SELF-CONSTRUCTION OF STATE SPACES OF SINGLE AND MULTI-LAYERED LEARNING SYSTEMS FOR VISION-BASED BEHAVIOR ACQUISITION OF A REAL MOBILE ROBOT

Yasutake Takahashi

January 2002

SELF-CONSTRUCTION OF STATE SPACES OF SINGLE AND MULTI-LAYERED LEARNING SYSTEMS FOR VISION-BASED BEHAVIOR ACQUISITION OF A REAL MOBILE ROBOT

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF
MECHANICAL ENGINEERING FOR COMPUTER-CONTROLLED MACHINERY FACLTY
OF ENGINEERING
AND THE COMMITTEE ON GRADUATE STUDIES
OF OSAKA UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

By
Yasutake Takahashi
January 2002

Minoru Asada
(Principal Adviser)

Yoshiaki Shirai
(Dept. of Computer-Controlled Mechanical Systems,
Graduate School of Engineering, Osaka University)

Tadahiro Kitahashi
(The Institute of Scientific and Industrial Research, Osaka University)

Approved for the University Committee on Graduate Studies:

Dean of Graduate Studies & Research

Preface

This thesis tells you all you need to know about almost works in which Yasutake Takahashi has engaged from 1994 to 2001. These works aim at building an autonomous robot which is able to develop its knowledge and behaviors from low level to higher one through the interaction with the environment in its life.

Acknowledgments

I would like to thank my adviser, Professor Minoru Asada for his support, his patient guidance, and constant encouragement throughout this work. I would like to thank all members of laboratory for their helpful assistance and discussion. I would like to thank my family, especially my parents Minoru and Hiroko for all their love, understanding, and patience.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In this chapter, a brief overview of conventional approaches to behavior acquisitions for autonomous robots is given in order to make clear the purpose of this dissertation.

1.1 Autonomous Mobile Robot In the Real World

There are a lot of works on development of autonomous mobile robots in the real world. The given tasks and the environments around the mobile robots are various such as map building, objects delivering, navigation in the office room, under water, desert area, and so on. Many approaches are also proposed to design robot behavior to control robot motions and/or to improve them.

The conventional approaches require the robots to accomplish specific tasks. The purposes of their works are to acquire a certain behavior automatically, or to improve its efficiency. However, the demands for the autonomous robots would be increasing, or changed after the behavior developments. Therefore, the robots are expected to require much more adaptability to work in the dynamically changing real worlds.

In order to develop a autonomous robot, usually human designers write programs of behaviors for a robot in advance based on their own experiences and insights. Generally speaking, the monolithic, deliberative behavior controller needs a complicated environment-robot system model, large computational resources to derive a solution, and careful attention to its maintenance. It is also difficult to be robust against breakdowns of the mechanical devices or changes of the environment because of its lack of a adaptation capability. Even though human designers are able to adjust parameters of the behaviors, it is really hard and time consuming because the number of parameters tends to be easily huge, the parameters have complex relationships with each other,

and it is almost impossible to know the characteristics of the whole system and/or to predict all kinds of changes in the environment. Therefore, autonomous robots need learning capability when various kinds of tasks are given, not only in the familiar environments but also in newly encountered ones.

One of the main concern about autonomous robots is how to implement a system with learning capability to acquire both a variety of knowledge and behaviors through the interaction between the robot and the environment during its life time. There have been a lot of works on learning approaches for robots to acquire behaviors based on the methods such as reinforcement learning, genetic algorithms, and so on. However, direct applications of these methods have still limitations to cope with change in the tasks/environment.

The objective of our study is to cope with such limitations by acquiring both knowledge and behaviors through the interaction between the robot and the environment, within a reasonable learning time, with human designer's help as little as possible. We have realized our ideas and applied them to the real robot on the RoboCup as test-bed to evaluate our systems.

The Robot World Cup Soccer Games and Conferences (RoboCup) are a series of competitions and events designed to promote the full integration of AI and robotics researches. The robotic soccer provides a good test-bed for evaluation of various kinds of research disciplines, e.g. artificial intelligence, robotics, image processing, system engineerings, multi agent systems, and so on. Reactive and adaptive behaviors of robots are required in a dynamic and hostile environment which is one of the most important characteristics provided by RoboCup competitions.

There are two major approaches related to our research. One is so-called behavior-based architecture approach, and the other is machine learning one. We briefly review them.

A Behavior-Based Architecture

Brooks [10] proposed a behavior-based architecture, called "subsumption architecture", which is a kind of layered control system. Layers consist of asynchronous modules, each of which is a fairly simple behavior controller. The higher layers can subsume the lower behaviors by suppressing their outputs. The simplicity of each control module makes its development and maintenance easy, and leads the whole system to be flexible and robust. His group invented several kinds of behavior-based robots [11].

The simplicity of control modules also means that they have the capability of reactive behaviors. They do not need enormous computational resources (memory and processing power) to generate adequate behaviors. This property is the most

important one in order to develop real autonomous robots. However, we still need a capability of a purposive behavior acquisition of each module, and the design principle of the hierarchy system.

Machine Learning Approach

In the machine learning area, several approaches have tried to make agents learn purposive behaviors autonomously to achieve their goals through agent-environment interactions. Especially, reinforcement learning has recently been receiving increased attention as a method for behavior learning with little or no a priori knowledge and higher capability of reactive and adaptive behaviors [15].

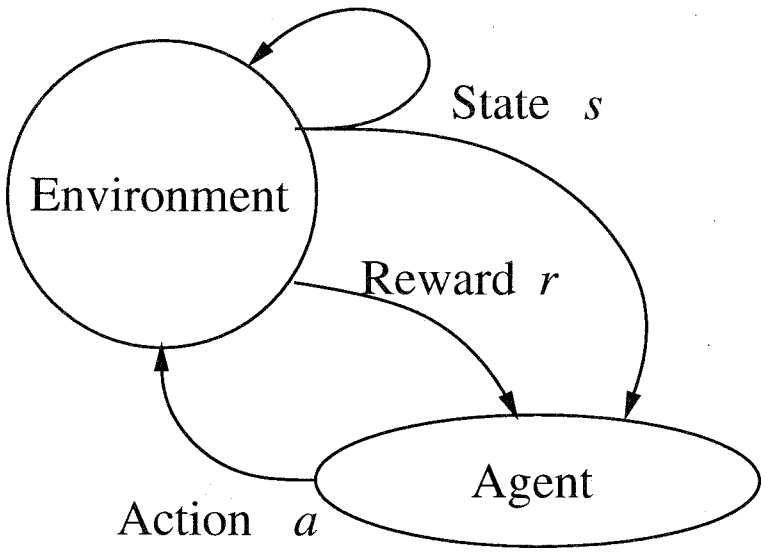


Figure 1.1: a basic model of agent-environment interaction

Figure 1.1 shows a basic model of reinforcement learning. We assume that the agent can discriminate a set $\boldsymbol{S}$ of distinct world states, and can take an action from an action set $\boldsymbol{A}$. The world is modeled as a Markov process, making stochastic transitions based on its current state and the action taken by the agent. For each state-action pair (s, a), the reward $r(s, a)$ is defined. The general reinforcement learning problem is typically stated as finding a policy that maximizes the discounted sum of the reward received over time. Watkins' Q-learning algorithm [14] gives us an elegant method to do this.

However, the applications of conventional approaches based on machine learning methods seem limited, such as the navigation on the grid world ([71, 74, 73]), object manipulation in a toy world ([72]), and the traditional control domain like car-on-the-hill ([9, 60, 61]), pole balancing (Pendulum Swing Up) ([61, 20, 21, 8, 48]), while only a small number of real mobile robot applications have been reported which are simple and less dynamic [15]. The concept of reinforcement learning would be meaningful if it could be applied to more complex tasks in a real environment.

1.2 Issues of Applying RL to the Real Robots

There are a number of advantages in applying reinforcement learning methods in order to develop an autonomous robot (ex. [7, 4]). However, they have also one critical disadvantage, that is **the lack of the scalability**.

Since the reinforcement learning generally begins to updates the action values from the state given the reinforcement, the experiences before the reinforcement is given or far from the states where the reinforcements are given might be vain. This leads the learning system to take enormous time to have adequate experiences to acquire a purposive behavior over large state and action spaces. Whitehead [71] has theoretically shown that learning converges very slowly with such infrequent rewards; the search time scales at least exponentially in the step size to reach the goal state from start one in the state space. This theoretical result has been also confirmed in many experimental studies (ex.[43, 17, 41]).

There are two major approaches to solve the lack of scalability in applying reinforcement learning methods to real autonomous robots.

1. **Keeping the state and action space small enough.** Many experimental studies show that the reinforcement learning system works very well with a small state/action space. It is an easy solution to keep the state and action spaces small enough to acquire a purposive behavior within reasonable learning time.

   However, the construction of the state and action spaces is also an important issue in the reinforcement learning system. Even though we have already known that the reinforcement learning system works based on the well defined state and action spaces, it is really hard for human designers to construct such state and action spaces in advance, especially in a scaled up, complex task/environment, therefore the robot should find the adequate state/action space by itself.

2. **Introduction of hierarchy and multi-module architecture.** In order to realize an autonomous robot which acquires various behaviors by itself in real

world, it is necessary to be able to manage a wide range of state and action variables according to situations, to keep the spaces as small as possible, and to learn/control behaviors based on the small state and action spaces.

It is almost impossible or impractical that robot acquires the various behaviors for the given tasks based on a huge monolithic state/action space which consists of all kinds of sensory information and actuator commands, because the computational resources are limited and learning time is not eternity from a practical viewpoint.

Fortunately, almost long time-scale behavior may be generally decomposed into some simple behaviors so that the searching space could be divided into some smaller ones. Connel and Mahadevan [34, 33, 16] decomposed the whole behavior into sub-behaviors each of which can be independently learned. However, task decomposition and behavior switching are designed by the human programmers.

Above two approaches have dependence between them. A multi-module architecture enables the system to decompose a large state-action space into small subspaces, based on which the reinforcement learning system can generate purposive behaviors within reasonable learning time. On the other hand, the state and action space of higher level learning modules can be small because it would consist of abstracted states and actions based on the already learned control modules on the lower levels. In this thesis, we put focus on the self-construction of state space of single and multi-layered learning systems.

