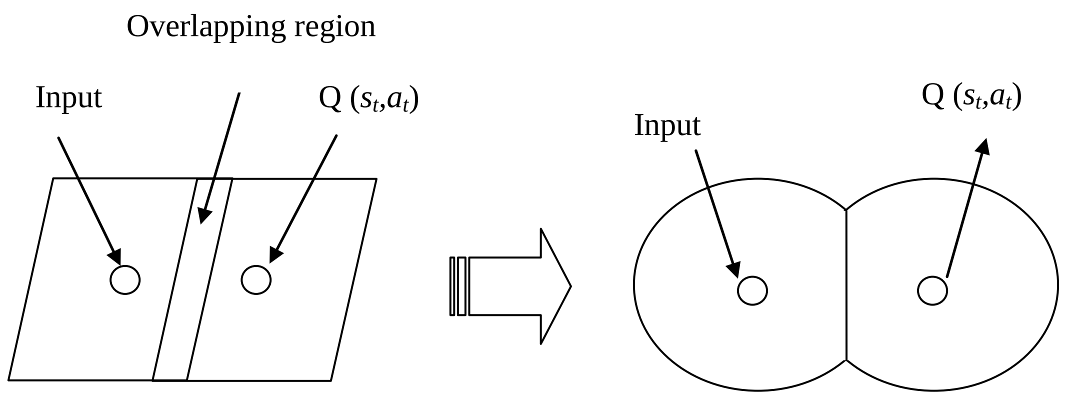


Figure 3.6 : Differences of Q-Block and VQE for Voronoi space division

3.2.5 Advantages of Q-Learning using VQE

Furthermore, there are some advantages of Q-Learning using VQEs. These are:

1. The number of states and the size of state can adjust by executing the addition and deletion of VQE.
2. It is thought that the state which is not required for learning is treated in a rough way, and the required state can reduce the number of states, and can save the memory usage.
3. Q-Table divides the state space in a lattice structure thus the size of all states will become the same size. Therefore, it is difficult to adjust the state partially.
4. In Q-Learning using VQE, it can solve the problem of increasing memory usage when the number of states is increased.

3.3 Nearest-Neighbor Search Method

In this section, we investigate a number of nearest neighbor search methods. Nearest-neighbor searching is a fundamental problem design of geometric data structures. It defined as a given collection of ***n*** points build a data structure which given any query point and reports the data point that is closest to the query. It has applications in many areas, including data mining, pattern recognition and classification, machine learning and data compression are some examples. Many data structures have been proposed for nearest neighbor searching.

We want to find the nearest neighbor of a given query vector without computing all distances. However, there was a problem to search time exponentially increase if the dimension goes up, and the Curse of dimensionality problem exact search inefficient therefore we try to use the method of MD-Tree, LSH (Locality-Sensitive Hashing), ANNS (Approximate Nearest Neighbor Searching) to obtain the nearest VQE in a given point. Then, we implement the programs using the algorithm of those techniques, and apply those. Each search method is specified by a data structure for storing the data and algorithms for building, and searching the structure. For example, given a distance function ***D***, a collection of points ***B*** (in ***k***-dimensional space), and a point ***P*** (in that space), it is often desired to find ***P***'s nearest neighbor in ***B***.

3.3.1 MD-Tree (Multi-dimensional Tree)

A multi-dimensional tree (MD-Tree) is one of the multi-dimensional management structures, and developed by extending the concept of the B-Tree data. MD-Tree can be used for improving the balance factor and to high the storage utilization and the algorithm of insertion and deletion methods are included in the structure. The special characteristics of MD-Tree are as follows:

(1) The processes of insertion and deletion data are high-speed.

(2) It can adjusts the balance of level and keeps high memory storage utilization.

(3) There are no overlaps and no needs to periodical restructuring of the tree structure by not integrating internal nodes on its delete process.

(4) It can combine the delete-insert algorithm in each data cut down the processing data.

MD-Tree has two novel concepts: internal leaf and improvements in bottom-up search. MD-tree has two novel concepts, internal leaf and improvements in bottom-up search. The internal leaf that is managed by corresponding internal node in a tree has pointers to moving objects and helps reduce the update cost of the tree. The improved bottom-up search of the tree reduces the retrieval costs by managing the non-overlapped areas of split data space.

3.3.2 LSH (Locality-Sensitive Hashing)

Locality Sensitive Hashing (LSH) is a widely-used algorithmic tool which brings the classic technique of hashing to geometric settings in many existing approaches, and it provides some guarantees on the search quality for some distributions. It has many variants, some examples are Hamming space [Gionis, Indyk, Motwani, 99], Euclidean version (E2LSH) [Datar, Indyk, Immorlica, Mirrokni, 04], Leech Lattice Quantization [Andoni, Indyk, 06]] and Spherical LSH [Terasawa, Tanaka, 07], etc. It is an algorithm for solving the approximate neighbor search in high dimensional spaces. The basic idea is to hash the input items so that similar items are mapped to the same buckets with high probability.

3.3.3 NNS (Nearest Neighbor Searching)

Nearest Neighbor Searching (NNS) also known as proximity search or closest point search, is an optimization problem for finding closest points in metric spaces. It is a technique which greatly reduces processing time and required amount of memory for nearest neighbor search.

Chapter 4

The experiments of Q-Table and VQE

4.1 Introduction

In this chapter, we examined the efficiency of proposed VQEs and Q-Table also called lattice which is an existing method. In order to evaluate the effectiveness of proposed VQE, very simple and efficient techniques of experimental models are constructed and computer simulation has been performed. We first explain two types of experimental model in this sub-section 4.2. Then the experimental parameters are explained. Moreover, we show the result of computer simulations on 4-dimensional spaces and the effectiveness of the proposed method.

4.2 Experimental Model

In order to examine the proposed methods, there are two types of experimental model. These are as follows:

Experimental Model I : Non-coincidence of State Space and Action Space model

Experimental Model II: Coincidence of State Space and Action Space model

In the experimental model I, the action space and the state space are not coincided each other. It has two state input values such as the distance and the angle. And the possible numbers of control actions are 3-actions. In the experimental model II, the action space and the state space are totally coincide each other. It has one state input values which is the position of agent and the maximum number of control actions are N-actions. These two experimental models are based on continuous states and discrete actions of feeder mouse.

4.2.1　Experimental Model I

The experimental model is non-coincidence of state space and action space model. There is the action space which the agent is learning toward the goal-area shown in Figure 4.1, and the state space which is trying to segment the continuous space as a Voronoi space division shown in Figure 4.3. In the action space, the reward-area is denoted by square shape and the agent is denoted by triangular arrowhead which indicates the direction in which the agent is moving. The agent receives a reward of +1 when it enters the reward-area, then the agent's position is randomly changed on this action space. Otherwise, if the agent collides into a border or wall, it gets a reward of -1.

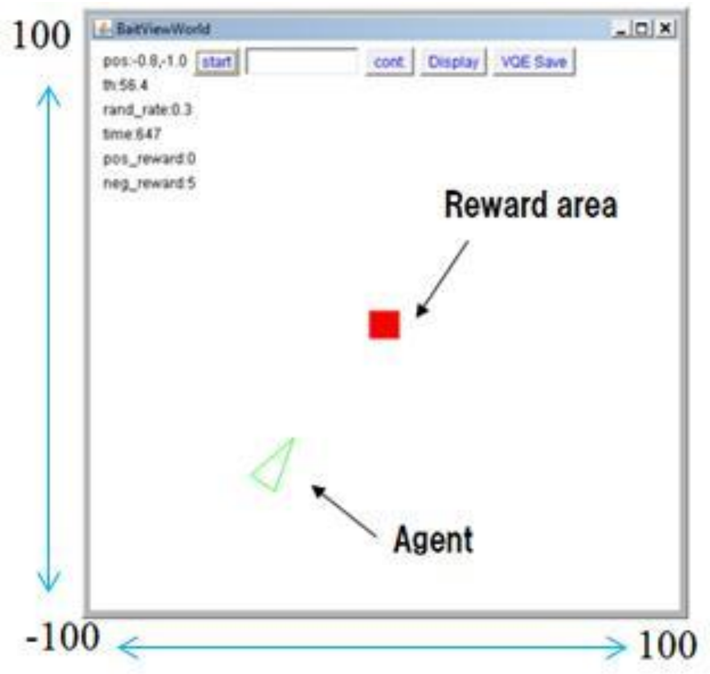

```
pos:-0.8,-1.0
th:56.4
rand_rate:0.3
time:647
pos_reward:0
neg_reward:5
```

Figure 4.1：　Action Space of Model I

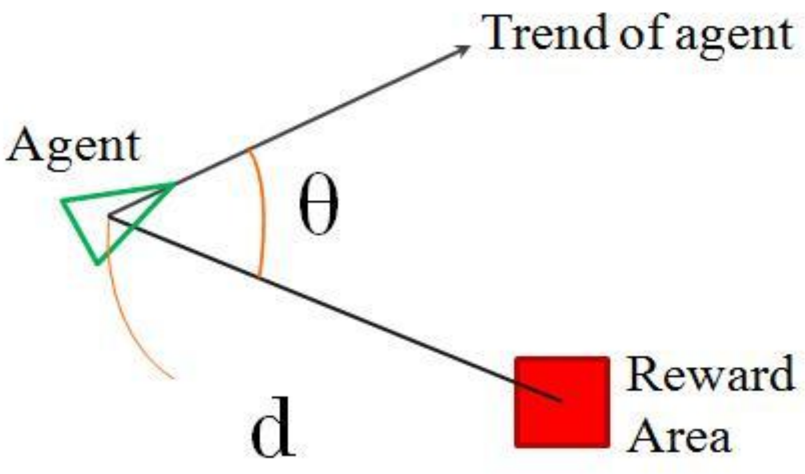


Figure 4.2： Two inputs values for Model I

The objective of agent is to get reward as much as possible during a specific period. The reward-area is put in the center of the action space as a constant in fixed position. The number of dimensions can increase by 2-dimensions when one reward-area is increased. The size of state and action space is -100 to 100.

4.2.1.1   2-Inputs values and 3-Types of agent's actions

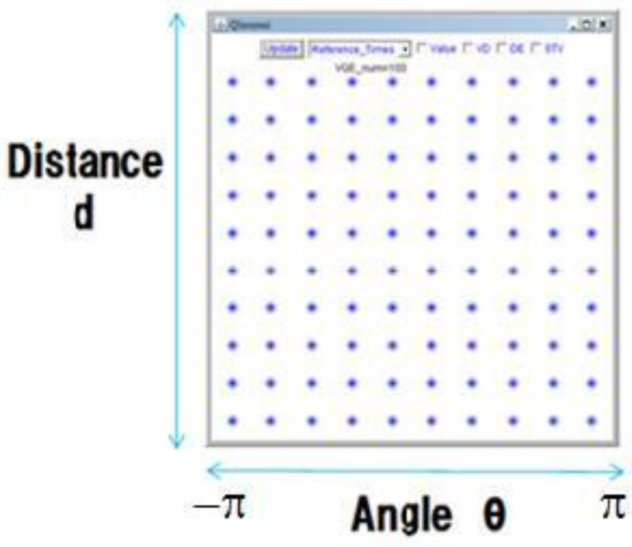


Figure 4.3： State Space of Model I

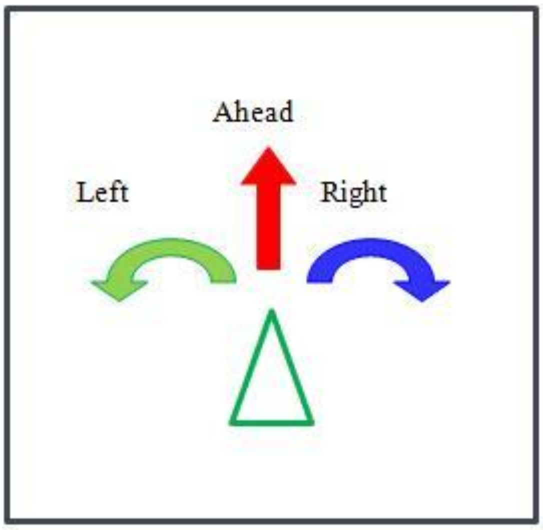
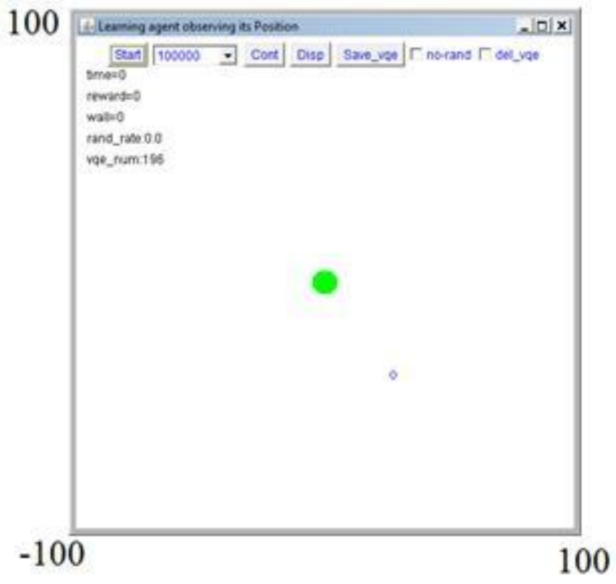


Figure 4.4 :   Three types of agent's action selections for Model I

The agent observes the distance (***d***) from the position of the agent to the reward-area, and the angle (θ) between the trend of agent and reward-area as shown in Figure 4.2. These 2-state input values generate the 2-dimensional state space as shown in Figure 4.3. In this state space, VQEs are arranged into 10×10 lattice structure therefore it has 100 numbers of VQEs on the state space. The 3-kinds of action selections are 1) straight ahead display by red color, 2) right rotation display by blue color, and 3) left rotation display by green color as shown in Figure 4.4.

4.2.2   Experimental Model II

The experimental model II is coincidence of state space and action space model. It means the action space and the state space totally coincide with each other. In this model II, the agent is supposed to head toward the reward-area subjected to various performance criteria in which the reward-area is denoted by closed circle and the agent is denoted by open circle in the working environment of action space as shown in Figure 4.5. The agent has N-kinds of action selections and the position of the agent is assumed as state input. The reward-area is placed at the center of the action space in



```
time=0
reward=0
wall=0
rand_rate:0.0
vqe_num:196
```

Figure 4.5： Action Space of Model II

fixed position as constant. The size of state and action space is -100 to 100. In the state space, VQEs are arranged into 10×10 lattice structure therefore it has 100 number of VQEs on the state space as shown in Figure 4.6. The initial position of agent and position of VQEs can change by the different seeds of random number which is sets of digits (i.e., 1,2,3,4,5,6,7,8,9).