1.3 State and Action Space Construction

Generally, the state and action spaces are dependent on each other. That is, the design of the state space in which necessary and sufficient information to accomplish a given task is included depends on the capability of agent actions. On the other hand, the design of the action space also depends on the capability of perception. This resembles the well-known "chicken and egg problem" that is difficult to be solved (see Figure 1.2).

There are two major problems in the construction of the state space.

1. Selection of the information to represent a state of the agent and its environment, and to take an action against the state

2. Definition of state/action based on the selected information

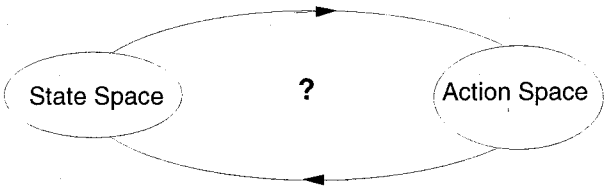


Figure 1.2: The inter-dependence between the state and action spaces

In the following, we explain these problems in more detail in the cases of single and multi-layered learning systems.

1.3.1 Spaces for A Single Learning Module System

Almost works about the state and action space construction are based on sensory information and actuator commands equipped on the robot. In a case of the single learning module system, there are following two major problems in the construction of the state space.

1. It is difficult to select the sensory information to represent the state of robots and their environment. If one uses all the sensory information, the amount of the data the robot has to deal with would easily exceed the capability of the robot (memory and processing power). This is called "curse of dimension" problem. This means that the computational cost must increase exponentially with the dimension of the search space.

2. Even though the sensor information is well selected for the given task, the segmentation problem remains. The state space designed by the programmer is not guaranteed as an optimal one for the robot to perform the task. The coarse segmentation causes so-called "perceptual aliasing problem" [72] by which the robot cannot discriminate the states important to accomplish the given task. On the other hand, the fine segmentation to avoid the perceptual aliasing problem produces too many states to generalize the experiences. Further, the robot needs an enormous amount of learning time since it increases exponentially with the size of the state space.

A simple and straightforward method to handle the above problem is to gather many data of the interactions between the environment and itself, to selects the information related to the given task, and then to quantize this space in a certain way. However, this approach seems limited because the robot has to have all kinds

of experiences in the environment before the it tries to execute the given task. One alternative to avoid such a limitation is an adaptive, online construction of the state and action spaces in applying reinforcement learning methods to real autonomous robots.

1.3.2 Spaces for Multi-Layered Modules Architecture

Another approach to the problem of the curse of dimension and the perceptual aliasing is to adopt a hierarchical structure within leaning control system. That is, the system

1. prepares learning/control modules of one kind each of which deals with a subspace divided from a whole state/action space,

2. abstracts situations and behaviors based on the acquired learning/control modules, and

3. acquires higher level, new behaviors based on the state and action spaces which is newly constructed from the abstracted situations and behaviors.

This approach can suppress the explosion of state and action spaces since the higher level learning/control system manages adequately small size spaces already abstracted in the lower levels.

The state and action space construction is also one of the important issues in the multi-layered learning system. The system must construct a state/action space for learning/controller modules of all layers based on the situations and the behaviors constructed by the lower modules. The bottom layered modules have state and action spaces based on the sensory information and motor commands. To the best of our knowledge, there is no research on the construction of state and action spaces in the multi-layered modules architecture.

In the multi-layered learning system area, there are following two major problems in the construction of the state space.

1. Selection of the information from the lower learning modules to describe the situation of the agent on the environment. Even though the lower modules abstracted the situations and behaviors, information selection is still important in order to construct a meaningful state space for the higher layer modules.

2. Definition of state based on the information of selected lower modules. It is easily caught by a curse of dimension if the system combine the all kinds of information as a set of their direct products.

1.4 The objective of the dissertation

The purpose of our study is to develop methods for an autonomous robot with learning ability of knowledge and behaviors through the interaction between the robot and the environment within a reasonable learning time with human designer's help as little as possible.

This dissertation shows ideas on the state and action space construction in the single and multi-layered learning systems and the concrete models of the hierarchical control system for an autonomous robot, and their verifications through the real robot experiments.

1.5 The organization of the following chapters

This dissertation consists of six chapters including this one. In the next chapter, we shows a survey of related works. In chapter 3, we propose a method of state space construction for a single layered learning system. In chapter 4, we propose a method by which a hierarchical structure for behavior learning is organized by itself. In chapter 5, we propose an approach to the problem of decomposing the large state space at the bottom level into several subspaces and to merge these subspaces at the higher level. Finally, we conclude this study in chapter 6.

Chapter 2

Related Works

One of the main concern about autonomous robots is how to implement a system with learning capability to acquire both a variety of knowledge and behaviors through the interaction between the robot and the environment during its life time. There have been a lot of works on learning approaches for robots to acquire behaviors based on the methods such as reinforcement learning, genetic algorithms, and so on. Especially, reinforcement learning has recently been receiving increased attention as a method for behavior learning with little or no a priori knowledge and higher capability of reactive and adaptive behaviors.

Since the lack of its scalability prevents the reinforcement learning approach from being applied to real robot systems, many researchers have proposed several approaches to extend the scope of reinforcement learning applications. We categorize them into four major approaches;

1. "state space construction for a single learning module",
2. "multi-module based reinforcement learning",
3. "reuse/re-adaptation of acquired knowledge", and
4. "hierarchical control structure".

In the following, we survey the areas related to the above.

2.1 Low Level State Space Construction

There are two major problems in applying reinforcement learning to real robot tasks. One is a selection problem of the sensor information to represent the state of robots

and their environment, and the other is a problem of the state space segmentation.

2.1.1 Selection of sensor information

There are several methods for selection of sensory information. Whitehead and Ballard [72] proposed a selection method of the sensor information to avoid the perceptual aliasing, called "lion algorithm". The idea is that the overall decision system has a perceptual subcycle to identify the situation in order to suppress the perceptual aliasing. This system was applied to a simple block manipulation task in a computer simulation.

Tan [63, 64] proposed a cost-sensitive learning method. The cost-effective sensing features of the external world are selected in order to build a consistent internal state representation. The method is demonstrated under a navigation task in a grid world. Sensing a distant state needs more cost than sensing a nearby one.

Chapman and Kaelbling [13] proposed "G-algorithm" which recursively splits the state space based on statistical measures of the differences in reinforcements received. They applied their method to an action video game environment in which the sensory information is described as one hundred of bits.

Munos [42] proposed a method to build a kind of decision tree according to some reinforcement dissimilarity criterion. The sensory information are symbolized in advance, and the environment is a grid world in which the given task for the agent is foraging for food with obstacle avoidance. Even though the perceptive capacities are limited to the local area around the agent, it seems easy to obtain an adequate policy because the foods are distributed aboundingly and the agent receives reward frequently in this circumstance.

McCallum [36, 37, 35] proposed a method which uses selective attention and short-term memory to deal with the sensor selection problem and hidden state one. The method is applied to solve a highway driving task. The sensory information is symbolized well.

The conventional approaches have dealt with the discrete state space, therefore, these methods are difficult to be directly applied to a continuous state space in a real world.

2.1.2 Sensor Space Segmentation

For the segmentation problem, roughly speaking, there are two approaches: learning the value function with a method of function approximation or with segmentation of continuous state space.

There are a number of studies of the former approach based on artificial neural networks. Lin [29] has shown reinforcement learning methods based on neural network; AHC-learning and QCON are based on TD(λ) method and Q-learning one, respectively. He applied those methods to the maze exploration problems.

Boyan and Moore [9] reported on the method for the function approximation that the combination of dynamic programming and function approximation had shown poor performances even for benign cases under several experiments such as a 2D grid world, a puddle world, and a mountain car. Then, they proposed Grow-Support algorithm for the function approximation. However, they need the environmental model and assume a deterministic state transition on the world.

Sutton [60] has used a sparse, coarse-coded function approximator known as the CMAC (Celebellar Model Articulation Controller) [2, 1] as a method of the function approximation although CMAC has its own problem of quantization (segmentation), He has shown the proposed method can learn the purposive behavior robustly on the three continuous-state control problems studied by Boyan and Moore. Ono and Fukumoto [44] have also applied the Q-learning and CMAC combination to the multi-agent block pushing problem and shown a successful result.

Saito and Fukuda [46, 47] used CMAC to estimate the Q values, and applied it to the two link robot brachiation system. However, the sensor space was huge and they needed enormous learning time, therefore they reduced the searching space by using the initial controller.

As a method for state space segmentation, Kröse and Dam [32], Dubrawski and Reignier [23], and Sato et al. [49] used only reinforcement signal to divide the state space, therefore the space far from the states given the reinforcement signal have not been segmented.

Moore [39] proposed Variable Resolution Dynamic Programming, which produces a partitioning with high resolution only in important regions. This is one kind of off-line dynamic programming methods. It calculates a mental trajectory using the prepared forward model, and quantizes the area of the state space in which the trajectory passed over with highest resolution, under the assumption which is that those states encountered during mental practice are particularly important to know about detail. However, the system does not consider how fine the resolution should be. It has possibilities to produce a partitioning with full resolution if the agent explore the state space uniformly.

Moore and Atkeson [40] proposed a partitioning method for continuous observation space. However, this method is formulated only for the shortest path problem of robot navigation and requires the pre-defined local controllers for transition between partitioned states.

Asada et al. [5, 6] proposed a method which cuts off regions from the sensory space

as states recursively from the goal state. The construction process corresponds to behavior learning, and as a result the purposive behavior is acquired, simultaneously. They applied the method to a ball shooting behavior using vision system on a real robot. In order to construct the state space suitable for the robot to perform the given task, they need a sufficient amount of uniformly sampled data.

Summary

Until we started the study on state-space construction, there are few works which divide a sensor space by the robot itself. The incremental segmentation approaches need immediate reward information to evaluate the division of the sensory space, and this assumption limits the application of the real robot tasks such as room wandering behavior with obstacle avoidance. Though the off-line segmentation approaches provided sophisticated methods which manage segmentation process and action planning one simultaneously, the presumption which learning agent has to experience all kinds of situations in the environment and store the all data of sensory informations and motor commands is impractical. The function approximation approach has been shown as no robust in several experimental studies. Only CMAC can approximate successfully state action value function, however, it has its own problem of quantization.

2.2 Multi-Module based Reinforcement Learning

The lack of scalability of reinforcement learning algorithm has limited its applications to simple tasks. One of the approaches to applying it to tasks with large state and action spaces is an introduction of multi-module structure in the system. Fortunately, almost long time-scale task might be generally decomposed into some simple subtasks so that the searching space could be divided into some smaller ones. We can assign learning modules to subgoals, then switch them while the system executes the long time-scale sequential tasks. The learned policies for the subtasks are reusable even if the sequence of the given subtasks has changed.

Singh [53, 50] has proposed Compositional Q-learning in which an agent learns multiple sequential decision tasks with multi learning modules. He assumes that a complex, sequential task is built up in a systematic way from simpler elemental tasks. Each module learns its own elemental task. On the other hand, the system has a gating module for the sequential task, and this module learns to select one of the elemental task modules. He has applied his method to navigation tasks in a grid

world. He assumes that the reward is zero in all state except for the goal state of elemental tasks.

Tham and Prager [68] have improved Singh's method so that the reward function can be non-zero in states other than the desired final states of elemental tasks, and extended the elemental task modules using CMAC in order to handle a large state/action space. Their method is applied to a reaching task with a two-link manipulator.

Mahadevan and Connell [34, 33, 16] described a system that decomposes a single complex activity into a series of sub-activities, each of which is learned by an individual module. They applied their method to the box pushing task using a real mobile robot. They prepared three subtasks of the box pushing task such as "finding a box", "pushing a box", and "getting unwedged", and coordinated the learned behavior in the subsumption architecture manner.

Whitehead et al. [74] have also proposed one of the modular architectures. Their objective is to study a system that coordinates behaviors in order to pursue multiple, independent goals in parallel. They have addressed the issue of behavior coordination between parallel competing goals.

Ono and Fukumoto [43, 45] have also proposed a very similar method to the Whitehead's one. They prepared subspaces based on the selected sensory information, assigned them to learning modules. The all learning modules try to acquire the purposive behavior for the same task. The system selects the module which will return the highest action value. The Whitehead's method aims to acquire multiple, independent behaviors, and coordinate them for the parallel goals. On the contrary, the Ono's method is designed to reduce the searching space by dividing the enormous state space into subspaces.

Carrearas et al. [12] also proposed another behavior coordination method. The proposed method has a number of behavior learning modules based on reinforcement learning algorithm with a feed-forward neural network, and the coordinator blends the outputs of all modules based on the activation level of the behaviors and the predefined priority. The merit of this coordinator is taking an advantage of both the competitive and cooperative coordination systems; the competitive coordinators show good robustness in the behavior selection, while the cooperative ones show the smoothness of the behavior changes. The demonstration of application to an autonomous underwater vehicle was given.

Doya et al. [22] have also proposed MOdular Selection and Identification for Control (MOSAIC), which is a modular reinforcement learning architecture for nonlinear, non-stationary control tasks. The basic idea is to decompose a complex task into multiple domains in space and time based on the predictability of the environmental dynamics. Each module has a state prediction model and a reinforcement

learning controller. The models have limited capability of state prediction as linear predictors, so the multiple prediction models are required for the non-linear task. A domain is specified as a region in which one linear predictor can estimate with its own prediction capability. The responsibility signal is defined by a function of the prediction errors, and the signals of the modules define the outputs of the reinforcement learning controllers. Haruno et al. [25, 26] have proposed another implementation of MOSAIC based on multiple modules of forward and inverse models.

Summary

The most important problem on the multi-modules reinforcement learning is that subgoals may require a significant amount of a priori knowledge about the task domain. In the all of above methods, human designer have to define the subtasks based on their own experiences and insights.

Doya et al. and Haruno et al. show interesting task decomposition methods, which do not need any designer's help to decompose a complex task into multiple domains in space and time. The key idea is that the decomposition should be done based on the controller capability. This idea is similar to the concept of our proposed approach.

2.3 Reuse of Acquired Knowledges

Reuse of the learning/control modules is one of the most important issues. Adaptation of already learned behaviors to newly encountered situations is a very useful property in applying reinforcement learning methods to real robots. A new behavior generation by combination of already learned behaviors is also important for the acquisition of a variety of behaviors in various situations. There are several works which deal with reuse of previous obtained policy.

Thrun [69] proposed "lifelong learning" which accelerates the learning by storing invariant knowledge for a environment in advance and reuses it as *a priori* knowledge in a new environment.

Tanaka and Yamamura [65] applied the similar idea to a simple navigation task on a grid world using a reinforcement learning method, in which the knowledge invariant in previous environments are extracted in advance and checked in a new environment if it is effective or not.

Minato and Asada [38] proposed a method which adapts robots to environmental changes by transferring a learned policy in the previous environments into a new one and modifying it to cope with these changes.

These methods concentrate on adjustment of certain behaviors to cope with new environments based on the knowledge obtained in the previous environments. However, the problem is that the state spaces of these methods are fixed and they cannot handle the situations in which the state space or the state valuables are varying,

On the other hand, there are a number of studies which lead to make efficient the learning process in a higher order state space by a new behavior generation by combination of already learned behaviors each of which has divided small state space (ex. [70][75]). However, the final state space consists of all state variables, each of which is divided finely, as a result, the number of state becomes enormous large.

2.4 Hierarchical Control Structure

There have been a variety of works on a multi-layered control architecture. Albus[3] proposed Real-time Control System, which consists of task decomposition, world modeling, and sensor processing modules at each layer of the hierarchy. The task is decomposed in advance into parallel and sequential subtasks, to be performed by cooperating sets of subordinate agents. However, he proposed only conceptual principle in his paper, and there is no concrete procedure or algorithm of task decomposition, cooperation between modules, nor development of the modules.

Singh [51, 52] has proposed a method of learning a hierarchy of models of the environment that abstract temporal detail as a means of improving the scalability of reinforcement learning algorithms. The concrete system shown in his paper has two levels of the hierarchy, the given task was decomposed into a few sub-tasks by hands, the lower module learns the behaviors of sub-tasks, and the higher learns the sequence of the sub-tasks for to accomplish the given tasks. The demonstration is shown in the simple grid world.

Kaelbling [31] proposed DG learning algorithm, which learns efficiently to dynamically changing goals and exhibits good knowledge transfer between goals. Then, she [30] proposed HDG algorithm which has a landmark network as an upper level of hierarchy and enables to learn the behavior more quickly. The demonstration is shown in the simple grid world, too.

Dayan and Hinton [17] have proposed Feudal Reinforcement Learning. The system has a number of levels at multiple resolutions in a grid-division state space; the higher level has coarser resolution while the lower has finer one. There are managers(modules) on each level who have sub-managers on the lower level. The manager sends a sub-goal set to one of the sub-managers, and give a reward if the sub-manager achieves it. The system needs to specify an explicit subgoal structure in order to achieve a hierarchical decomposition of the state space in advance. This work has

been applied in a rather simple grid maze world.

Digney [18, 19] has proposed Nested Q-learning algorithm that generates of hierarchical control structures in a learning system. The system has a number of Q-learning modules which has fixed, discrete states as a state space, and fixed primitive actions and the commands to the other modules as a action space. Task decomposition is done under two criteria; One is based on the received reinforcement signals, and the other is based on the frequency of visits to particular state space locations. One module can executes the behaviors of other modules as its own action, so this is regarded as the system has an abstraction mechanism of action. However, this system does not have an situation abstraction mechanism. This work has also been applied in a simple grid maze world.

Stone and Veloso [54] proposed layered learning, and applied it to a multi-agent learning system. At the lower-level, individual skills are learned, and more social skills are learned at the higher-level [55].

Morimoto and Doya [41] applied a hierarchical reinforcement learning method by which an appropriate sequence of subgoals for the task is learned in the upper level, and behaviors to achieve the subgoals are acquired in the lower level. However, the problem is that the human designers have to define the subtasks, landmarks, skills, and subgoals based on their own experiences and insights.

Fujii et al. [24] proposed a multilayered reinforcement learning scheme for collision avoidance. This method allows the system to reduce the computational memory by decomposing whole state space into several subspaces, and to merge them at the higher level. However, selection of the sensor information for each learning module is constructed by human designer based on his/her own experiences and insights.

Hasegawa and Fukuda [27, 28] proposed a hierarchical behavior controller, which consists of three types of modules, behavior coordinator, behavior controller and feedback controller, and applied it to a brachiation robot. However, the whole structure of this hierarchical controller is designed by human, and the system only adjusts some parameters of a certain behavior coordinators and behavior controllers.

Tani and Nolfi [67, 66] developed an on-line multi-layered sensory flow pattern learning scheme. A set of recurrent neural net (RNN) modules is self-organized as a set of experts to account for different categories of sensory flow which the robot experiences. Meanwhile, another set of RNNs at the higher level learns the sequences of module switching observed at the lower level. Their scheme, however, does not have any control learning structure, which makes it difficult to acquire a purposive behavior by itself.

Summary

One of the most important issues on the multi-layered learning system is self-construction of the hierarchy, in addition to the self-generation of subgoals. In the all of the above methods, human designers have already prepared hierarchical structures in advance based on their own experiences and insights.

The state and action space construction is also one of the important issues in the multi-layered learning system. To the best of our knowledge, there is no study on the construction of state and action spaces on the multi-layered modules architecture.

Chapter 3

State Space Construction by Incremental Sensor Space Segmentation for Vision based Behavior Acquisition

3.1 Introduction

In this chapter, we propose a method by which a robot learns a purposive behavior within less learning time by incrementally segmenting the sensor space based on the experiences of the robot. Here, we do not deal with the sensor selection problem. [1] The incremental segmentation is performed by constructing local models in the state space, which is based on the function approximation of the sensor outputs to reduce the learning time and the reinforcement signal to emerge a purposive behavior. The method is applied to a soccer robot which tries to shoot a ball into a goal. The experiments with computer simulations and a real robot are shown. As a result, our real robot has learned a shooting behavior within less than one hour training by incrementally segmenting the state space.