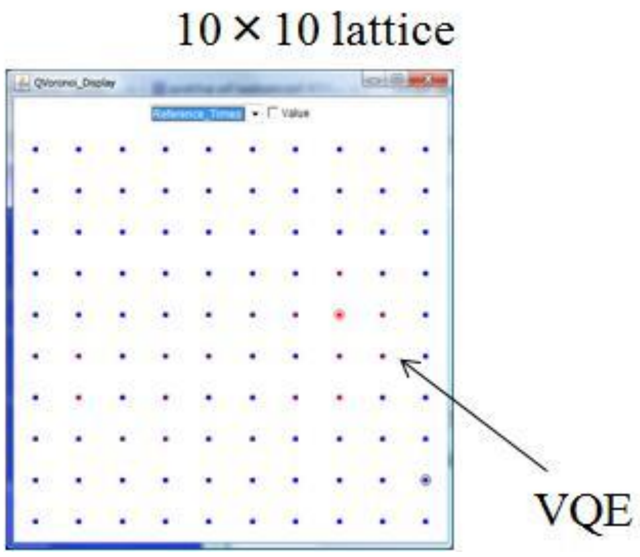


Figure 4.6： State Space of Model II

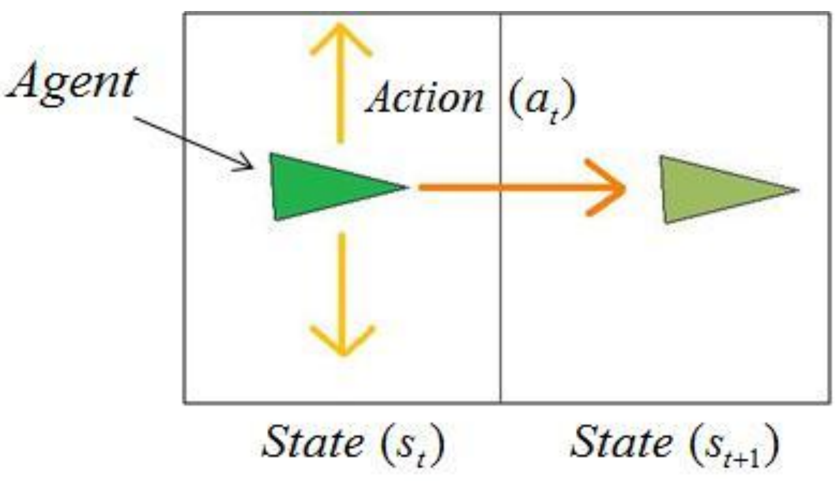


Figure 4.7 : Deciding the action

4.2.3 Decision making technique of agent's action

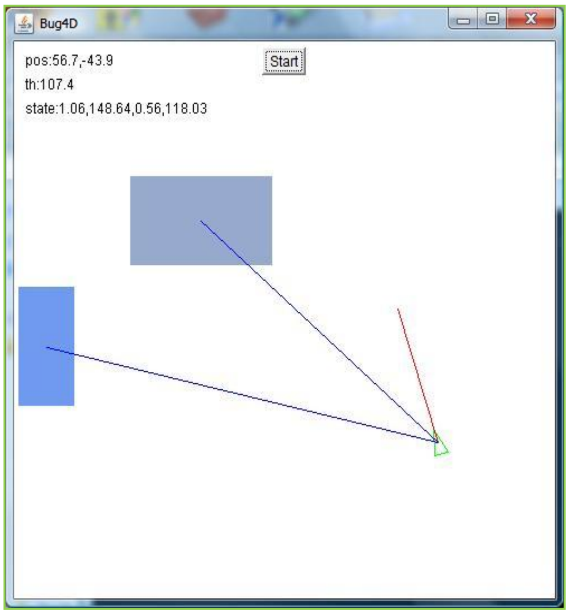
The agent selects the next action which has the highest Q-value. An action which has a large Q-value is considered to be a good way to achieve the goal. However, selecting the highest Q-value continually decreases the opportunity to find a better way. Therefore, the agent sometimes selects the next action at random. This random selection is useful for exploring the state space and finding a new and better way which has not been found yet. An example of deciding action is described in Figure 4.7. This action selection method is epsilon-greedy method. In our experiments,

Probability of random action rate: $\varepsilon < 1$ (30%)

Probability of maximum Q-value action: $1-\varepsilon$ (70%)

4.3 The experiments of Q-Table and VQE

In order to examine the performance of Q-Table and VQE, we used the experimental model I. In this simulation, a bug is supposed to head toward the bait areas subjected to various performance criteria. This bug is an agent in this model and a bug moves in a closed 4-dimensional world. We call this model "bait world" because



pos:56.7,-43.9
th:107.4
state:1.06,148.64,0.56,118.03

Figure 4.8 :   Experimental Model I in a closed 4-D world

bait-area for the bug is put on this world. The bug eats the bait when it enters the bait-area. The objective of the bug is to eat as many of the good baits as possible during a specific period.

Figure 4.1 shows an image of the “bait world”. The two reward-areas are randomly placed in this closed 4-dimensional world. Since the number of bait areas is 2, the agent observes 4 parameter input values. These 4 parameters construct the 4-dimensional state space. In this study, VQEs posses radius to use the multi-dimensional tree (MD-Tree) search method because Voronoi diagram cannot able to use over 2-dimensions.

4.3.1 Experimental Parameters

The parameter values were set as shown in Table 4.1. In our experiment, one “turn” means one cycle of reinforcement learning, i.e., from an observation of the agent to an update of the Q-value. Here, 100,000 turns make one “episode” and executed for 400 episodes in one “experiment”. And then, we count the number of rewards that the agent

Size of action space	100×100
Partition No. of Q-Table	10×10, 15×15 lattice size
Initial value of Q	$0 \le Q(s,a) \le 0.01$
Probability of random action	0.3
Probability of optimal action Q-value	0.7
Velocity of agent	
Straight ahead, right rotation, left rotation	5
10^6 continuous action times	1 episode
Number of episodes	400 episodes
Learning rate α	0.1
Discount rate γ	0.9
Experiment times by changing random seed number	10 trials
Radius of VQE	0.03, 0.04, 0.044, 0.045, 0.05, 0.053, 0.06, 0.07, 0.075, 0.08, 0.085, 0.09

Table 4.1：Experimental parameters of 4-D world

entered during one period, and did 10 trials for one “episode” by changing the different initial seed of random numbers. Then we took an average of those 10 trials. The learning rate α was set to 0.1 and the discount rate γ was set to 0.9.

4.3.2 Experimental Results

In order to prove the effectiveness of the proposed VQE method, the first simulation is carried out in 4-dimensions with radius of VQE. We did several experiments by changing the radius of VQE and size of lattice. However, we show some results in here.

The obtained behavior of reward numbers for the size of the lattice on 10×10 and 15×15 are shown in Figure 4.9, and the changes of states number also illustrated in Figure 4.10 respectively. The horizontal axis represents the number of episodes, and the vertical axis represents the number of rewards or the number of states. The number of rewards in 10×10 Q-Table is extremely good than 15×15 Q-Table. In figure 4.10, the size of the state space for 10×10 Q-Table is extremely large at 50,000 states. As a result, since the number of rewards that are subject to learning is relatively related to the size of lattice, the learning speed is extremely fast in small lattice size.

In the second simulation, the size of partition number of Q-Table and the radius of VQE are experimented as 15×15 and 0.045 by proper adjustment. The experimental results on the difference of reward numbers and state numbers between VQE and Q-Table are shown in Figure 4.11 and 4.12, respectively. As a result, the proposed technique gives a better performance than Q-Table at 2000 reward numbers as shown in Figure 4.11.

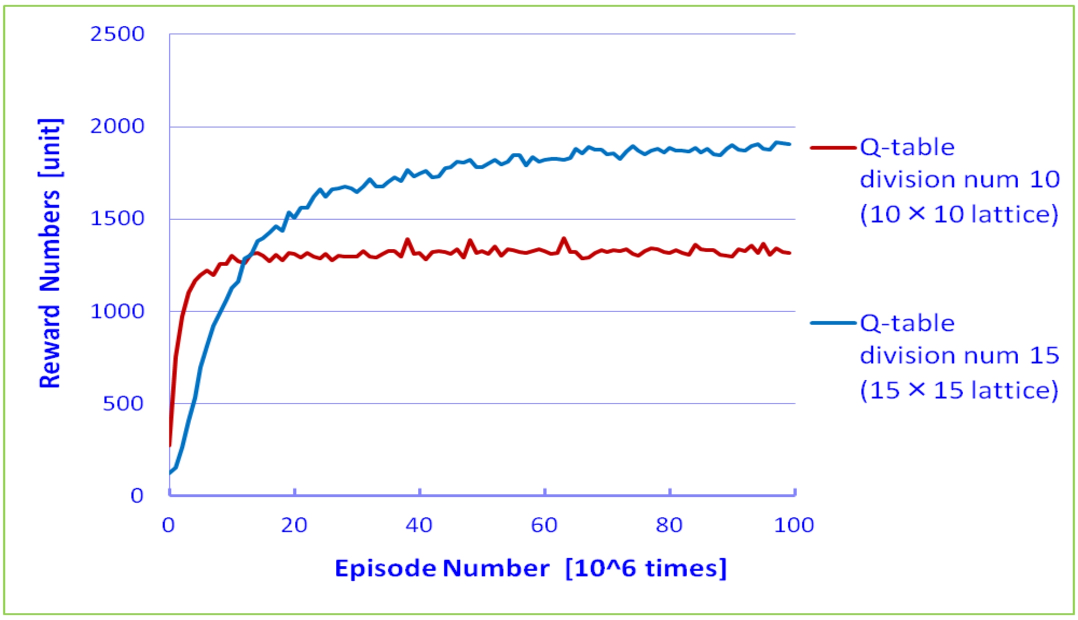


Figure 4.9 : Changes of rewards number in a 10×10 and 15×15 lattice size

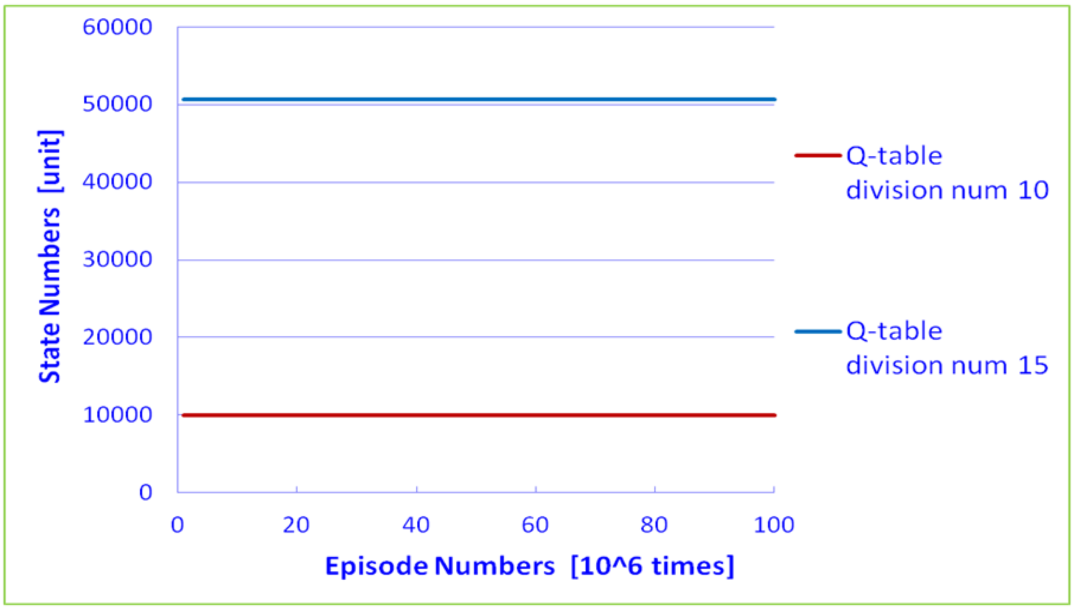


Figure 4.10 : Changes of states number in a 10×10 and 15×15 lattice size

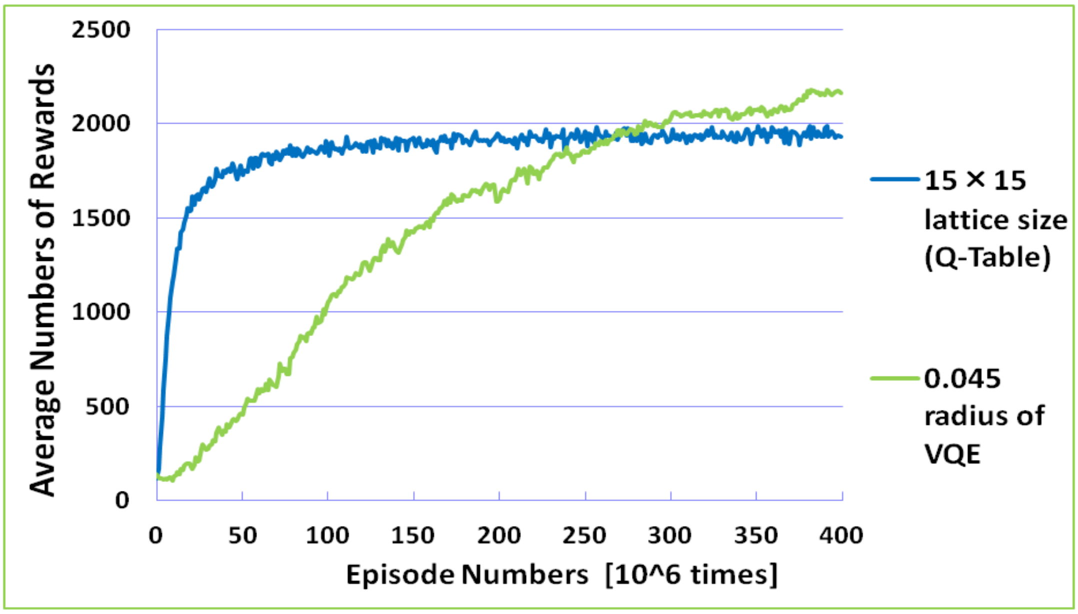


Figure 4.11 : Changes of rewards number in the case of 0.045 radius of VQE and 15×15 lattice size

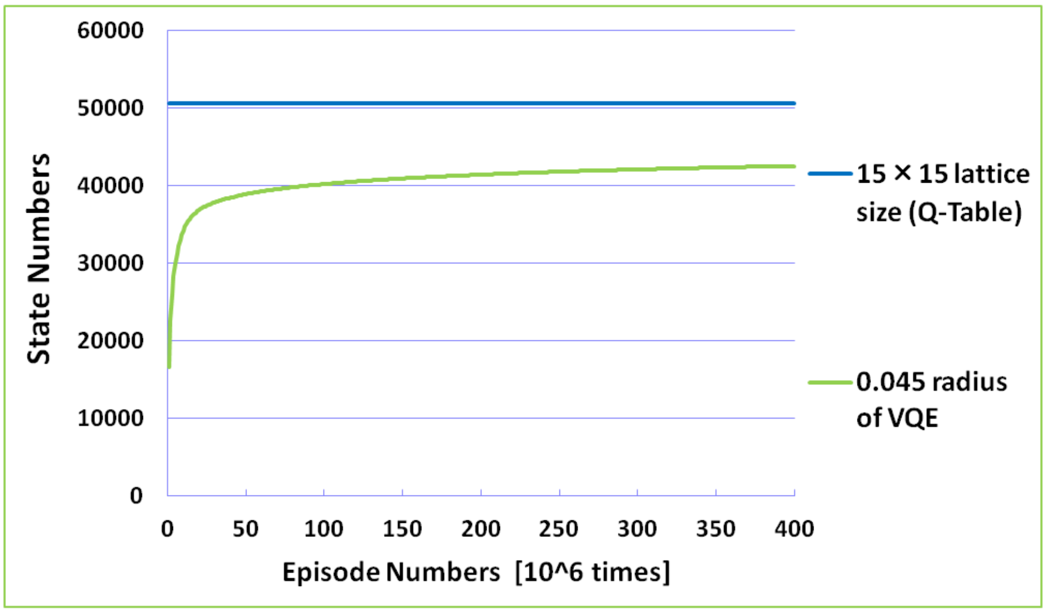


Figure 4.12 : Changes of states number in the case of 0.045 radius of VQE and 15×15 lattice size

4.4 Conclusion

From the results given above in Figure 4.9 and 4.10, the division number 10 in Q-Table of learning efficiency is better than 15×15 lattice. The division number 15 is five times greater than the division number 10. Therefore, the delay means that the reward is getting bigger. However, the ultimate reward acquisition is around 1500.

As the results of experiment shown in Fig. 4.11 and Fig. 4.12, compared to Q-Table, VQE can increase the number of rewards and it take less number of states than Q-Table. Therefore, VQE is effect extremely for Voronoi space division in continuous state space. And we can say that Q-Learning using VQEs are more effective than using normal Q-Learning.

Chapter 5

A comparison of learning performance in two-dimensional Q-learning by the difference of Q-values alignment

5.1 Introduction

In this chapter, we examine the learning efficiency of VQE in various strategies under different situations on 2-dimensional state space. In order to check the effectiveness of VQE, 2-types of simulations experiments for each model are carried out.
For experimental Model I;

Exp-1: Random arrangement of VQEs

Exp-2: Turning angles of VQEs by degrees in lattice arrangement

For experimental Model II;

Exp-1: Random arrangement of VQEs

Exp-2: Turning angles of VQEs and agent's action by degrees in lattice arrangement

In these experiments, we tested the learning performance with one agent and one reward area in a closed 2-dimensional space in case of the reward-area is placed at the center of the action space with a fixed position. Each experiment was done 10 times and the results were averaged. It means the agent learns the continuous actions with one hundred thousand times as one “episode” and executed for 20 episodes in one “experiment”. Then, the number of rewards that agent entered to reward-area is counted for each episode and did 10 trials for each episode and take an average of those 10 trials by changing the different initial seeds of random number. We examined the effectiveness of VQE without using radius in 2-dimensional state space using 2-types of experimental model. The agent learns to reach the reward area successfully during the reward-based learning process under various conditions.

5.2 Experiments of Model I

5.2.1 Random arrangement of VQEs

Firstly, we consider the VQE arrangement into lattice structure. It means VQEs are arranged in an orderly lattice. A general 2-dimensional lattice in a 10×10 size of Q-Table (10×10 grid environment) is shown in Figure 5.1. When this original 2-dimesional lattice is conducted by random noise of flat distributions (i.e., 0.01, 0.02, 0.03, 0.04, 0.05), we get the random arrangement of VQEs looks like Figure 5.2.