[1] We have already tried to apply principal component analysis to acquire feature vector representing the environment. We show the details in the end of this chapter as an appendix.

3.2 Basics of Reinforcement Learning

Before getting into the our method, we briefly review the basics of the reinforcement learning.

We assume that the robot can discriminate the set $\boldsymbol{S}$ of distinct world states, and can take an action from the action set $\boldsymbol{A}$. The world is modeled as a Markov process, making stochastic transitions based on its current state and the action taken by the robot. For each state-action pair (s, a), the reward $r(s, a)$ is defined.

The general reinforcement learning problem is typically stated as finding a policy that maximizes discounted sum of the reward received over time. Watkins' Q-learning algorithm [14] gives us elegant method for doing this.

In the Q-learning algorithm, the robot takes an action $a \in \boldsymbol{A}$ in a state $s \in \boldsymbol{S}$ and transits to the next state $s' \in \boldsymbol{S}$, then it updates the action-value function $Q(s, a)$ as follows.

$$Q(s, a) \Leftarrow (1 - \alpha)Q(s, a) + \alpha(r(s, a) + \gamma \max_{a' \in \boldsymbol{A}} Q(s', a')) \tag{3.1}$$

where α is a learning rate and γ is a discounting factor.

After a sufficient number of trials, the optimal decision policy is obtained as taking the action a which maximizes the $Q(s, a)$ value at the state s.

3.3 Self-Segmentation of Continuous State Space

We focus on the state space segmentation, and reduction of learning time, now. As a basic idea coping with these problems, we adopt the incremental segmentation of the state space by which the state space is autonomously segmented, and we expect the reduction of the learning time and the capability of coping with dynamic change of the environment.

A key issue is to find the basic policy to segment the state space so as to realize the desirable features described above. The following two principles can be considered.

A: Segment the state if the prediction of sensor outputs is incorrect.

B: Segment the state if the same action causes the desirable or undesirable result (ex., transition to the goal states or non-goal states) even though the prediction itself is correct.

According to the first, the robot can discriminate the world situations with as few states as possible based on the experiences until the current time. This contributes to the followings:

1. as long as the prediction of sensor outputs is correct, tedious exploration process can be eliminated, and therefore

2. reinforcement learning converges immediately. Further

3. the robot can cope with dynamic change of the environment due to its incremental property of the segmentation process.

However, **A** does not care where the goal state is. On the other hand, **B** contributes to the emergence of the purposive behavior. Even though the prediction is correct, it would be nonsense if the same action from the same state resulted in different situations. This state should be separated so that the same action can always cause the desirable transition.

From the above arguments, **A** is related to the world model construction by coarse mapping between states and actions far from the good states. While, **B** is related to the the goal oriented segmentation based on the reinforcement signals. As a result fine mapping between states and actions near the goal states is obtained.

3.4 Algorithm

Figure 3.1 shows the rough flow of the proposed method. First, the robot acquires sensor outputs as data. If the data are consistent with the current local models, the robot updates the local models. Else, the robot builds new local models, and initialize the action value function by reusing the knowledge obtained by the past experiences. Then, it learns the policy using reinforcement learning, and returns the beginning. The robot iterates this cycle forever.

3.5 Action Space and Data Structure

In the conventional reinforcement learning methods, an "action" is defined as an execution of motor command per fixed sampling interval. In a real situation, this definition often causes "state-action deviation problem" as pointed out by Asada et al. [7]. They defined such an action as an action primitive, and a action is defined as a sequence of action primitives until the current state changes. Here, we follow their definition.

We define a data set $d_i \in D$, $(i = 1, 2, \cdots)$ as a triplet of action primitive $m_i \in M$, sensor value $\boldsymbol{s}_i \in S$ and its gradient $\dot{\boldsymbol{s}}_i \in \dot{S}$.

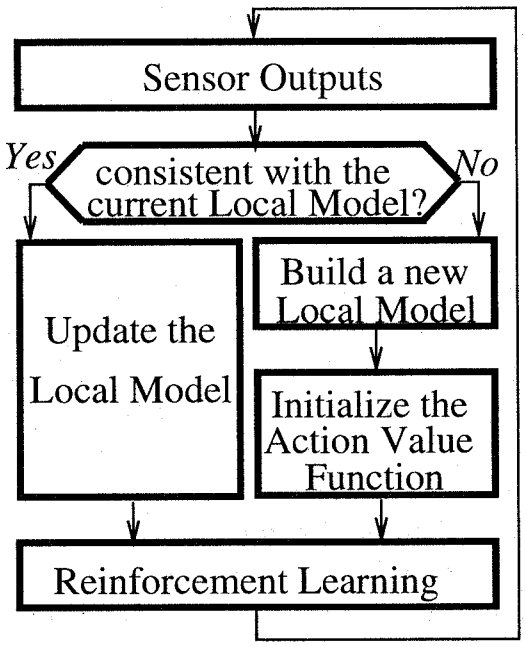


Figure 3.1: The rough flow of the proposed method

If the robot stores the all data of its experiences, the amount of data will exceed the capacity of the robot. Therefore, it is not practical to store the all data. Further, the robot often receives incorrect data because of sensor noise, change of the environment, and the uncertainty of motor commands. Then, we update the gradient of inputs vector $\dot{\boldsymbol{s}}_i$, when the robot receives a new data set d_j.
if

$$|\boldsymbol{s}_i - \boldsymbol{s}_j| < \epsilon \quad and \quad m_i = m_j$$

then

$$\dot{\boldsymbol{s}}_i = (1-\beta)\dot{\boldsymbol{s}}_j + \beta\dot{\boldsymbol{s}}_i$$

else

register $\boldsymbol{s}_j$ as a new datum.

Here, $0 < \beta < 1$ and ϵ stands for a similarity threshold. $|\cdot|$ means weighted Euclidean norm.

3.6 Local Model Construction

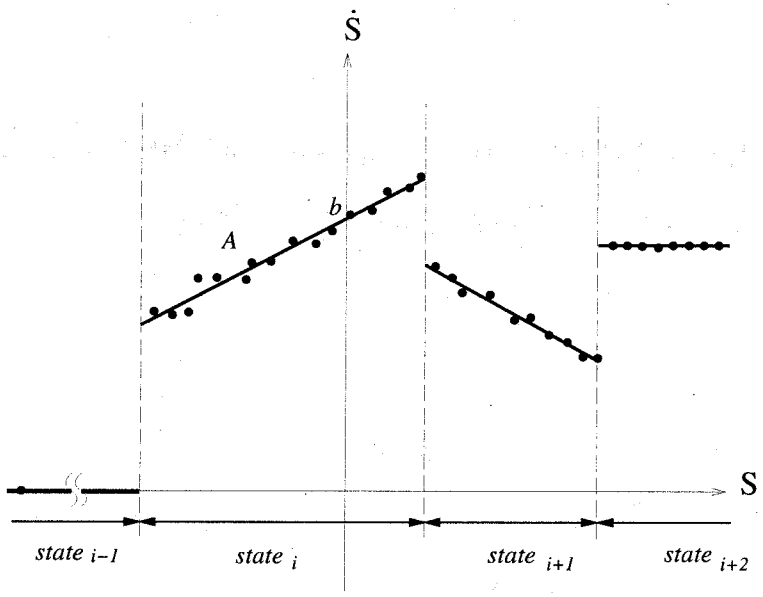


Figure 3.2: The construction of local model and the segmentation of sensor space

We first explain the method of local model construction by using a linear model of the the gradient of sensor values, that is,

$$\dot{\boldsymbol{s}} = \boldsymbol{A}\boldsymbol{s} + \boldsymbol{b}.$$

The algorithm for local model construction and segmentation is as follows:

1. Gather data sets which have the same action primitive.
2. Apply the weighted linear regression method to fit a linear model to the data sets.
3. Divide the data into two with a method of cluster analysis using weighted Euclidean norm as similarity and return 2 if the unbiased variance of the residual exceeds a certain threshold, else stop.

Figure 3.2 shows an example of the construction of local model and the segmentation of sensor space in case of one dimension of the sensor value.

The segmented regions obtained by the above process are regarded as "states" for the reinforcement learning method. Each segmented region has a number of data sets d_i, and let the $\boldsymbol{s}_i$ $(i = 1, 2, \cdots)$ be the representatives of the region. A new sensor values $\boldsymbol{s}_q$ is classified into one of the states by finding a representative in the corresponding state based on NN(nearest neighbor) methods.