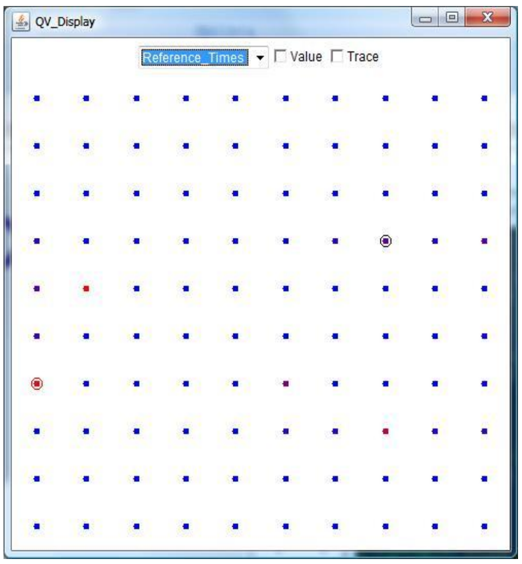


Figure 5.1 : Original 2-dimensional lattice

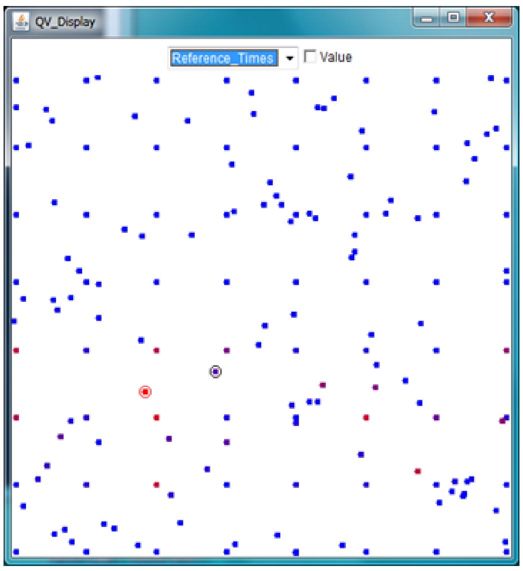


Figure 5.2 : Random arrangement of VQEs

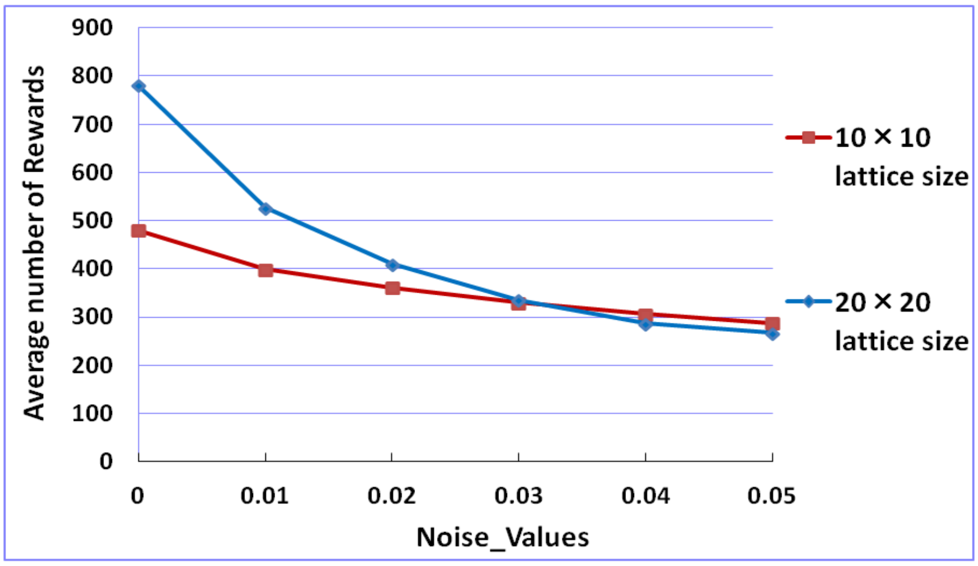


Figure 5.3 : Result of random arrangement of VQEs

Figure 5.3 shows the experimental results that were obtained by using original lattice arrangement and random arrangement of VQEs in a 10×10 lattice size and a 20×20 lattice size. The horizontal axis represents the values of noise of flat distribution, and the vertical axis represents the average number of rewards that the agent received. In Figure 5.3, 0 is lattice arrangement. As you can see, the number of rewards is going down from the original 2-dimensional lattice arrangement to random arrangement. Therefore, we can say that the learning performance is changed depend on the position of VQEs.

5.2.2 Turning angles of VQEs by degrees

In this subsection, we rotate VQEs that are arranged into lattice structure between the ranges of 0 degrees to 90 degrees by 5 degrees intervals of counter-clockwise rotation in a 20×20 lattice (Figure 5.4). If we rotate the 2-dimensional 20×20 lattice by 45 degrees, we get the following rotated lattice looks like Figure 5.5.

The experimental result of rotated VQEs that are arranged in a 20×20 lattice is shown in Figure 5.6. As a result of turning angles of VQEs experiment, the number of rewards most decreased at 45 degrees. It is considered that if VQEs are rotated, when a

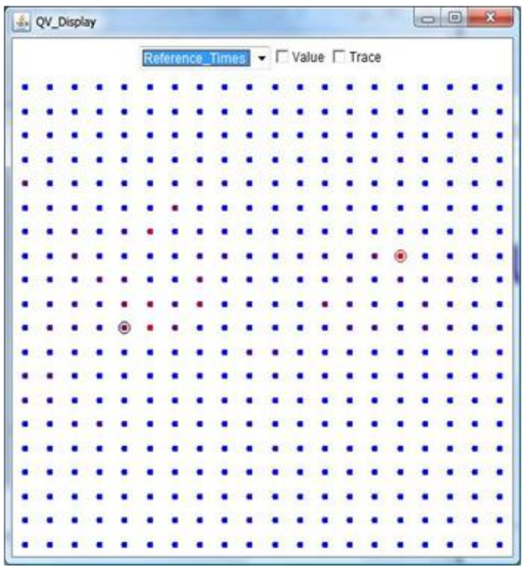


Figure 5.4 : 20×20 lattice

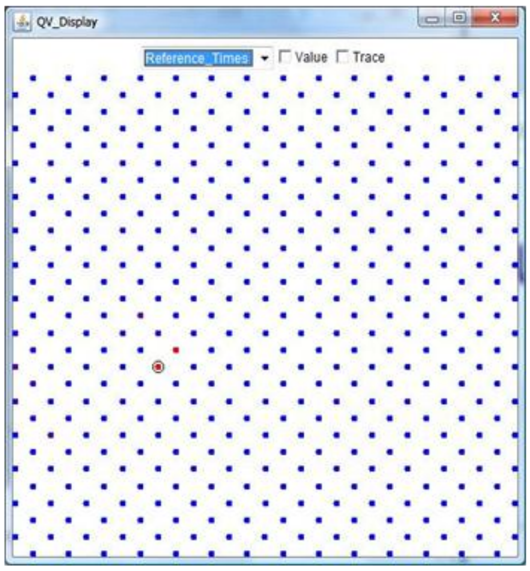


Figure 5.5 : A rotated VQEs in a 20×20 lattice by 45 degrees

state is changed a state goes to the same state although the agent takes a different action as shown in Figure 5.7. In the above Q-Learning equation, Q-value is decreased if the

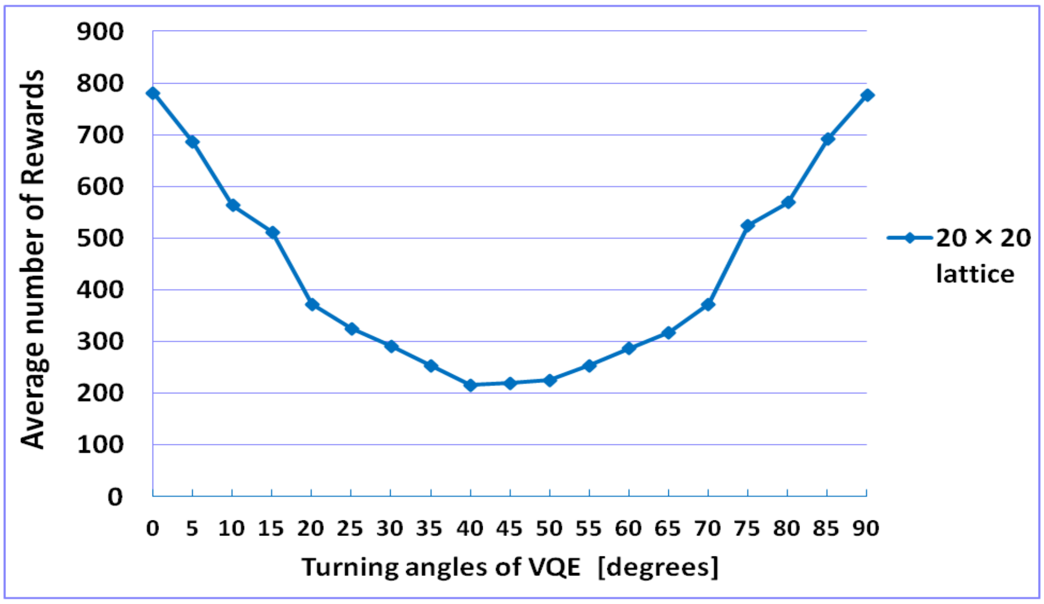


Figure 5.6 : Result of turning angles of VQE by degrees in a 20×20 lattice

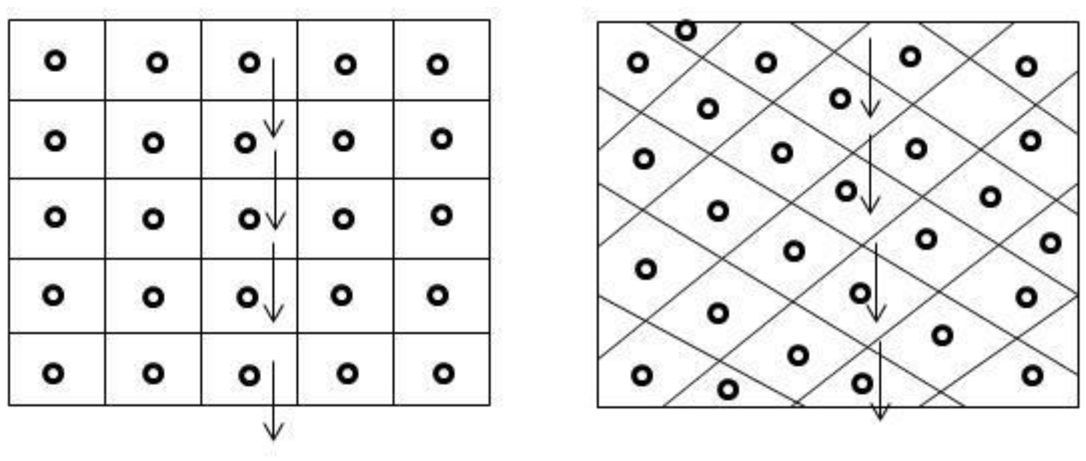


Figure 5.7 : The most common cause of decreased learning in a rotated VQEs

action acts in the same area. It may the most common cause of decreased learning (Figure 5.7). Actually, when a state is changed, the agent takes different action and a state also must be changed to different state.

5.3 Experiments of Model II

5.3.1 Random arrangement of VQEs

To clearly show the effectiveness of the proposed technique of VQE, we also used similar experiment but different experimental model estimates in lattice arrangement (Figure 5.8) and random arrangement of VQE (Figure 5.9) with noise of flat distribution (i.e., 1, 2, 3, 4, 5), which were used as comparison techniques in this experimental model II. In the environment described above, we compared the proposed technique of VQE with lattice arrangement and random arrangement in terms of the number of lattice size using the parameter values that described in above.

As Figure 5.10 reveals, when VQEs are randomly arranged on the state space, the learning speed was clearly slow in any experimental model. Therefore, it needs to decide the most suitable position of VQE because the learning speed decreases depend on the position of VQEs.

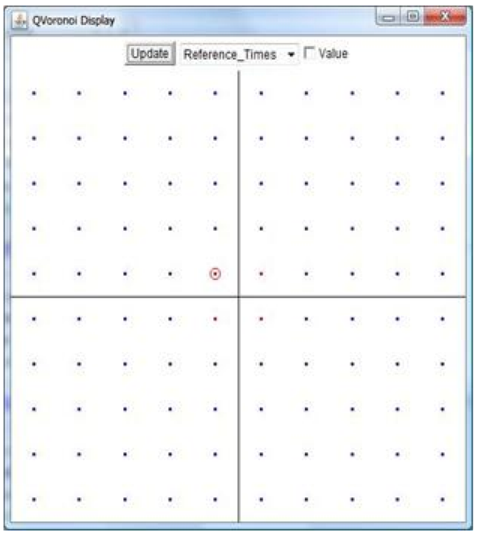


Figure 5.8 : 10×10 size of lattice

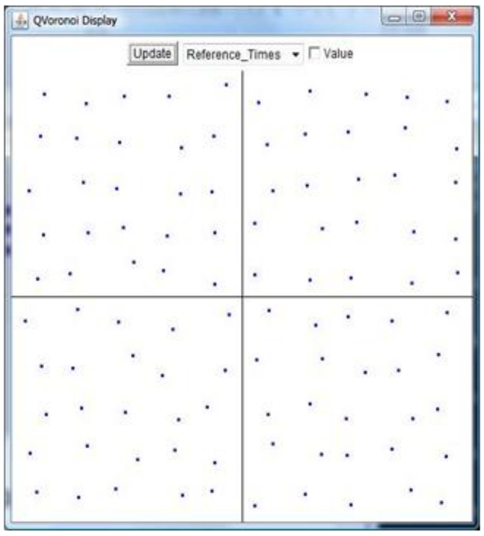


Figure 5.9 : Lattice to random arrangement of VQEs

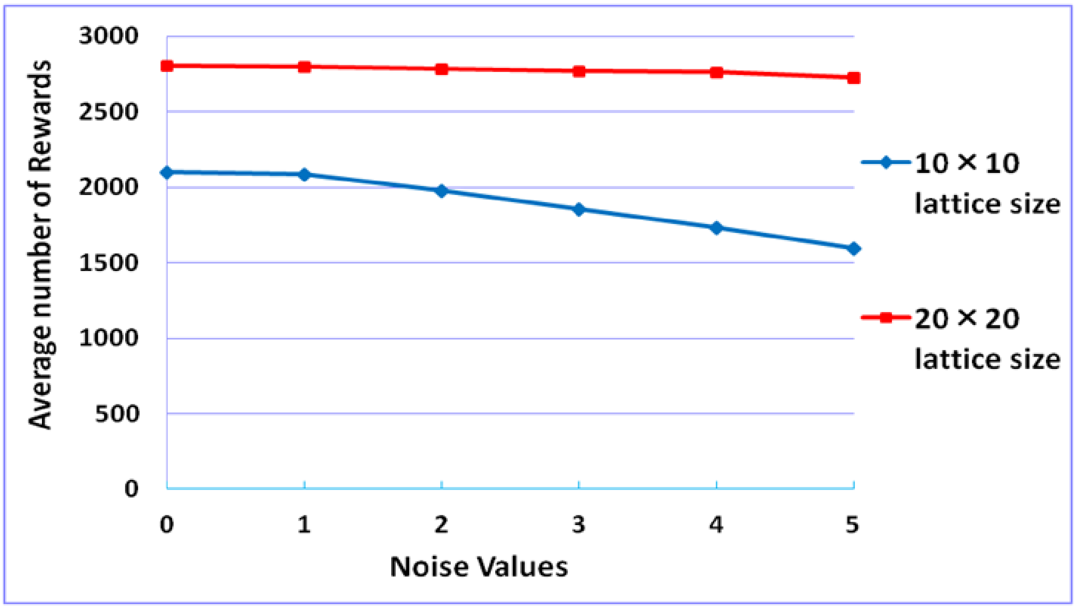


Figure 5.10 : Result of random arrangement of VQEs in Model II

5.3.2 Turning angles of VQEs and agent’s actions by degrees

In the case of Model II, to investigate the effect of VQEs we rotated VQEs and agent’s actions by degrees between 0 degrees to 90 degrees by 5 degrees intervals of counter-clockwise rotation in a 10×10 lattice and 20×20 lattice. In this simulation, the possible number of control actions is assumed as 4-actions such as 1) go up, 2) go down, 3) go left, and 4) go right. Figure 5.11 shows 4-kinds of actions at 0 degrees. If we rotate these normal actions by 45 degrees, we get a rotated action looks like this Figure 5.12. VQEs that are arranged into lattice structure also rotated.

There are 2-elements rotations such as angle of VQE and angle of action. The obtained results are shown in Figure 5.13 and 5.14, respectively. We performed the experiment using a lattice size of 10×10 and 20×20. According the result of Figure 5.13, in a 10×10 lattice size of the state space for rotation, the number of rewards increase when either the angles of VQEs or agent’s action is just 45 degrees out of alignment in case of 4-actions. Additionally, the learning speed increase when either turning angles of VQEs or agent’s action is just 60 degrees out of alignment in case of 3-actions and in

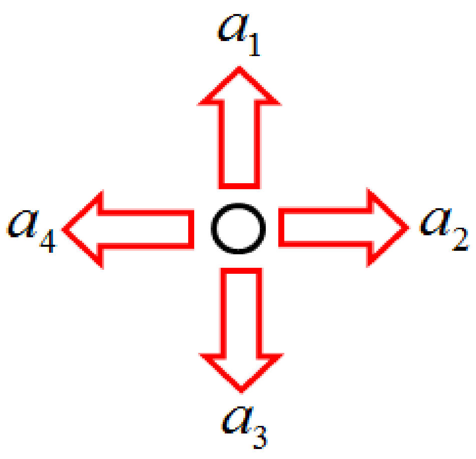


Figure 5.11 : 4-kinds of actions at 0 degree

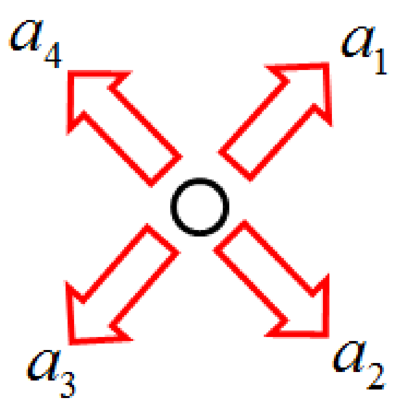


Figure 5.12 : A rotated actions by 45 degrees

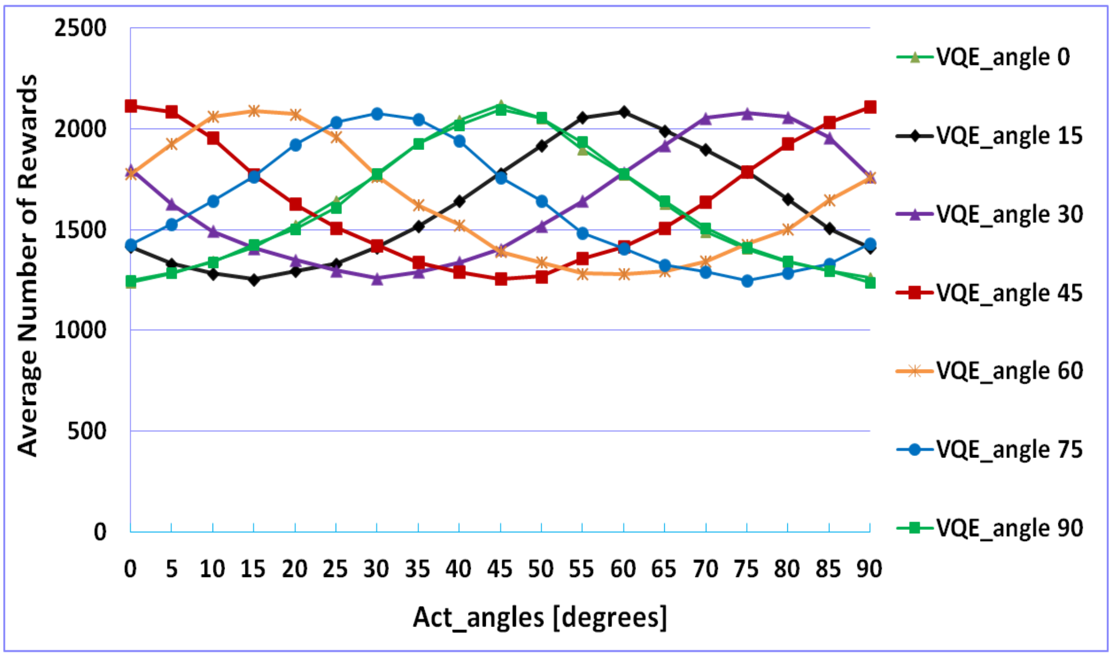


Figure 5.13 : Result of turning angles of VQEs and actions in a 10×10 lattice

case of 6-actions when just 60 degrees out of alignment, respectively. Otherwise, it decreases if both have the same turning angles.

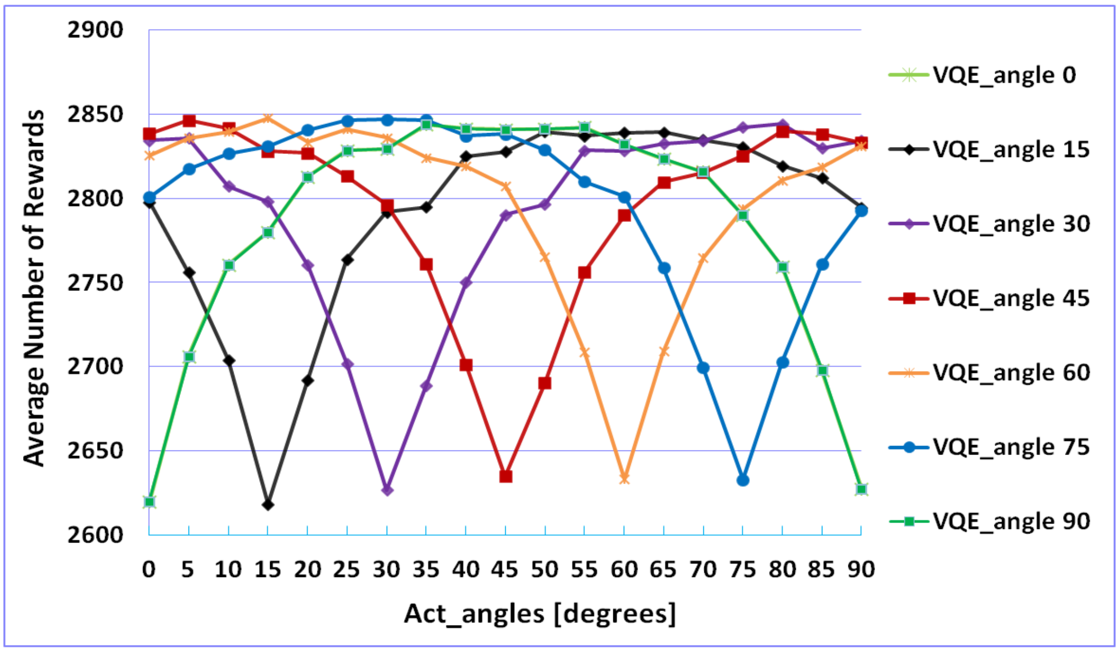


Figure 5.14 : Result of turning angles of VQEs and actions in a 20×20 lattice

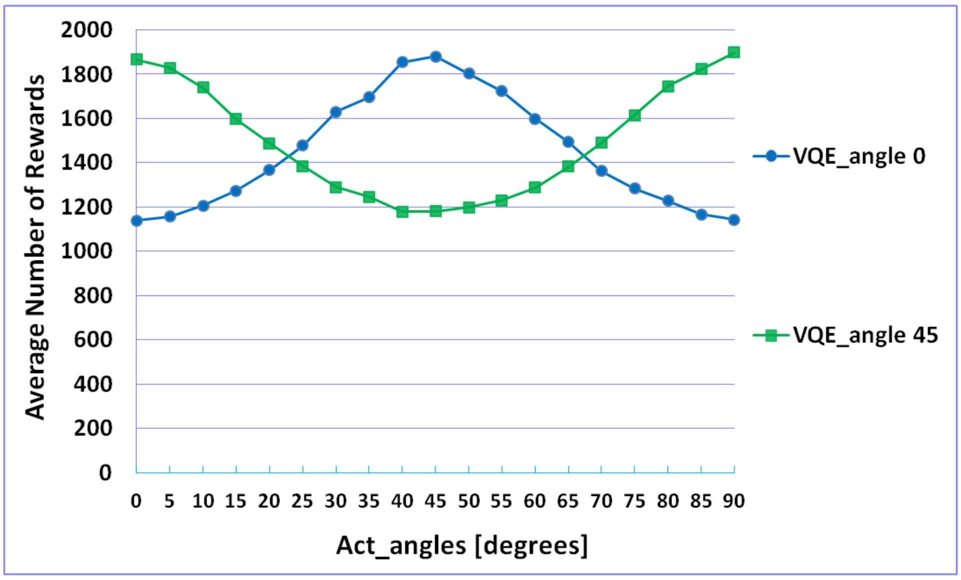


Figure 5.15 : Result of turning angles of VQEs and actions

Figure 5.15 clearly shows in order to realize Figure 5.13, and Figure 5.14. It means the number of reward increase either turning angles of VQEs or action is just 45 degrees out of alignment in case of 4-actions. On the other hand, if the turning angles of VQEs and actions have both same angles, the number of reward is decreased. Figure 5.14 shows that a similar result also occurs in a 20×20 lattice.

Moreover, when the turning angles of action at 45 degrees in 4-actions, the number of rewards has increased at 0 degree, 90 degrees, 180 degrees, 270 degrees, and 360 degrees turning angles of VQEs. Conversely, the number of rewards has decreased at 45 degrees and 135 degrees turning angles of VQEs.

5.4 Conclusion

In this chapter, we examined the learning performance of VQEs on various strategies under different conditions using 2-types of experimental model with 2-kinds of simulation environments in case of the reward-area is placed at the center of the action space with a fixed position. At the current stage, we only performed the experiments in 2-dimensions.

According to the results of experiments, in any experimental model the learning speed decreased in the case of random arrangement of VQEs. We believe that the decrease in the learning speed of random arrangement of VQEs was depended on the position of VQEs. In the experiment of experimental model II, if we rotate VQEs and actions, the learning speed increase when either turning angles of VQEs or actions is just 45 degrees out of alignment in case of 4-actions. Additionally, the learning speed increase if 30 degrees out of alignment in case of 6-actions, and 60 degrees out of alignment in case of 3-actions. On the other hand, the learning speed has decreased if both have the same turning angles.

Furthermore, it is also concluded that when the turning angles of actions at 45 degrees in 4-actions, the number of rewards has increased at 0 degree, 90 degrees, 180 degrees, 270 degrees, and 360 degrees turning angles of VQEs. Conversely, the number of rewards has decreased at 45 degrees and 135 degrees turning angles of VQEs. From this experiment, the optimal position of VQE is obtained from the group of state transition vector (STV) of each action. As future topics of research, we plan to propose the addition method of VQE to decide the position of VQEs because we want to make the optimal state space. By deciding the position of VQEs with the actions, we will implement the method of VQE in division and integration. Moreover, we plan these techniques to a high-dimensional problem by generating an N-dimensional state space.

Chapter 6

A proposition of addition method of VQEs in Q-Learning

6.1 Introduction

In this chapter, we present the addition method of VQEs to decide the position of VQE. Additionally we introduce the block-counting method, and LBG method for vector grouping based on adaptive state space partitioning algorithm. This study will also seek to develop the efficiency of reward learning based on state-space partitioning technique. Accordingly, we conducted our purposed method on continuous state space and discrete actions, and compare the performance with size of square partition lattice. Since we consider a single-agent RL problem on continuous state space and discrete actions, we apply 2-types of experimental models I and model II in different environments as mentioned in previous section 4.2. These 2-types of models have same purposes but different state input variables and different number of control actions.

This chapter is organized as follows. First, the addition method of VQEs using model II is presented in Section 6.3. In section 6.3.2, the block counting method is explained, and the results for model II are shown in Section 6.3.3. In section 6.4, the same addition method of VQEs using model I is presented. And then, the LBG adaptive vector quantization algorithm is described in Section 6.4.2 to organize the number of

STVs into several groups. The efficiency of rewards learning is investigated and the simulation results using model I are explained in Section 6.4.3. A conclusion is also given in Section 6.5.

6.2 Addition Algorithm

From the previous experiment of turning angles of VQE and actions using experimental model II, the optimal position of VQE is obtained from the group of state transition vector (STV) of each action. The addition algorithm is as follows:

1) Collect STVs during a specific period of time.
2) Categorize the STVs into several groups.
3) Calculate the centroid of each group.
4) Add a VQE to that midpoint or centroid of STV's group.

To employ the addition algorithm, 2-types of experimental model are used.

6.3 Addition method of VQEs using Model II

6.3.1 First-stage of addition of VQEs

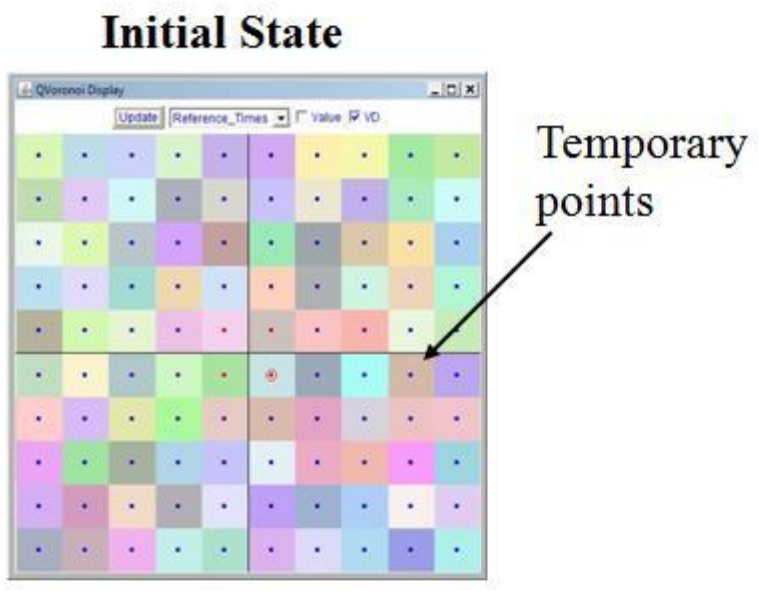


Figure 6.1 : Initial-state of addition of VQEs on state space

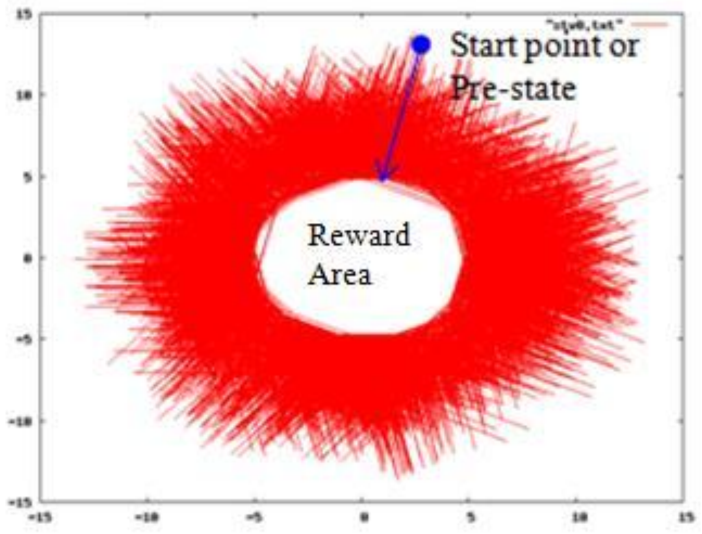


Figure 6.2 : Collecting STVs during a specific period of time

In the first-stage of VQE's formation using model II in the case of reward-area is placed at the center of the action space with fixed position, we first arrange the temporary points into lattice structure on the state space as shown in Figure 6.1. When the learning process is started, we take the state transition vectors (STVs) which is a distance that entered from the position of the agent to the reward-area (Figure 6.2).