3.7 Composite of the Segmented Sensor Spaces by A Number of Action Primitives

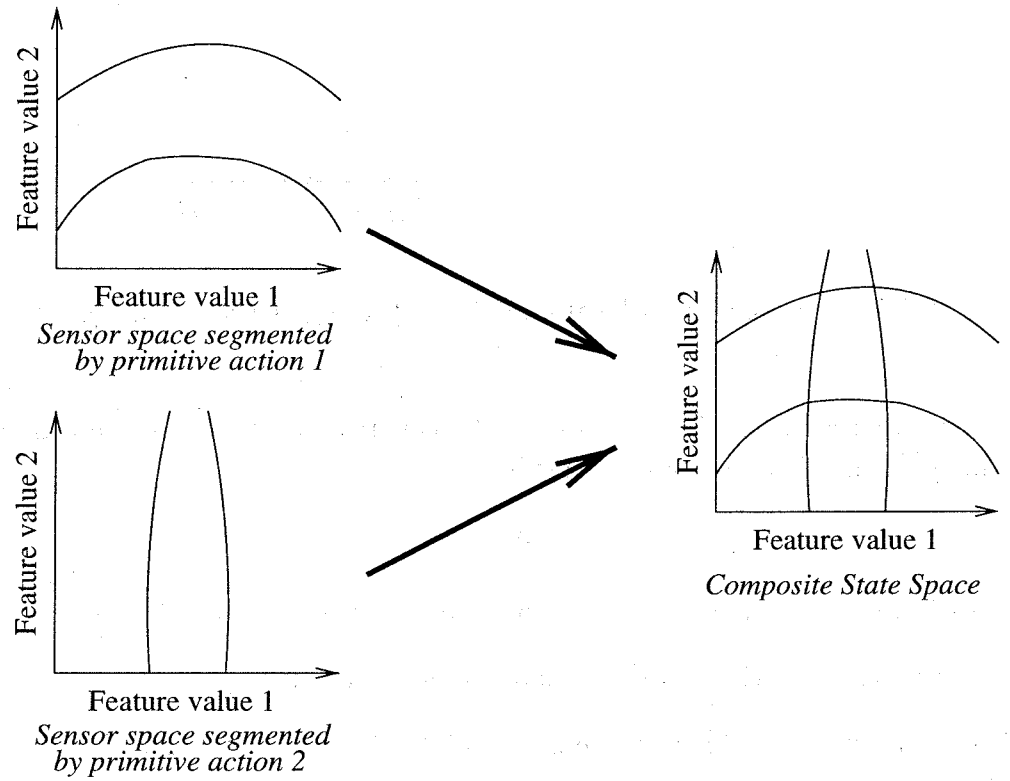


Figure 3.3: Example of composite State Space based on segmented sensor spaces by two action primitives

The learning states are defined as the segmented regions as shown in previous subsection. However, each action primitive may result in a different segmentation of

the feature space in general. The system segments a sensor space as shown in previous subsection for each action, then it constructs new state space as direct product of all segmented spaces.

Figure 3.3 shows an example. There are two action primitives, and the system segments the sensor space (see the left side of the figure). Then, it constructs a new state space (see the right side of the figure).

3.8 Sensor Space Segmentation Based on Reward Distribution

Near the goal state, the segmented region obtained by the above process is not always appropriate because multiple transitions (success in the reaching the goal state or failure) from a same pair of the sensor values in the same region and the action primitive can be often observed. Then, we use the reinforcement signals to divide the segmented region so that the same action primitive from the divided region can reach the unique state (the goal state or others).

There are a number of methods which segment sensor space based reward distribution [32, 23, 77]. The goal state can be specified the easily if the reward is given correctly when the situation comes into the goal state. Since acquiring th reward is the most important for the robot, accurate recognition of situations around the goal state might enable the robot to behave accurately.

Then, the system records the sensory-motor sequence before it receives the reward, and it segments the feature space based on the reward distribution. We define a data set $d_i^r \in D$, $(i = 1, 2, \cdots)$ as a triplet of action primitive $m_i \in M$, sensor value $\boldsymbol{s}_i \in S$ and the given reward $r_i \in R$. The detail algorithm of storing, updating data and local model construction is almost same as described in the former sections.

3.9 Composite of the Segmented Sensor Spaces

3.6 shows the sensor space segmentation based on local liner model, and **3.8** shows the segmentation based on reward distribution. If the system adopts these two approaches simultaneously, the sensor space would be divided too much.

Therefore, we set a priority; reward based segmentation is first and model based segmentation is second. This means that if the space is already divided by the reward based segmentation, there is no more segmentation by model based approach. The reason is that reward based segmentation is likely more appropriate than model

based one because the reward is the most essential element for purposive behavior acquisition.

3.10 Action Generation

As we stated in the section 3.5, we define "action" as "a sequence of several action primitives until the current state changes". We prepare two kinds of action. One is the a sequence of the same action primitive as one action until the current state changes. The other is a sequence of action primitives which is generated with the local models as follows.

One can calculate the desired gradient of sensor values $\dot{\boldsymbol{s}}_d$ from the current sensor values $\boldsymbol{s}_j$ and desired sensor values $\boldsymbol{s}_d$, that is,

$$\dot{\boldsymbol{s}}_d = \boldsymbol{s}_d - \boldsymbol{s}_j.$$

Since the linear model parameters have been obtained in each local model, we can predict a desirable action to satisfy the above equation. The robot carries out the action primitive m_d which is closest to the desired gradient of sensor values.

$$m_d = arg \min_{m_i} (\dot{\boldsymbol{s}}_d - \dot{\boldsymbol{s}}_{m_i})^2 \tag{3.2}$$

3.11 Reuse of the Knowledge Obtained by Experiences

Theoretically, the action value function should be reset every time the new state space is constructed during the incremental segmentation process of the state space. This prevents the knowledge obtained by the past experiences from being used efficiently in the learning process. Then, we consider to reuse the knowledge by calculating the new action value function for the new segmented state space from the old state space and its action-value function.

Basic idea is to adopt a new action value function calculated by weighted sum of the old action value function as the initial knowledge for reinforcement learning. The weights are calculated based on the numbers of the sensor value representatives in both the new and old states. Concrete procedure is given as following.

S^{old} and S^{new} denote the old and new state spaces, respectively. $\boldsymbol{s}_k (k = 1, 2, \cdots, n)$, $state_j{}^{old} (j = 1, 2, \cdots, n^{old})$ and $state_i{}^{new} (i = 1, 2, \cdots, n^{new})$ denote the sensor value of stored data d_i, a state of the old state space and a state of the new state space. We prepare a $n^{old} \times n^{new}$ matrix $T(state^{old}, state^{new})$ of which component $t(state_i^{old}, state_j^{new})$

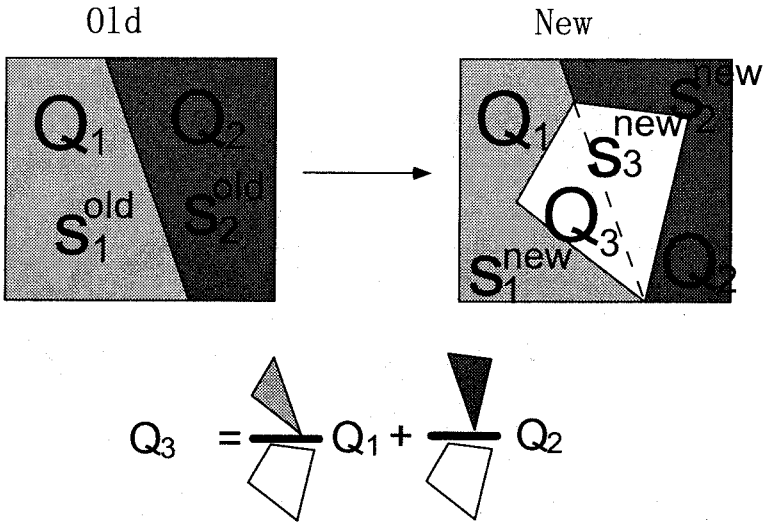


Figure 3.4: Recalculation of Q value

represents the number of sensor value representatives s_k that are classified into $state_j^{new}$ from $state_i^{old}$. Then, we can calculate the action-value function of the new state space $Q(state_i^{new}, a)$ as follows.

$$Q(state_i^{new}, a) = \sum_{j=1}^{n^{old}} \omega_{ji} Q(state_j^{old}, a), \tag{3.3}$$

where

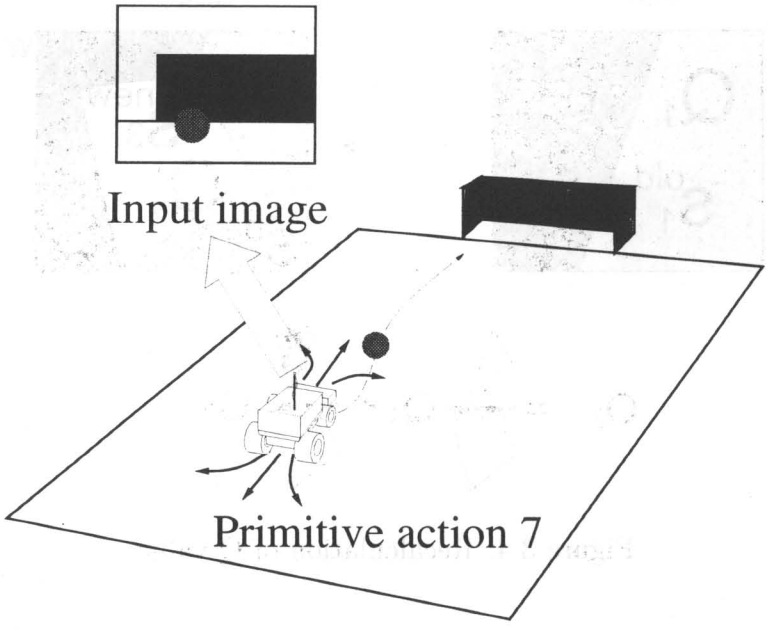
$$\omega_{ji} = \frac{t(state_j^{old}, state_i^{new})}{\sum_{l=0}^{n^{old}} t(state_l^{old}, state_i^{new})}. \tag{3.4}$$

3.12 Experiments

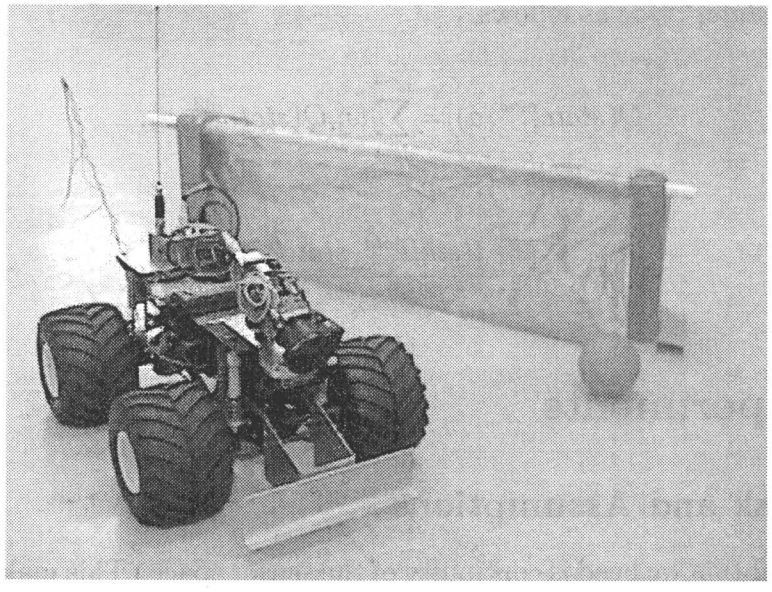
3.12.1 Task and Assumptions

Only one assumption we need is continuity of the sensor space. This makes local model construction efficient, and therefore contributes to eliminate unnecessary exploration.