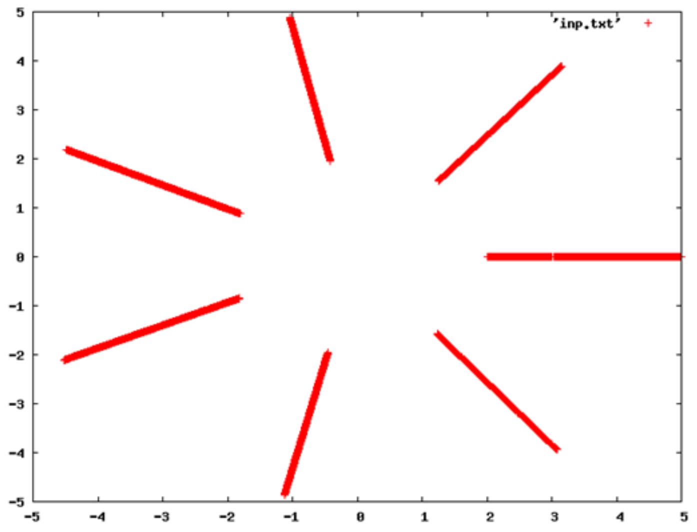


Figure 6.3 : Sketching the representative vectors

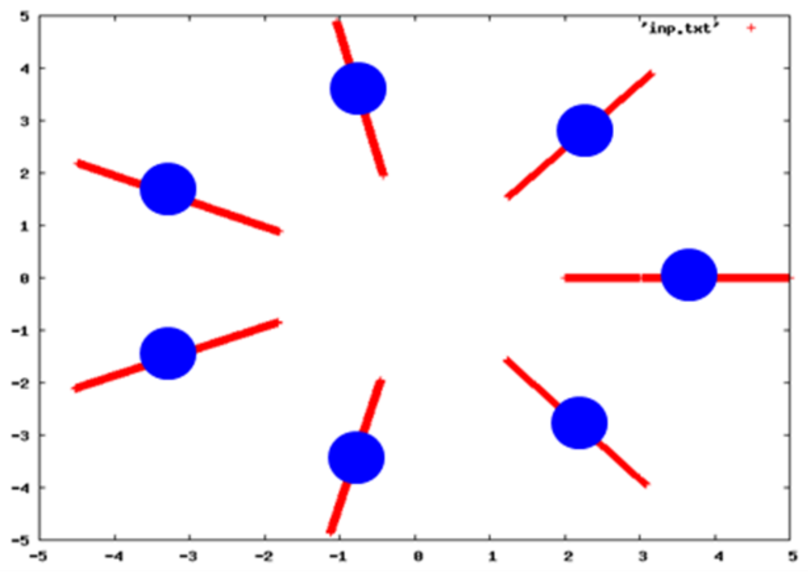


Figure 6.4 : Generating the VQEs at the centroid of representative vectors

Alternatively, it is discounted sum of the rewards that the agent received. If the amount of STVs gets 1000 vectors or more, we group or quantize the STVs by continuously taking the same actions using Block-counting algorithm (section 6.3.3). Additionally,

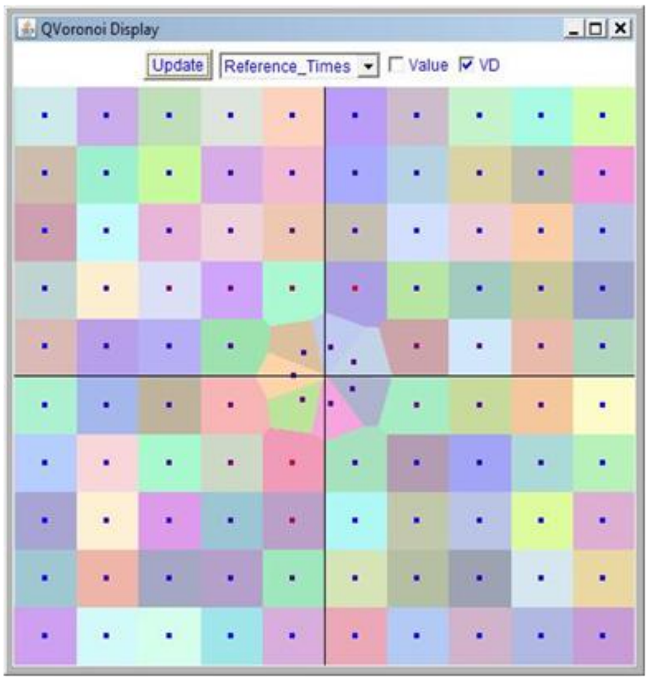


Figure 6.5 : Image of first-stage formation of VQEs in 7-actions

we seek the representative vectors and describe those STV groups as representative vectors (Figure 6.3). Thereupon, VQEs are generated at the place of each representative vectors in accordance with the number of control actions in regard to the reward-area on the state space (Figure 6.4). In essence, the continuous state space is partitioned into the number of N-actions subspaces as Voronoi regions. In this case, the possible numbers of control actions are assumed as 7-actions. Finally, we delete the temporary points in which VQEs are added on the state space. Figure 6.5 shows the image of first-stage output of addition of VQEs in case of 7-actions by drawing Voronoi diagram.

The implementation of partitioning algorithm is conducted on 2-dimensional state space with one state input variable and N-actions to examine the impact of learning. The state space is partitioned into the number of N-actions subspaces.

6.3.2 Next-stage of addition of VQEs

In the second-stage of addition of VQEs, we collect lots of STVs again that come from temporary points to new VQEs of the first-stage output as shown in Figure 6.6. We do not take the STVs that come from new VQEs to new VQEs. In this stage, we do the same process as mentioned in section 6.3.1 of first-stage formation. However in this second-stage, VQEs are added in regard to the created VQEs but it does not exactly produce in accordance with the number of control actions as the first-stage in generating. Moreover, a threshold value is calculated to make the group of STVs. A threshold value is the quotient of dividing the number accumulated STV (1000 vectors) by the number of actions for example 7-actions in this case (Figure 6.7). A threshold value can change depending on the number of actions. If the sum of STV values in taking the same action has greater a threshold value, we make the group which exceeds a threshold value, and represents as representative vectors.

Moreover, we calculate the minimum distance between two new added VQEs to merge the points into one if there were too close to each other. If the distance among the new VQEs of the second-stage is less than the ratio of a minimum distance, the first

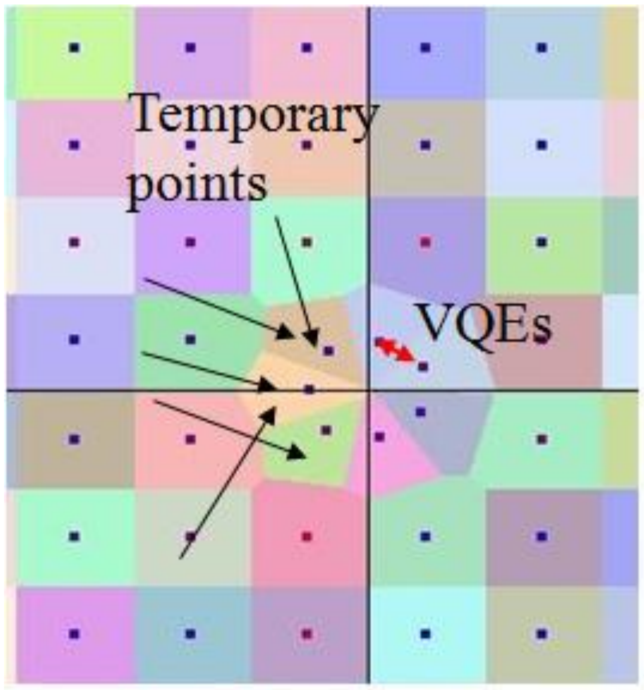


Figure 6.6 : Collection of STVs for next-stage of addition of VQEs

entry VQE is added. If not, there were no new VQEs around are added. The image of addition of VQEs in case of 7-actions is illustrated in Figure 6.8.

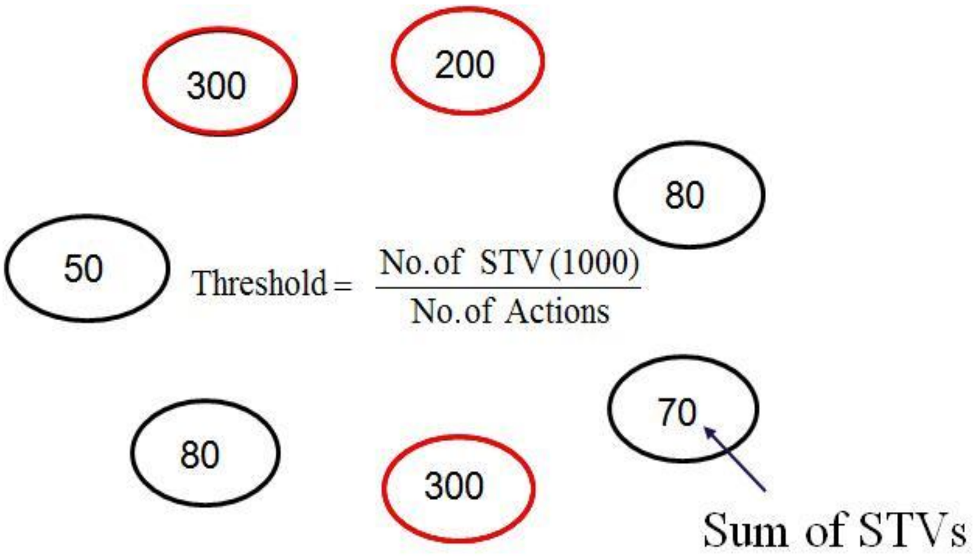


Figure 6.7 : Calculation of threshold values in second-stage for vector grouping

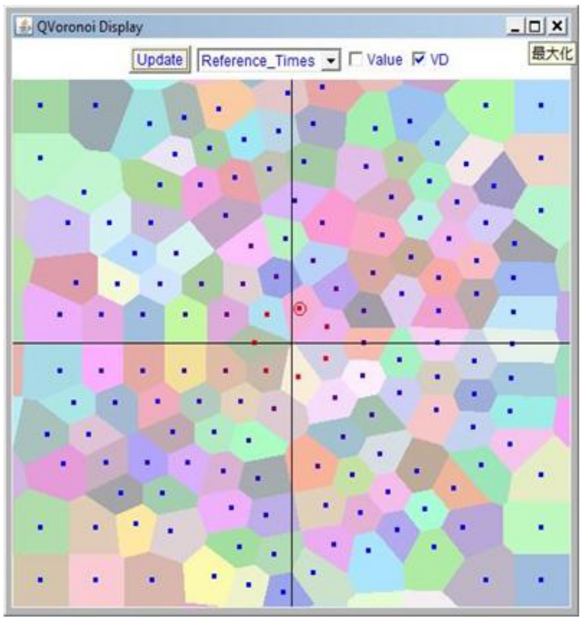
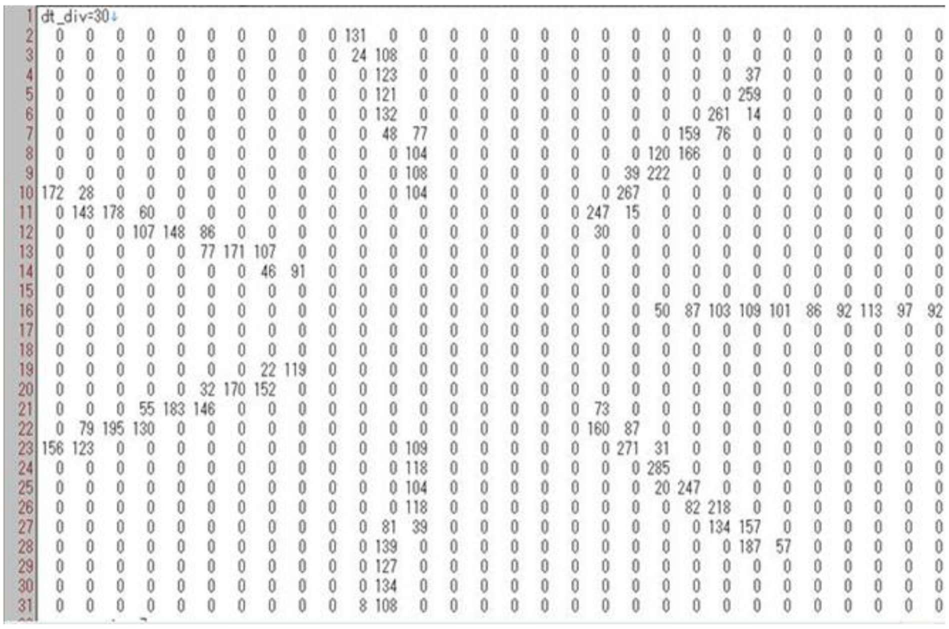


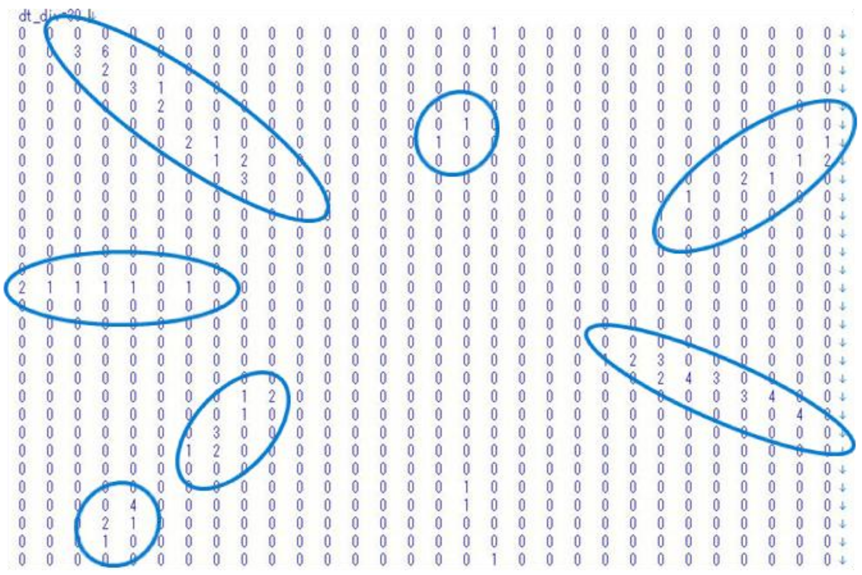
Figure 6.8 : Image of addition of VQEs in 7-actions

6.3.3 Block-counting method



```
dt_div=30
   0   0   0   0   0   0   0   0   0   0 131   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0  24 108   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0 123   0   0   0   0   0   0   0   0   0   0   0  37   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0 121   0   0   0   0   0   0   0   0   0   0   0 259   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0 132   0   0   0   0   0   0   0   0   0   0 261  14   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0  48  77   0   0   0   0   0   0   0   0 159  76   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0 104   0   0   0   0   0   0   0 120 166   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0 108   0   0   0   0   0   0  39 222   0   0   0   0   0   0   0   0   0
 172  28   0   0   0   0   0   0   0   0   0   0 104   0   0   0   0   0   0 267   0   0   0   0   0   0   0   0   0   0
   0 143 178  60   0   0   0   0   0   0   0   0   0   0   0   0   0   0 247  15   0   0   0   0   0   0   0   0   0   0
   0   0   0 107 148  86   0   0   0   0   0   0   0   0   0   0   0   0  30   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0  77 171 107   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0  46  91   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  50  87 103 109 101  86  92 113  97  92
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0  22 119   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0  32 170 152   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0  55 183 146   0   0   0   0   0   0   0   0   0   0   0   0  73   0   0   0   0   0   0   0   0   0   0   0
   0  79 195 130   0   0   0   0   0   0   0   0   0   0   0   0   0   0 160  87   0   0   0   0   0   0   0   0   0   0
 156 123   0   0   0   0   0   0   0   0   0   0 109   0   0   0   0   0   0 271  31   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0 118   0   0   0   0   0   0   0 285   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0 104   0   0   0   0   0   0   0  20 247   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0 118   0   0   0   0   0   0   0   0  82 218   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0  81  39   0   0   0   0   0   0   0   0   0 134 157   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0 139   0   0   0   0   0   0   0   0   0   0   0 187  57   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0 127   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0 134   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   8 108   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
```

Figure 6.9 : In case of 1000 threshold value of STVs



```
dt_div=30 ↓
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 ↓
0 0 3 6 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ↓
0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ↓
0 0 0 0 3 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ↓
0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ↓
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 ↓
0 0 0 0 0 0 2 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 ↓
0 0 0 0 0 0 0 1 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 2 ↓
0 0 0 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 1 0 0 ↓
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 ↓
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ↓
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ↓
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ↓
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ↓
2 1 1 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ↓
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ↓
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ↓
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ↓
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 2 3 0 0 0 0 0 0 ↓
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 4 3 0 0 0 0 ↓
0 0 0 0 0 0 0 0 1 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 4 0 0 ↓
0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4 0 ↓
0 0 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ↓
0 0 0 0 0 0 1 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ↓
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ↓
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 ↓
0 0 0 0 4 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 ↓
0 0 0 2 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ↓
0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ↓
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 ↓
```

Figure 6.10 : In case of 100 threshold value of STVs

The idea of proposed addition algorithm is to group together states with similar action for suitable partitioning. Consequently, the block-counting method of STVs is used to categorize STVs into several groups.

Here, the threshold value of STV was set to 1000 vectors and the division number of grid (lattice size) was set to 30×30. The threshold value is determined according to preliminary experiments, and this value was fixed throughout all episodes. However, if the number of STV is smaller than the current threshold value (1000 vectors) or if the block-space is divided more finely i.e. greater than the 30 division number of block-space, it is difficult to make the grouping. The method that was used to implement the vector quantization determines the size of the block-space and threshold value of STV for grouping.