We apply the method to shooting behavior acquisition by a soccer robot as an example of robot tasks. The task for a mobile robot is to shoot a ball into a goal as shown in Figure 3.5(a).



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



(b) A picture of the radio-controlled vehicle with a ball and a goal

Figure 3.5: A task and our real robot

We assume that the environment consists of a ball and a goal, that the mobile robot has a single TV camera and can get the primitive features of the ball and the goal, and that the robot does not know the location and the size of the goal, the size and the weight of the ball, any camera parameters such as focal length and tilt angle, or kinematics/dynamics of itself. Figure 3.5(b) shows a picture of the real robot, the ball and the goal used in the experiments.

The sensor information the robot discriminates consists of five features of the ball and the goal. The features are the size and the position of the ball on the image and the first, second, and third principal component of the goal region on the image which is described in the section 3.14. [2] The robot often loses the ball and/or goal because of its narrow angle of view (65°). In such a case, there are no feature values of the ball and/or the goal. However since the robot knows into which direction it lost the ball and/or the goal by memorizing the previous state, large absolute constants are assigned to these lost states. As a result, the local models for these states are obtained with their gradients equal zeros (The left side of Figure 3.2 indicates such a case).

3.12.2 Simulation

The number of action primitives which the robot can take is seven as shown in Figure 3.5(a). We assign a reward value 1 when the ball was kicked into the goal or -0.1 otherwise. 90% of the time the robot selects the action specified by its optimal policy, the remaining 10% of the time it takes a random action.

Figure 3.6 shows the success rate and the number of states during the incremental state space segmentation and the processes of shooting behavior acquisition. Here, the success rate indicates the number of successes in the last twenty trials.

Figure 3.7 shows a projection of the state space after 1,110 trials, where the state space in terms of the ball size and the goal size is indicated when the position of the ball and the goal are center of the screen and the orientation of the goal is frontal. The upper left region pained black means a situation in which the ball size is small, the goal size is large, and the positions of ball and goal are center of the image. Since such a situation is impossible in this experimental situation, the system extrapolates the situation from the region which is near to it.

After the learning, the number of states is 28, the number of data is 141 (for left-forward action, 17, for forward action, 32, for right forward action, 15, for left-backward action, 13, for backward action, 15, for right backward action, 13, and for

[2] These three principal components represent the size, the x position, and the inclination of the goal region on the image.

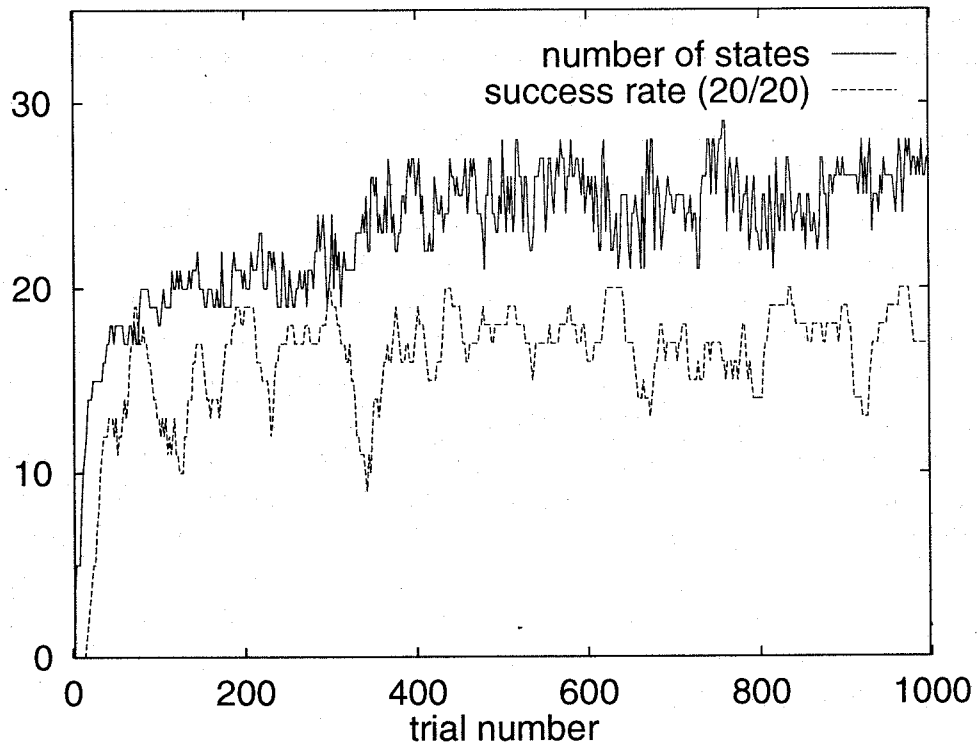


Figure 3.6: The success rate and the number of states

stop action, 36, for goal state, 94).

Figure 3.8 shows sequences of the learned behavior. The robot moves around and goes back to find the ball and the goal when they are out of the view as shown in Figure 3.8(a), then it goes forward and pushes the ball into the goal when it capture the both as shown in Figure 3.8(b).

Figure 3.9 shows the success rate and the number of states in the case that the ball diameter suddenly became twice at the 500th trial. It suggests the proposed method can deal with dynamic change of the environment.

Figure 3.10 shows the success rate while the system took the rate of optimal policy of 90%, 70%, and 50% during the learning time. Figure 3.11 shows the number of states in the situations. Figure 3.10 shows that the success rate is high when the agent took high rate of optimal policy, and Figure 3.11 shows that the system with exploration behavior divided the sensor space more than the system with exploitation (optimal behavior).

Figures 3.12 and 3.13 show the success rate and number of states in terms of

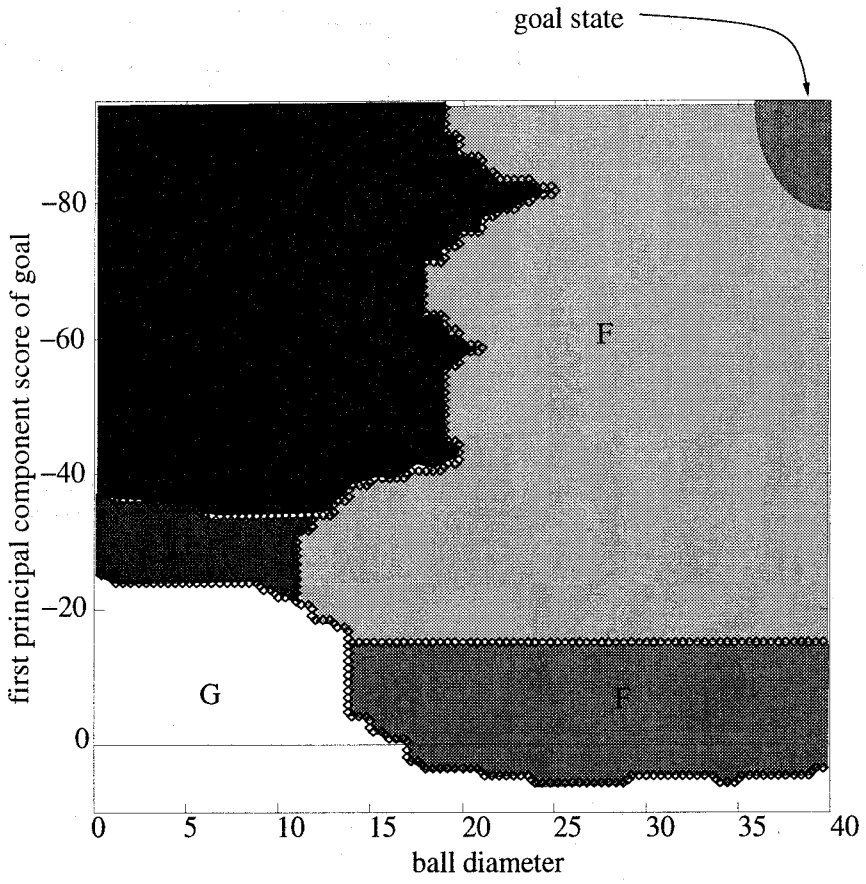
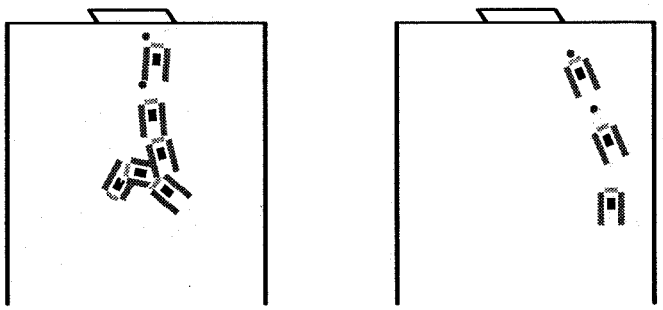


Figure 3.7: Result of state space construction

the fitting thresholds, respectively. We can see that the lower threshold leads better performance in general. The reason is that the system with the lower threshold tends to segment with more fine resolution than the system with the higher threshold. However, the lower threshold prevents the success rate drastically sometimes (see the curve at the 110 th step and the 340 th step). The reason might be that the fine incremental segmentation of the state space prevents the system from the convergence of state-action value (Q) function because the increase of the number of the state requires the system to update Q function more frequently.

Figure 3.14 shows the sequence of the state space during the learning. The horizontal and vertical axes indicate ball size and the value of the first principal component describeed in **3.14**, while the position of the ball and the goal are center of the screen



(a) Finding (b) Dribbling

Figure 3.8: Some kinds of behaviors during learning process

and the orientation of the goal is frontal. The upper left region on the figure means a situation in which the ball size is small, the goal size is large, and the positions of ball and goal are center of the image, and such a situation is impossible in this experimental situation.