6.3.4 The Experimental results of Model II

We compared the learning performance of addition method of VQEs using block-counting method for STV's grouping and Q-Table on 14×14 size of lattice (i.e.,

No. of all STV	No. of Actions	Ratio of a minimum distance
1000	3	0.5
	4	0.5
	6	1.0
	7	0.5 ~ 1.2

Width and height of space	－100～100
Size of reward-area	5
Learning rate (α)	0.1
Discount rate (γ)	0.9
Rate of random action	0.3
Initial Q-value	0≦Q(s,a)≦0.01
Magnification rate of new points	3.0
Execution times	1～10 (10 trials)
One trial	50 episodes
One episode	2× 10^5 times

Table 6.2 : Experimental parameters for experiment of addition method

196 states) in the case of the number of created VQEs are same with size of lattice in

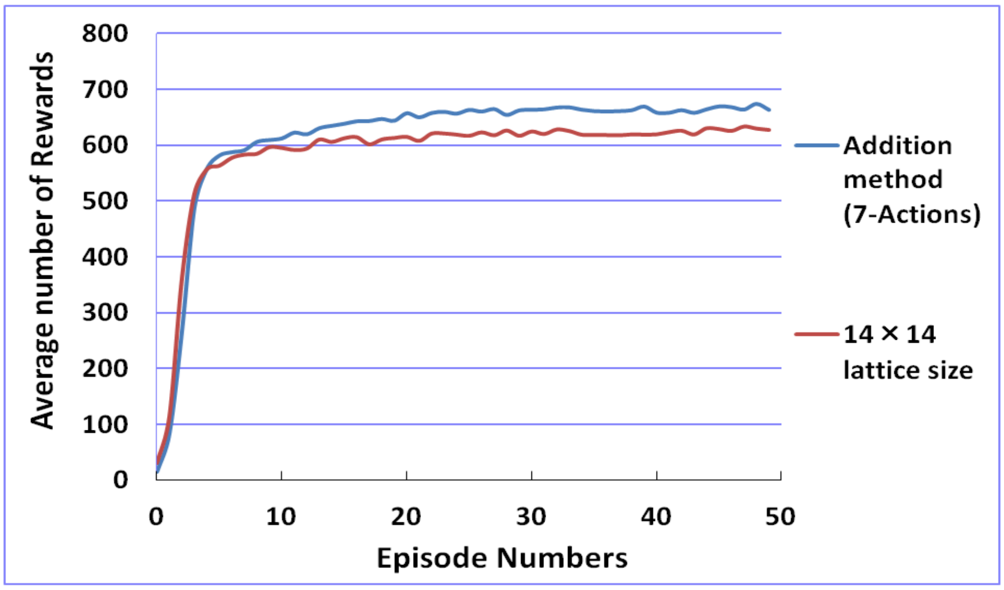


Figure 6.11 : Result of 7-actions

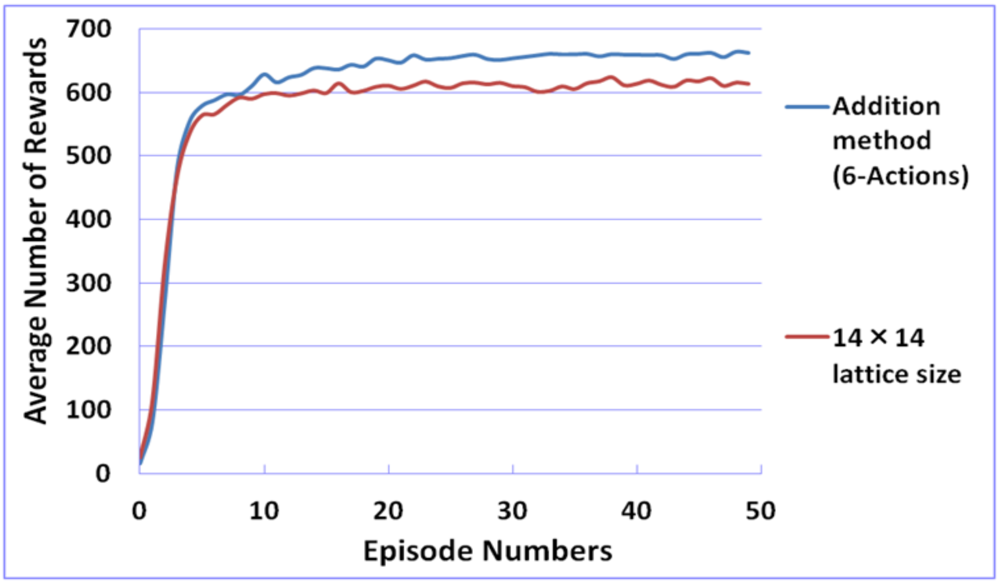


Figure 6.12 : Result of 6-actions

any N-actions. A continuous action learning time is 20 thousand time as one episode for 50 episodes, and performed 10 trials on each episode by changing the different initial seeds of random number. Furthermore, we did 4-experiments using 7-actions, 6-actions, 4-actions and 3-actions, and the results are shown in Figure 6.11, 6.12, 6.13, and 6.14

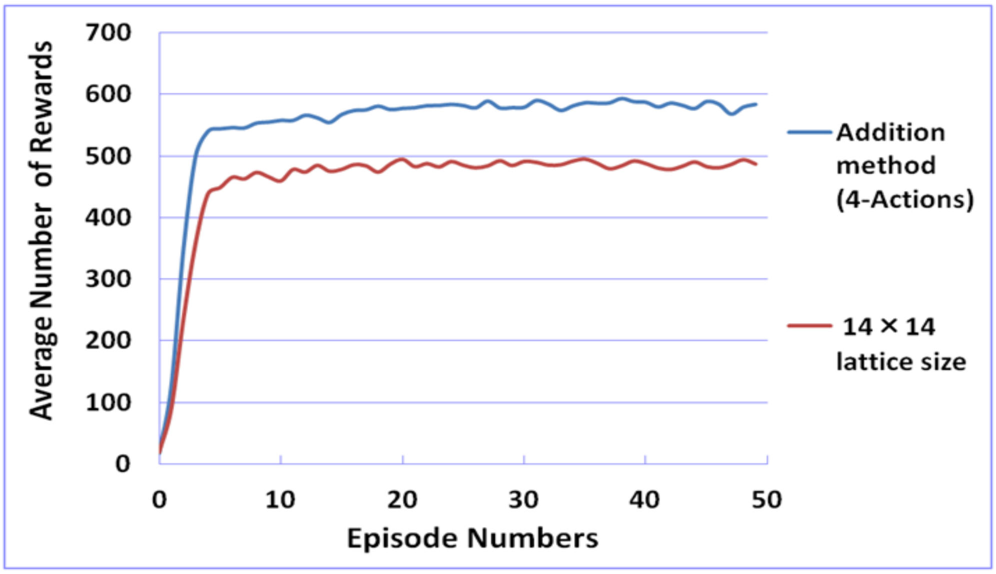


Figure 6.13 : Result of 4-actions

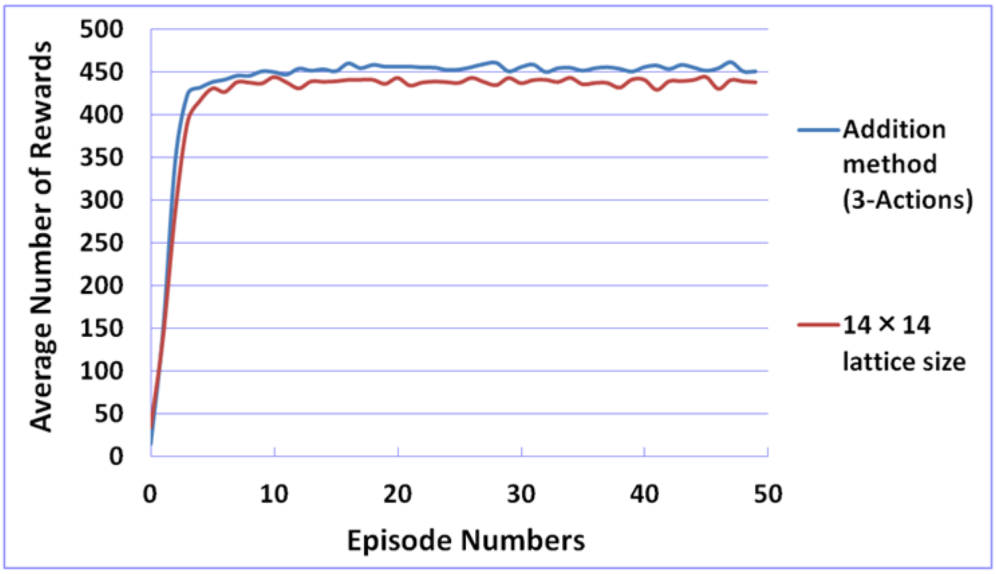


Figure 6.14 : Result of 3-actions

respectively. The experimental parameter values in both learning methods were set as shown in Table 6.2. The experiments for each task of control actions are carried out using these parameter values.

The result that were obtained by using the proposed technique which is addition method of VQEs, and lattice which is previous work conducted by coincidence of state space and action space model. The horizontal axis represents the number of episodes, and the vertical axis represents the number of average rewards that the agent entered to the reward-area. As a result, the learning speed of proposed addition method is slightly improved compared with Q-Table and these results show that the result can be changed depend on the behavior of the agent.

6.4 Addition method of VQEs using Model I

6.4.1 Creation of VQEs

In this section, the same addition algorithm is applied in different environment which is experimental model II. We add VQEs on continuous state space by implementing the state space partitioning formation with the same addition concept as

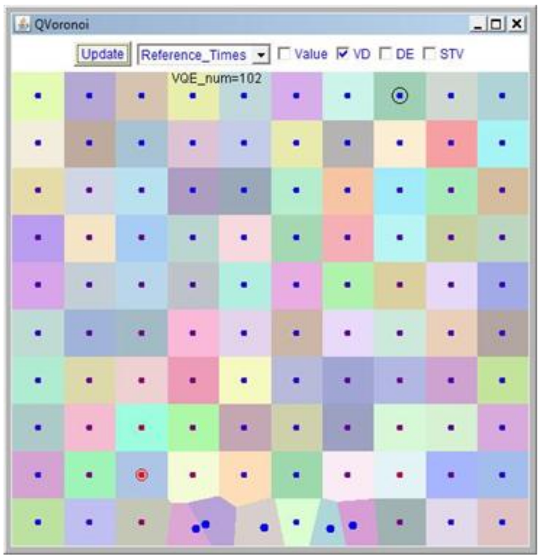


Figure 6.15 : An image of first-stage output of addition of VQEs

described in section 6.3. In the first-stage of addition of VQEs, since the experimental model is non-coincidence of state space and action space model, VQEs are produced on suitable multiple position of the direction of representative vectors as shown in Figure 6.15 but we do not judge yet whether that position is really appropriate location or not.

In next-stage of addition, lots of STVs is taken that entered from temporary points

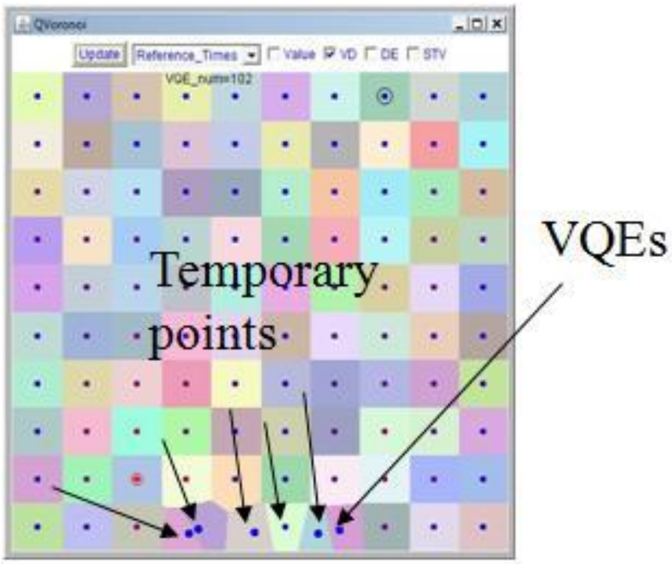


Figure 6.16 : Collection of STVs in next-stage of addition

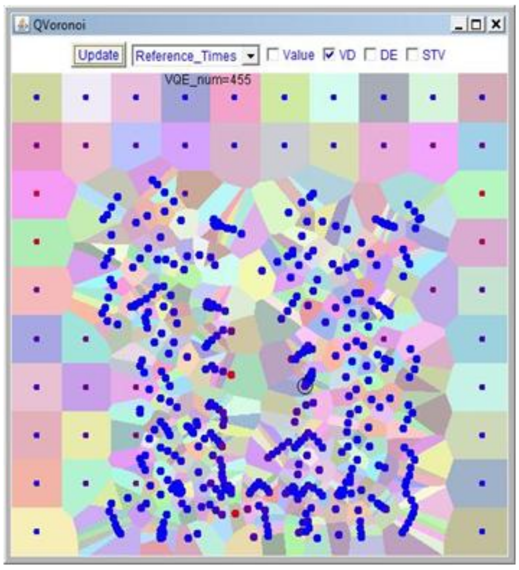


Figure 6.17 : An image of addition of VQEs in model I

to created VQEs of first-stage output, it does not take that come from VQEs to VQEs. An example of collecting STVs is illustrated in Figure 6.16. The image of addition of VQEs is described in Figure 6.17. In this addition, LBG algorithm is used for grouping of STVs.

6.4.2 LBG Algorithm

LBG (Linde-Buzo-Gray) algorithm is a typical technique of vector quantization algorithm and it looks like a clustering algorithm which takes a set of input vectors as input and generates a representative subset of vectors with a quantization vector. The vector quantization is a classical quantization technique from signal processing and image compression which allows the modeling of probability density functions by the distribution of prototype vectors. It was originally used for data compression. It works by dividing a large set of points (vectors) into groups having approximately the same number of points closest to them. Each group is represented by its centroid point. Since data points are represented by the index of their closest centroid, commonly occurring data have low error, and rare data high error. This is why vector quantzation is suitable

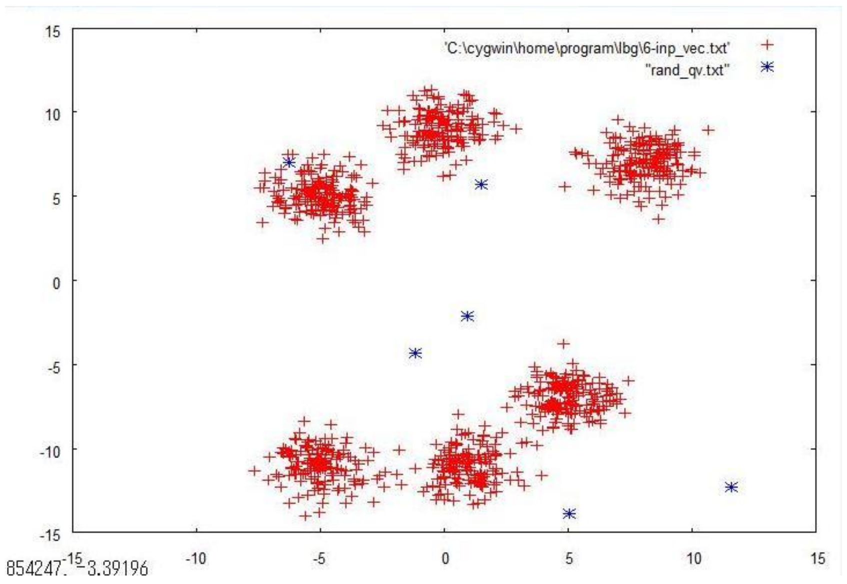


Figure 6.18 : Initial state of LBG algorithm

for lossy data compression. It can also be used for lossy data correction and density estimation. It assists to project a continuous input space on a discrete output space, while minimizing the loss of information.