The characters such as F, B, LF ,and G indicate forward action, backward action, left forward action, and the goal oriented action described in **3.10**.

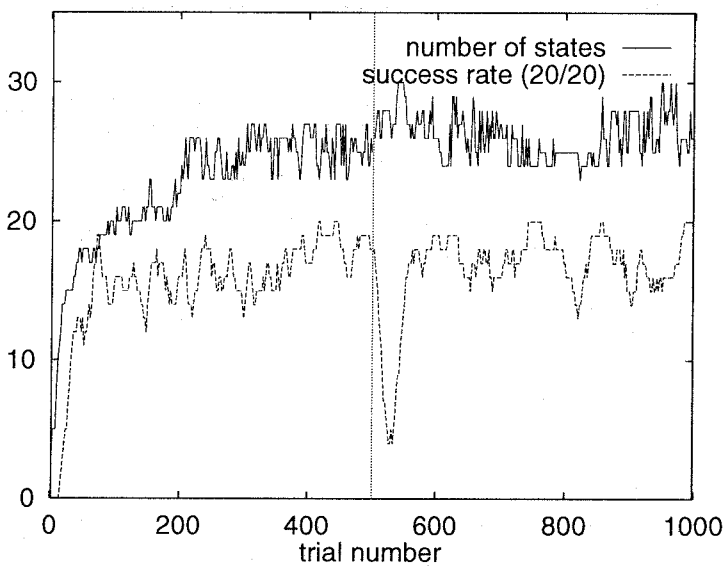


Figure 3.9: Success rate and the number of states in the case that environment change one the way

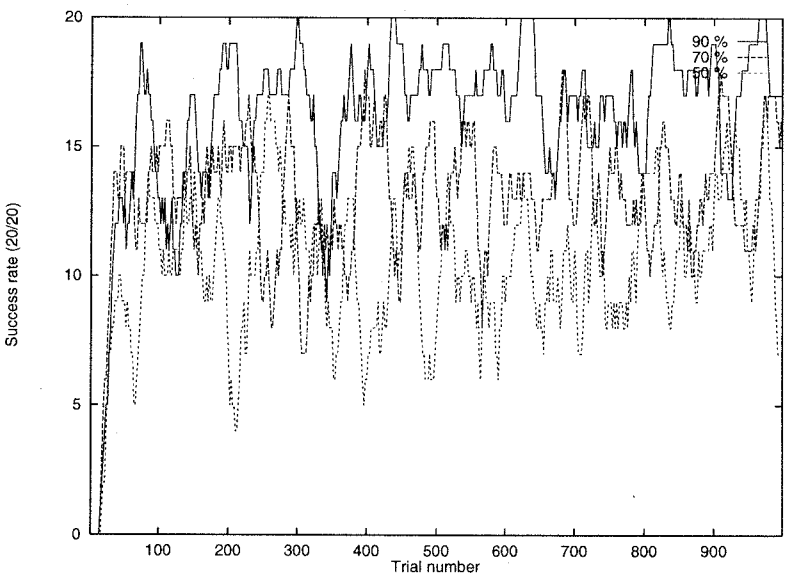


Figure 3.10: Success rate with different rates of optimal action

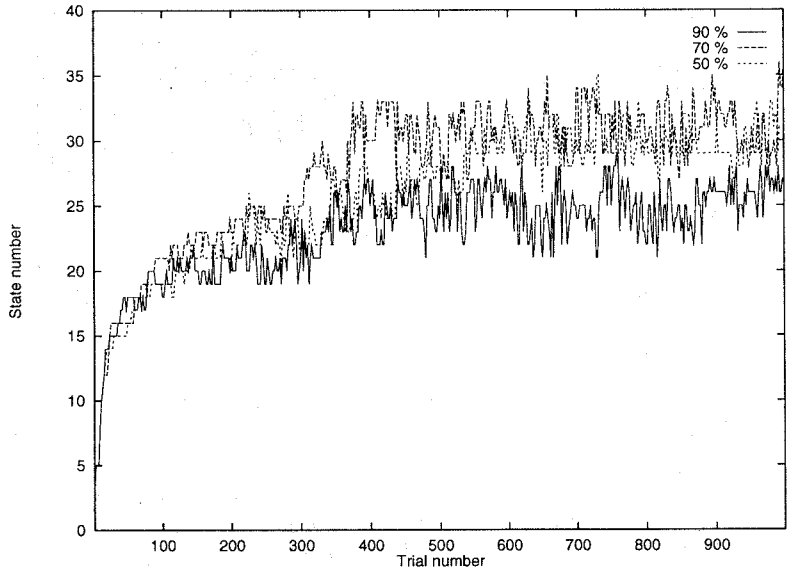


Figure 3.11: Number of states with different rate of optimal action

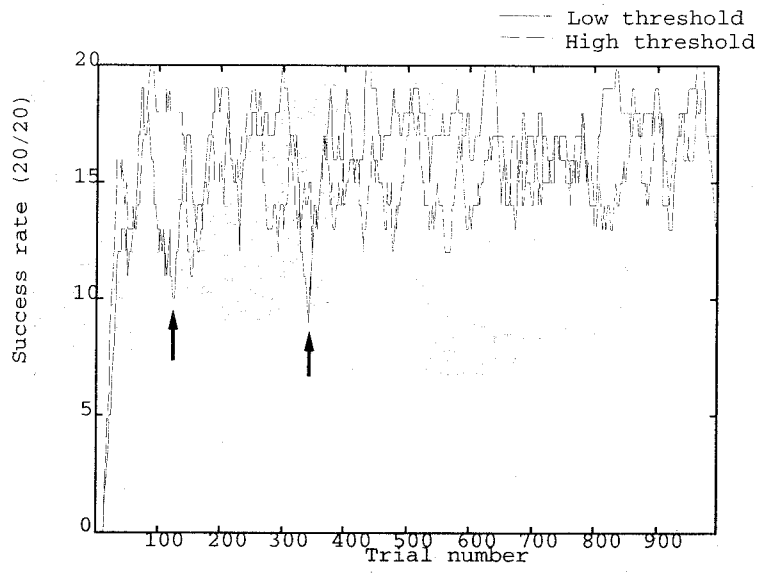


Figure 3.12: Success rate with two kinds of fitting thresholds

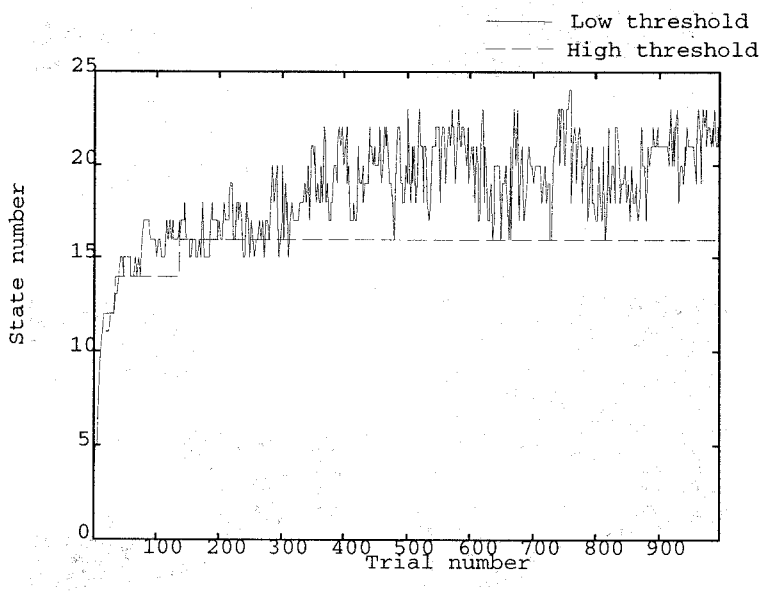
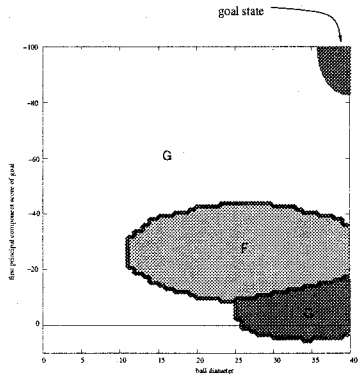
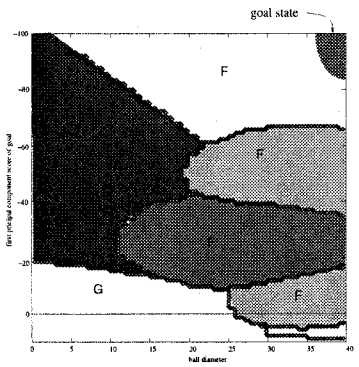


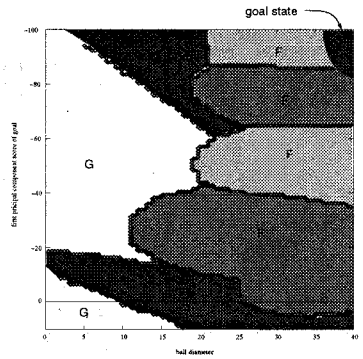
Figure 3.13: Number of states with two kinds fitting thresholds