The modification of adaptive vector quantization method was introduced enhanced

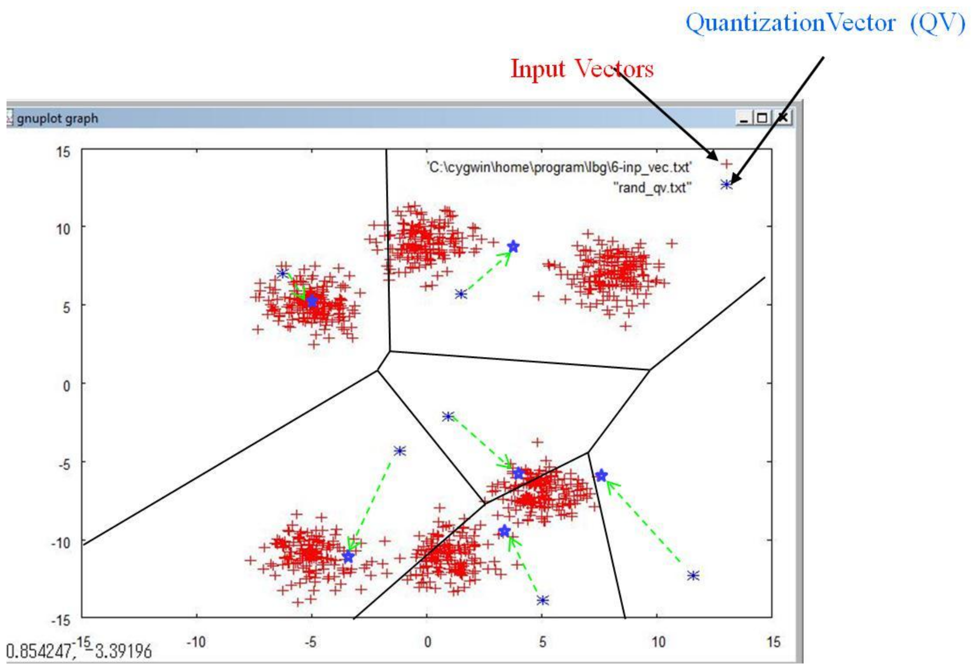


Figure 6.19 : Deciding the cluster of input vectors

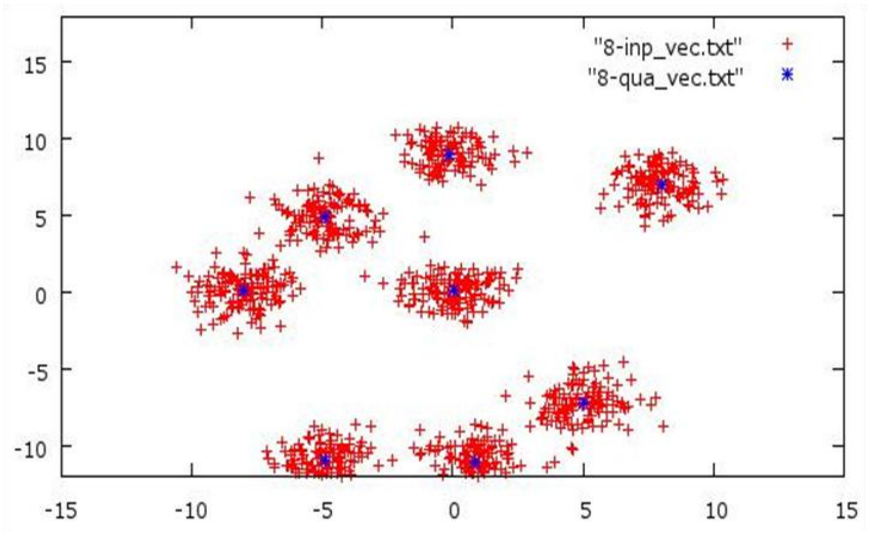


Figure 6.20 : The complete LBG algorithm

LBG (Patane & Russo, 2001) [6], and adaptive incremental LBG (Shen & Hasegawa, 2006) [7]. A simple LBG algorithm for our simulation is expressed as follows:

Step 1: Collect set of input vectors and put quantization vector (QV) at random (Fig. 6.18)

Step 2: Decide the cluster of input vectors that belongs to the nearest QV.

Step 3: Move QV towards the centroid of input vector by a small fraction of the distance. (Fig. 6.19)

Step 4: Repeat step 2 and 3 until the QV do not change.

Figure 6.18 describes an initial state of vector quantization algorithm. The input vectors are marked with a red +, and QVs are marked with blue *. Since the number of QV in clustering problem is not known a priori, the number of QV which has smallest measurement error rates in group is determined by changing the number of QV, and seeks the minimum distance of QV at a given time. Thus, the total measurement error is getting smaller. However, since one group is represented by two QVs, and two groups are represented by one QV, the QV of smallest measurement error is moved to the position of the largest measurement error if the standard deviation is large. A large standard deviation indicates the data are spread out or widely scatter condition in group. Figure 6.20 obtained by using the above mentioned LBG algorithm. Each group represents by its center point generally known as QV. Then put on noise and move QV in random direction again. In algorithm, we first initialize a threshold value to 0.1 and do repeat this above process in several times until the amount of motion QV is less than a threshold value. The convergence of LBG algorithm depends on the initial quantum vector and the threshold in implementation.

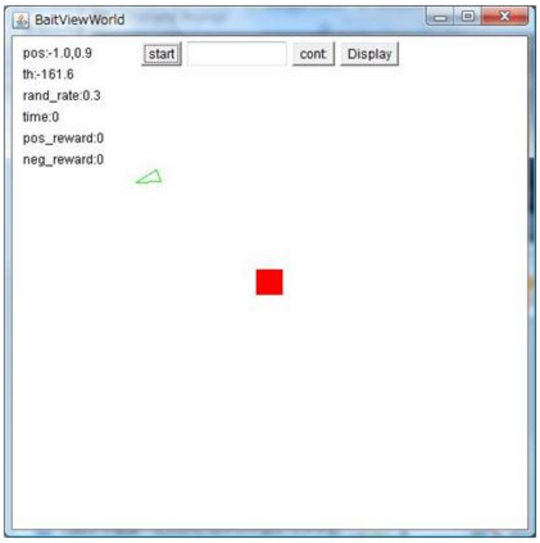
6.4.3 The Experimental results of Model I

When using experimental model I which is non-coincidence of state space and action space model to examine the learning speed of proposed addition method of VQEs, 2-types of simulation experiments are carried out.

Exp-1: Stationary reward-area at the center of the action-space (Fig. 6.21)

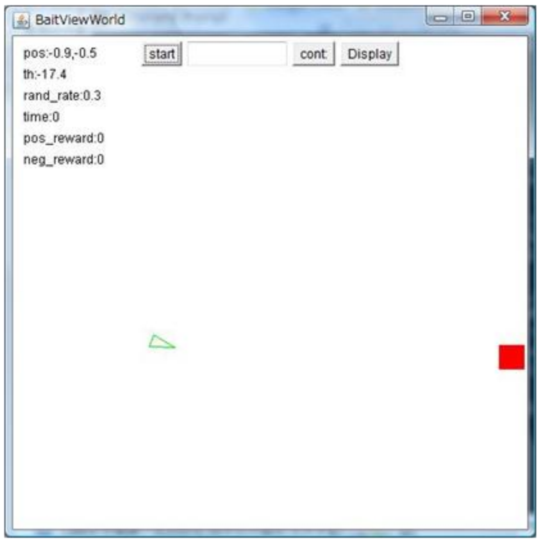
Exp-2: Moving reward-area like a circle (Fig. 6.22)

Each of the experiment were investigated using the same environment (Experimental Model I) described in Section 4.2.1 but the reward-area placed at the center of the action-space and the reward-area moving like a circle. The parameters for these tasks were set as described in above Table 6.2. We did an experiment for one time



pos:-1.0,0.9
th:-161.6
rand_rate:0.3
time:0
pos_reward:0
neg_reward:0

Figure 6.21 : Center feeding-area



pos:-0.9,-0.5
th:-17.4
rand_rate:0.3
time:0
pos_reward:0
neg_reward:0

Figure 6.22 : Random feeding-area

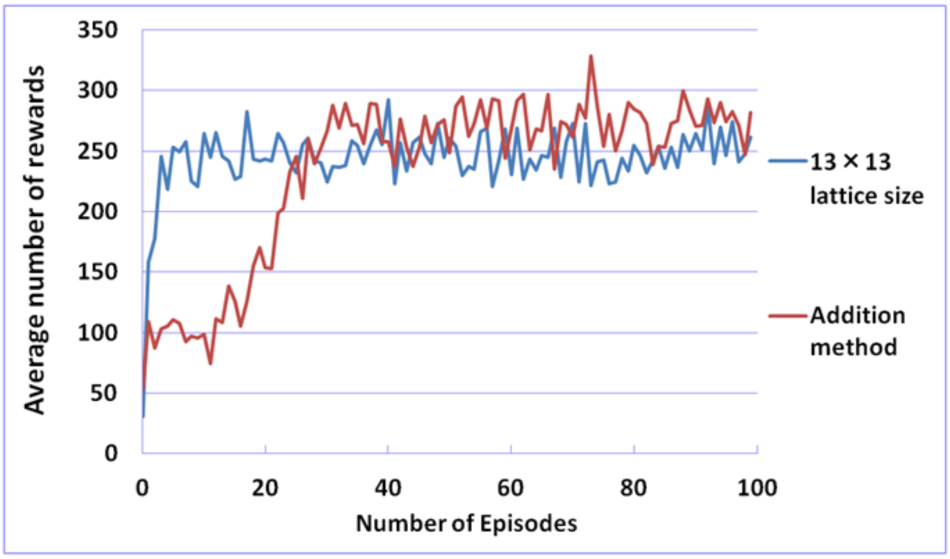


Figure 6.23 : Results for proposed addition method of VQEs and Q-Table in a 13×13 size of lattice in the case of stationary reward-area

on 10 trials by specifying proper initial seed values and take an average for those 10 trials on each episode. Moreover, we executed for 100 episodes and checked the number of rewards for each episode. In addition, we compared the learning speed of proposed

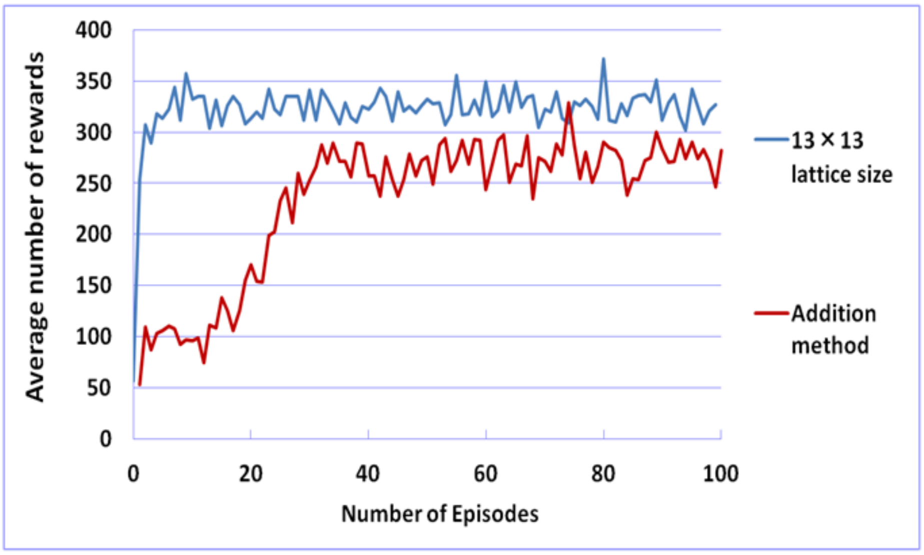


Figure 6.24 : Results for proposed addition method of VQEs and Q-Table in a 13×13 size of lattice in the case of moving reward-area like a circle

addition method and Q-Table on 13×13 size of lattice when the numbers of created VQEs have the same number of lattice size. The result of an experiment is depicted in Figure 6.23, and 6.24. As in the experimental results for Exp-1, the learning efficiency of proposed addition method is slightly improved than the learning performance of 13×13 size of lattice. On the other hand, in the result of Exp-2, the learning speed has decreased when the reward-area is moving on the action space. The results show that the behavior of the agent can be changed depend on the situation.

6.5 Conclusions

In this chapter, addition method of VQEs for position determination has been proposed and LBG algorithm is applied in this study for adaptive vector grouping of STVs. And we examined the performance of our proposed method using 2-types of experimental model in different situations. In the first-stage of VQE’s addition of both models, VQEs produce in regard to the reward-area on the state space. From the second-stage, VQEs produce in regard to the created VQEs of first-stage output.

In Model I, VQEs are generated in accordance with the number of control actions on the state space in the first-stage formation because the action space and the state space is entirely coincide each other. Moreover, we did 4-experiments using various actions such as 7, 6, 4, 3 actions. Consequently, the results have shown that the learning performance of proposed technique is much more developed than lattice in any actions.

In Model II, we examined the performance of proposed method in stationary (Exp-1) that indicates a reward-area is placed at the center of the action-space in a fixed position, and non-stationary situations of reward-area (Exp-2). In computer simulations for the non-stationary situations, it gives the decrease reward learning. Therefore, it showed that the performance of each strategy strongly depends on the behavior of agent. According to the result of number of rewards difference of stationary and non-stationary reward-area conditions, it is shown that the effectiveness of the proposed addition method in case of stationary condition.

Chapter 7

A proposition of integration method of VQEs in Q-Learning

7.1 Introduction

This chapter describes the integration of VQEs on continuous state space to reduce the number of states and memory usage in order to realize the position of VQE and to speed up the learning efficiency since the performance changes according to the arrangement of VQE. Moreover we investigate the performance and efficiency of our integration method using Model I also called “Bait View World” model. The key point of our integration method is to integrate the same optimal actions selection as shown in Figure 7.1.

7.2 Integration Algorithm

There are 5 conditions to integrate the adjacent VQEs. If

1. Two adjacent VQEs must be new added VQEs.
2. It must take the same optimal action.

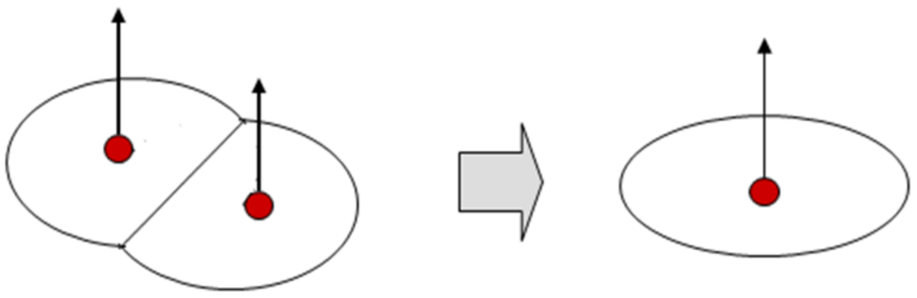
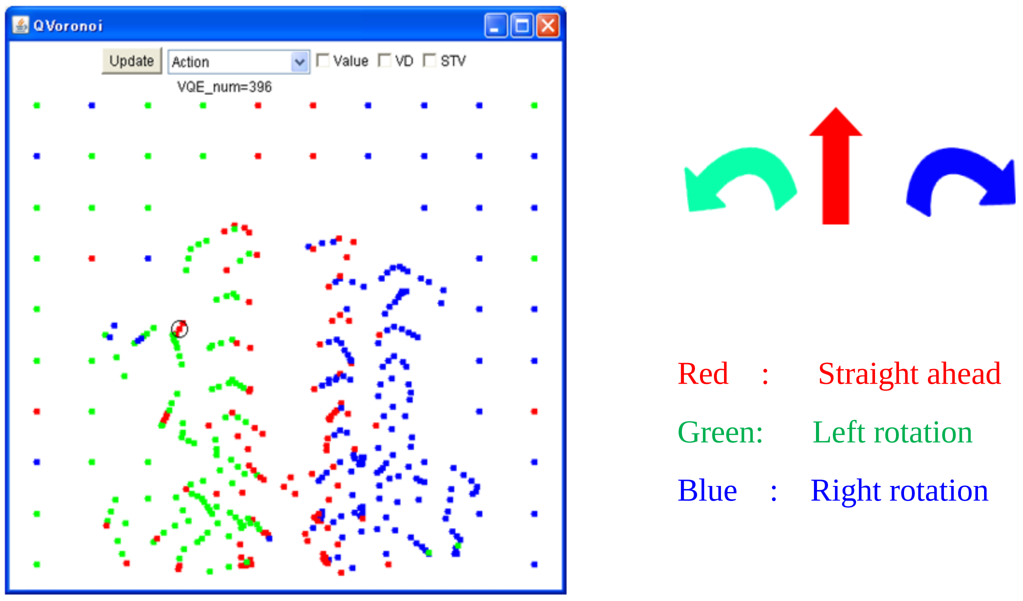


Figure 7.1 : An image of integration of VQEs

3. These optimal actions have not changed over the 10,000 action times in 1000,000 integrated timing of new added VQEs.
4. These are not already used for integration.
5. After adding new VQEs to 300 or more.

Then start the integration process and add the integration point to the center of the adjacent VQEs, and delete those two adjacent VQEs.



Red	:	Straight ahead
Green:		Left rotation
Blue	:	Right rotation

Figure 7.2 : Image of agent’s action on state space

7.3 Delaunay Tessellation Algorithm

In the previous section, we discussed about Voronoi diagrams. Now, we consider the related structure which is Delaunay tessellation technique for integration of VQEs because we want to integrate the adjacent VQEs therefore we used Delaunay tessellation technique to find the adjacent VQEs. The Delaunay tessellation is another fundamental computational geometry structure and dual tessellation of Voronoi diagram. The Delaunay triangulation is the straight-line dual of the Voronoi diagram obtained by joining all pairs of points belongs to the set. The triangulation i.e., all triangles of the Delaunay triangulation are obtained by joining the adjacent points of Voronoi diagram.

Delaunay tessellation algorithm is expressed as follows:

(1) Generate a random point.

(2) Find the closest point and 2nd closest point by measuring the distance from that random point and connect it.

(3) Repeat this process several times.

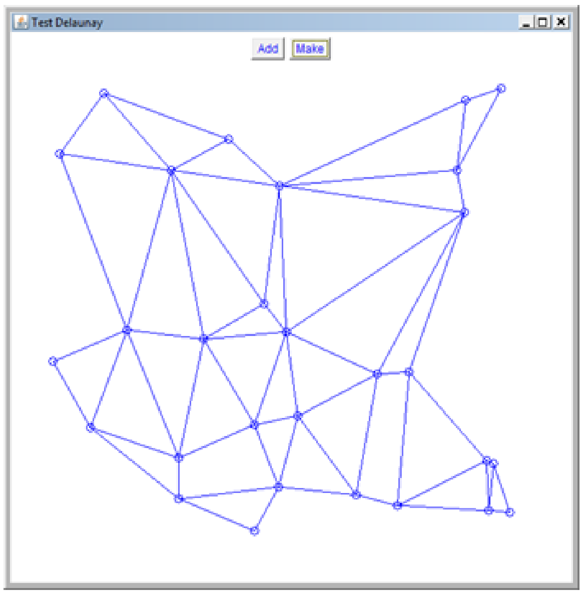


Figure 7.3 : An example of java applet animation of Delaunay tessellation

An example of java applet animation of Delaunay tessellation is illustrated in Figure 7.3. However, if the distance of point by itself is nearest point, it returns the null. Finally, all connection between the closest adjacent points is cut.

7.4 Experiments and Results using Model I

This section describes comparative experiments that were conducted on three different methods in a stationary condition of reward-area. These are given below:

(1) 13×13 size of lattice arrangement

(2) Addition method of VQEs using Model I

(3) Integration method using Model I

Size of action space	100×100
Initial value of Q	$0 \le Q(s,a) \le 0.01$ (Random)
Probability of random action	0.3
Probability of optimal action Q-value	0.7
Velocity of agent	
Straight ahead, Right rotation, Left rotation	5
Episode numbers	160 Episodes
Control action times for 1 episode	10^5 continuous action times
Learning rate α	0.1
Discount rate γ	0.9
Experiment times by changing random seed number	10 trials
Amount of agent's movement	2.0 ~ 5.0

Table 7.4 : Experimental parameter for integration of VQEs

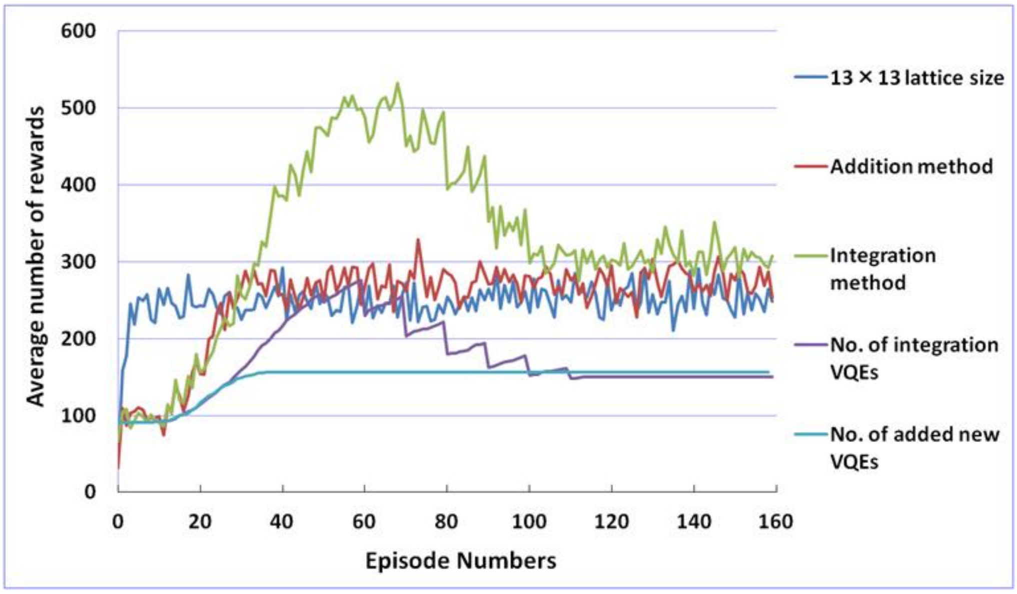


Figure 7.4 : Experimental result of 3-methods

Each method were investigated using the same environment (Experimental Model I) as the autonomous mouse robot task described in Section 4.2.1. The experimental parameters for these 3 tasks were set as described in a Table 7.3. We conducted the experiments with 100,000 continuous learning action times. It makes one "episode" and executed for 160 episodes in one "experiment". After that, we did 10 trials on each episode and take an average of these 10 trails. The learning rate $\boldsymbol{\alpha}$ was set to 0.1 and discount rate $\boldsymbol{\gamma}$ was set to 0.9. Moreover, we checked the learning performance of these three methods in stationary situation which is the reward-area is placed at the center of the action space in a fixed position.

Figure 7.4 shows the changes in the number of rewards over 0 to 160 episodes when each of the methods was used. In Figure 7.4, the average number of rewards first increase at there is many VQEs but it has decreased at less VQE in the integration method because the integration process starts after VQEs adds to 300 numbers of VQEs in the integration algorithm and reduces the number of VQEs to the same number of lattice size. Furthermore, this is thought that to be the reason that sometimes the agent may take the wrong action. For example, in the above Figure 7.2, the agent took left

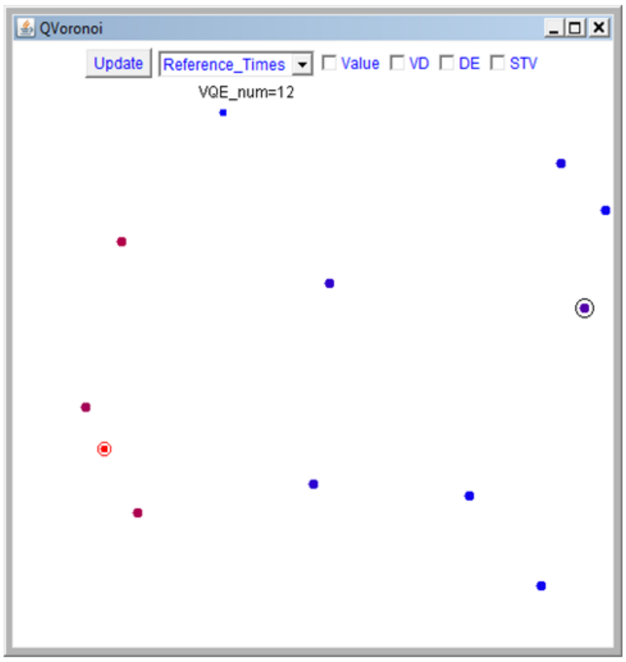


Figure 7.5 : Image of over-integration of VQEs on continuous state space

rotation instead of taking the right rotation. However, we can say that the integration method develops the learning efficiency than other 2 methods when the numbers of add VQEs and integration VQEs are same with t with the number of 169 (13×13 lattice size).

However in this simulation, Q-value is used only one-side of adjacent VQEs, and the agent did not take the state transition vector. According to the result of experiment, it is shown that the integration method improves the quality of learning and can reduce the number of states. Nevertheless, we occurs the over-integration problem in this integration method though the number of states has decreased shown in Figure 7.5. Therefore, we are still considering about this problem and will figure out as future challenges.

Chapter 8

Conclusions

In this thesis, we first presented Voronoi Q-value Element (VQE) using the concept of Voronoi space division to be able to apply the normal Q-Learning in continuous state space and to solve the Curse of dimensionality problem that described in Section 2.7.1. The advantages of VQEs are given in Section 3.2.4. Then the performance of proposed VQEs with radius on 4-dimensional space is examined by computer simulations on competitive situations of lattice which is Q-Table with previous work in Section 4.3. The results have shown that the proposed method is much more efficient than Q-Table to apply the Q-Learning in continuous state space.

In order to examine the efficiency of the proposed method, we briefly explained 2-types of experimental model which is Model I and Model II. These models are based on continuous states and discrete actions of feeder mouse. The experimental model I is non-coincidence of state space and action space model, and the experimental model II is coincidence of state space and action space model. There is the action space which the agent is learning toward the goal-area to get as much reward as during a specific period of time. The state space which is partitioning the continuous state space into Voronoi regions using the concept of Voronoi diagram, and it demonstrate the reference times of VQEs, kinds of agent’s action selection with color, and the value of Q.

In Chapter 5, we investigated the effectiveness of VQEs using several strategies of Model I and Model II, and the simulation results are showed. From these experiments, we realize that the Q-Learning using VQEs gives a good result but it has a position determination problem because of the Voronoi diagram has a lot of flexibility. Moreover, from these experiments of Model II, the optimal position of VQEs is obtained from the group of state transition vector of each action. Therefore, we have proposed the addition method of VQEs to solve this position determination problem in Section 6.2, and it also aims to show the improvement of a learning efficiency. Thus, we checked up the learning performance of proposed method using 2-types of experimental model in Section 6.3 and 6.4 which are different environment. Furthermore, it is compared to the existing method which is lattice. After that, we showed that proposed addition method is slightly improved than Q-Table. In the addition method, LBG vector quantization algorithm is used for adaptive state transition vector grouping.

In Chapter 7, we presented the integration method of VQEs to reduce the number of states and to speed up the learning efficiency. In this integration method, Delaunay tessellation algorithm is used for integration of adjacent VQEs. Then the performance of lattice, addition method, and integration method were examined by computer simulations using only experimental Model I with the reward-area place at the center of the action space in a fixed position. According to the results of experiment, the integration method can increase the total amount of rewards and slightly improved than Q-Table. Moreover, it can reduce the number of states greatly.

Summary

This thesis mainly presents a study on state space partitioning using Vorornoi diagram based on Q-Learning algorithm with the use of VQE. Here we manly present a study on solving curse of dimensionality problem conducting the normal Q-Learning on continuous state space in a single-agent environment. We aim by this research to speed up the learning efficiency in different situations as well as decrease the learning time. In order to do that we proposes VQE in various method such as Voronoi space division, rotating VQE, addition of VQE, addition and integration of VQE, etc. in several versions. In addition we present a better performance of learning for the algorithm.

Chapter 1 provides an introduction explains briefly the historical development the ideas to find alternatives to computer simulation and how it leads to think about the Q-Learning algorithm in continuous state space as a strongly possible alternative and the motivation for learning. This chapter also describes the aim and objectives of the study, and specifies the dissertation structure.

Chapter 2 gives an overview or the basic concepts and ideas related to this research namely the basic principles for the reinforcement learning such as the standard reinforcement learning model. Section 2.2 defines a technique for people to realize the learning ability, and to automatically perform what kind of action should take by computer machine in order to maximize the expected value of future reward in unknown environment. Section 2.3 describes the problem of reinforcement learning, and defines some classic model-free algorithms for reinforcement learning from delayed reward: Markov decision process, and value function. In section 2.4, the most important aspects of normal Q-Learning algorithm which is a typical technique of reinforcement learning was described. Section 2.5 also describes the action selection method of agent, and section 2.6 describes the Q-Table which divides the state space into lattice and the division method of Q-Table. Moreover, section 2.7 discusses the curse of dimensionality which increases the number of states exponentially in high dimensions. Furthermore,

section 2.8 gives and shows the result of Q-Block that execution times takes about twice than Q-Table when we used Q-Block to solve the curse of dimensionality problem.

Chapter 3 describes the VQE (Voronoi Q-value Element) which divides the state space using the concept of Voronoi space division in order to solve the above problem, and also gives the idea of Voronoi diagram, and how Voronoi diagram could be used to partition the state space. Although Q-Table needs to prepare Q-value in all states beforehand, VQE can be added to the state space as required. Section 3.1 describes the Voronoi division which is the division method of space whether arbitrary points being the closest to which mother point with respect to the mother point located on space. Section 3.2 presents the creation method of VQE, a reference method of VQE, method of space division using VQEs, and the advantages of Q-Learning that used VQE are enhanced learning speed and reliability for this task, and the essential characteristics of VQEs in a continuous state space are also described. This chapter also explains several methods of nearest neighbor search.

Chapter 4 evaluates the effectiveness of proposed Q-Learning technique by using VQEs, and performs the computer simulations as a comparison experiment of Q-Table that described in previous section 2.6 and VQEs. And 2-types of experimental models which are Model I and Model II are explained in Section 4.2 for full details. These 2-models are based on continuous states and discrete actions of feeder mouse (Esa-Hiroi Mouse). After that, we show the better performance using VQEs on continuous states and discrete actions for 4-dimensional spaces by comparing the normal Q-Learning (Q-Table) and Q-Learning with the use of VQEs. In addition, the conclusions are considered.

Chapter 5 examines the learning performance of various strategies using 2-types of experimental model I and model II with reward-area in a stationary situation in single-agent environment and decide how to act in certain state. In order to test our hypotheses, we experimented by rotating the angles of agent’s actions, angles of VQEs by the angle in 5 times interval between 0 degrees and 90 degrees in which VQEs are arranged in a lattice structure. Moreover, a random arrangement of VQEs experiment also conducted to correctly evaluate the optimal Q-values for state and action pairs in order to deal with continuous-valued inputs. As a result of experiments using experimental model II, the learning speed has most increased when the angles of VQEs and angles of actions is just 45 degrees out of alignment in case of 4-actions.

Chapter 6 presents the addition method of VQEs which is a position determination method to decide the position of VQEs in order to realize a Voronoi region since the performance of Q-Learning changes according to the arrangement of VQE. Moreover, the simulation was performed in both experimental models and the learning performance was examined. And also presents block-counting method and a new adaptive segmentation of continuous state space based on vector quantization algorithm such as LBG (Linde-Buzo-Gray) for high-dimensional continuous state spaces. The objective of adaptive state space partitioning is to develop the efficiency of learning reward values with an accumulation of state transition vector (STV) in a single-agent environment. Moreover, the study of the resulting state space partition reveals in a Voronoi tessellation. In addition, the experimental results show that this proposed method can partition the continuous state space appropriately into Voronoi regions according to not only the number of actions, and achieve a good performance of reward based learning tasks compared with other approaches such as square partition lattice on discrete state space.

Chapter 7 describes an algorithm of integration of VQEs to reduce the number of states, the memory usage and the learning time. It also aims to improve the performance of learning efficiency. Then it proceeds and described the topological structures of Delaunay network to find the adjacent VQEs for integration on continuous state space. We add VQEs on state space, and integrate which has the same optimal action selections. A computer simulation has been performed using experimental Model I, and the simulation results are explained compared with 3-methods such as lattice of a previous method which is Q-Table, addition method of VQEs, and integration method of VQEs with the reward-area in a stationary condition only

Bibliography

[1] Richard S. Sutton and Andrew G. Barto, Reinforcement Learning: An Introduction Bradford Books, MIT Press, Cambridge, USA, 1998.

[2] Watkins CJCH, Dayan P. Technical notes: Q-learning. Machine Learning, 1992; 8:279-292.

[3] Chris Gaskett, David Wettergreen, Alexander Zelinsky, Er Zelinsky; Q-Learning in Continuous State and Action Spaces: 12th Australian Joint Conference on Artificial Intelligence, ISBN:3-540-66822-5, Vol-1747, 1999.

[4] Tomoki Hamagami, Seiichi Koakutsu, and Hironori Hirata, An adjustment Method of the Number of States on Q-Learning Segmenting State Space Adaptively 電気情報通信学会論文詩, D-I Vol.J86-D-I, No.7, pp.490-499, 2003.

[5] Christopher J.C.H Watkins, Learning from Delayed Rewards; Ph.D thesis, University of Cambridge, 1989.

[6] G.Patane and M.Russo, The enhanced LBG algorithm, In Proceeding of Neural Networks, 2001. pp. 1219-1237.

[7] F.Shen, O.Hasegawa, An adaptive incremental LBG for vector quantization, Neural Networks, 2006, 694-704.

[8] Goto R, Matsui T,k Matsuo H. State generalization with support vector machines in reinforcement learning. 4th Asia-Pacific Conference on Simulated Evolution and Learning, 2002; I:51-55.

[9] Christopher J. C. H Watkins: “Q-Learning”, Kluwer Academic Publishers, Boston. Manufactured in The Netherlands. Machine Learning, 8, 279-292, 1992.

[10] Gavin Adrian Rummery; Problem solving with reinforcement learning: Ph.D thesis, Cambridge University, 1995.

[11] Juan C. Santamaria, Richard S. Sutton and Ashwin Ram; Experiments with reinforcement learning in problems with continuous state and action spaces Adaptive Behavior, 1998.

[12] Baird, L.C. and Klopf, A.H, Reinforcement learning with high-dimensional continuous actions, Technical Report WL-TR-93-1147, 1993.

[13] Ivan S.K. Lee, Henry Y.K. Lau, Adaptive state space partitioning for reinforcement learning, Engineering application of Artificial Intelligence 17, 2004, 577-588.

[14] Takahashi Y, Asada M. State-action space construction for multi layered learning system. J Robot Soc Japan, 2003; 21: 164-171.

[15] Kazuyuki Fujita and Hiroshi Matsuo, Multi-agent reinforcement learning with the partly high-dimensional state space, System and Computers in Japan, Vol.37, No.9, 2006.

[16] Kazuo Kiguchi, Hui He, and Kenbu Teramoto, A study on multi-dimensional Fuzzy Q-learning for intelligent robots; International Journal of Fuzzy Systems, Vol. 9, No. 2, June 2007.

[17] Kathy Thi Aung, Takayasu Fuchida, “Reinforcement learning using Voronoi space division”, Vol-15, Artificial Life and Robotics, 2010.