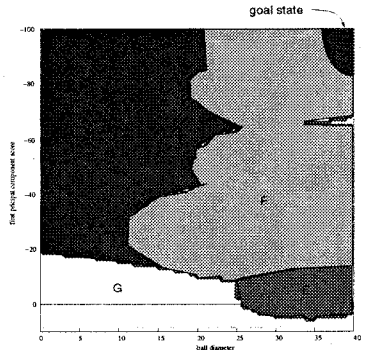
(a) 20 Trial



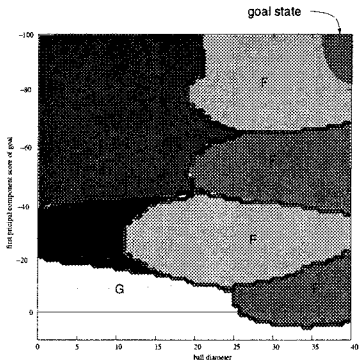
(b) 50 Trial



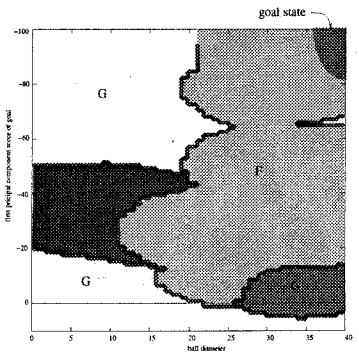
(c) 100 Trial



(d) 300Trial



(e) 600 Trial



(f) 1000 Trial

Figure 3.14: Change of the state space segmentation

3.12.3 Experiment on the Real Robot

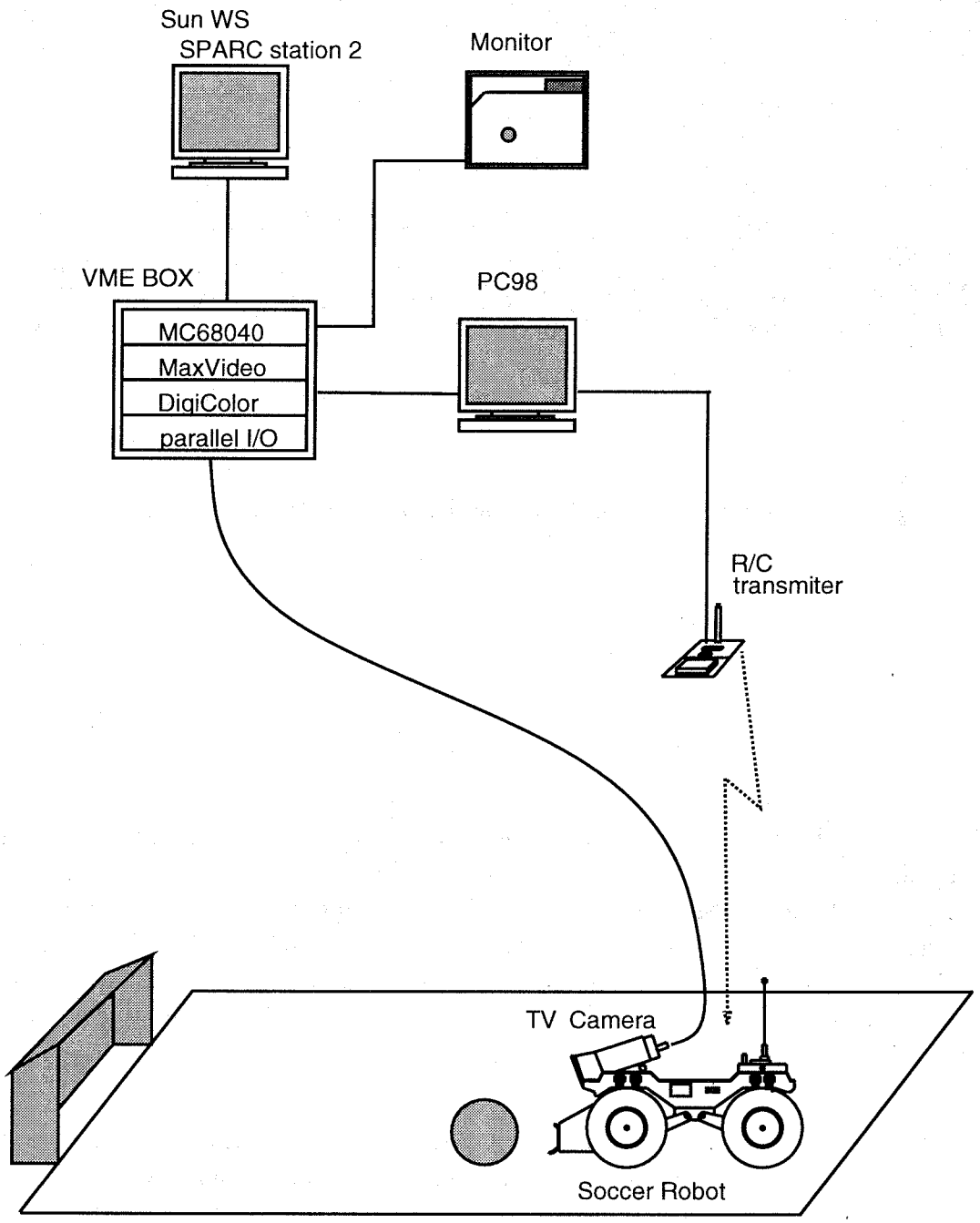
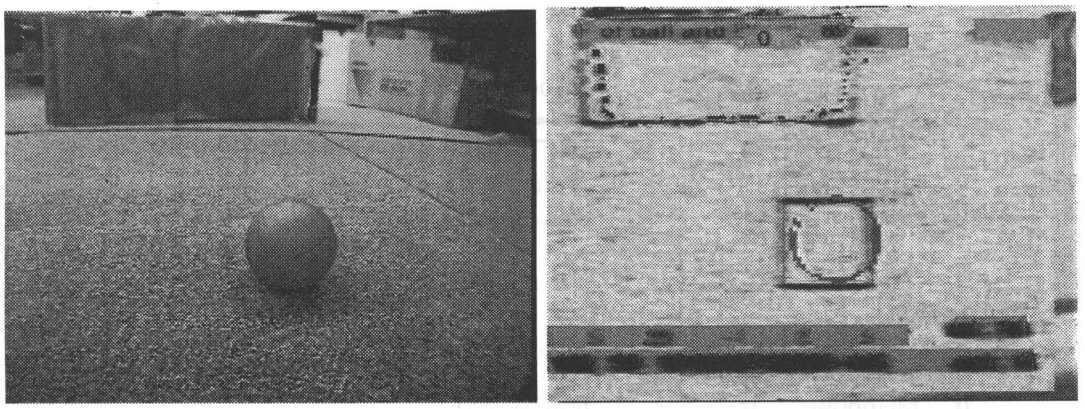


Figure 3.15: A configuration of the real robot



(a) input image (b) detected image

Figure 3.16: Detection of the ball and the goal

Figure 3.15 shows a configuration of the real mobile robot system. Figures 3.16(a) and (b) show an image taken by a TV camera mounted on the robot and an image processed by Datacube MaxVideo 200, a real-time pipeline video image processor. The image processing and the vehicle control system are operated by VxWorks OS on MC68040 CPU which are connected with host Sun workstations via Ether net. The result of image processing are sent to the host CPU to decide an optimal action against the current state. The sampling time is about 30ms.

Figure 3.17 shows an example of the linear model fitting for the situation in which the robot takes a forward action primitive when watching a ball. When the ball is located far from the robot and the robot takes a forward motion, then the ball diameter on the image increases. On the other hand, when the ball is located near from the robot, the ball diameter doesn't change any more while the robot takes a forward motion, because the ball is collided with the robot.

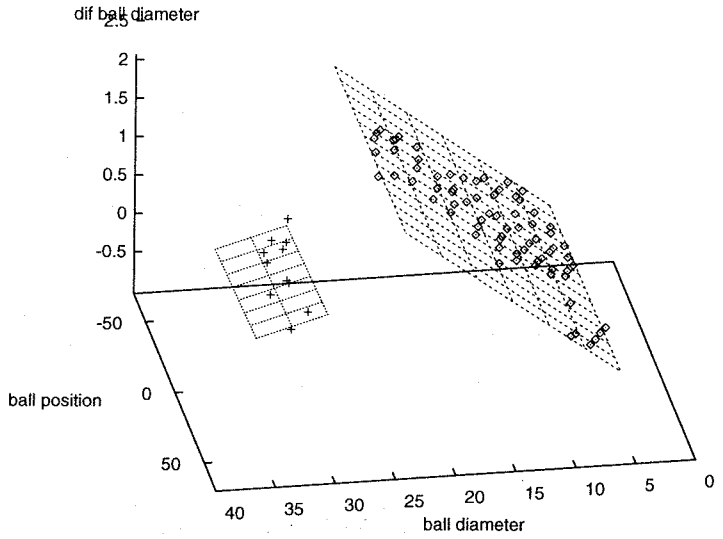


Figure 3.17: An example of linear model fitting : the data is obtained while the robot gets a forward action primitive.

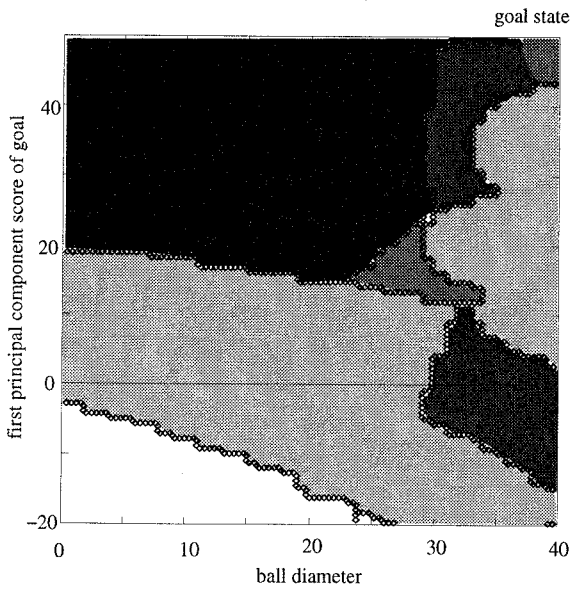


Figure 3.18: state space construction of real robot experiment

Figure 3.18 shows the state space after 72 trials. The state space in terms of the ball and the goal size is indicated when the position of the ball and the goal are center of the screen and the orientation of the goal is frontal. The numbers of acquired states and data are 18 and 151, respectively.

Figure 3.19 shows how the robot tries to shoot a ball into the goal. Because of the sensor noise and the uncertainty of the motor commands, the robot often misunderstands the states, and takes wrong actions, therefore it fails to do the task. ① indicates that the robot is going to shoot a ball into the goal and move forward. But it fails to kick the ball at ② because the speed is too hight to turn. The ball is occluded by the robot in ②. Then, it goes left back so that it can shoot a ball at ③. But it fails again at ④. Then, it goes left back again at ⑤. After all, the robot does the shooting task successfully at ⑥.

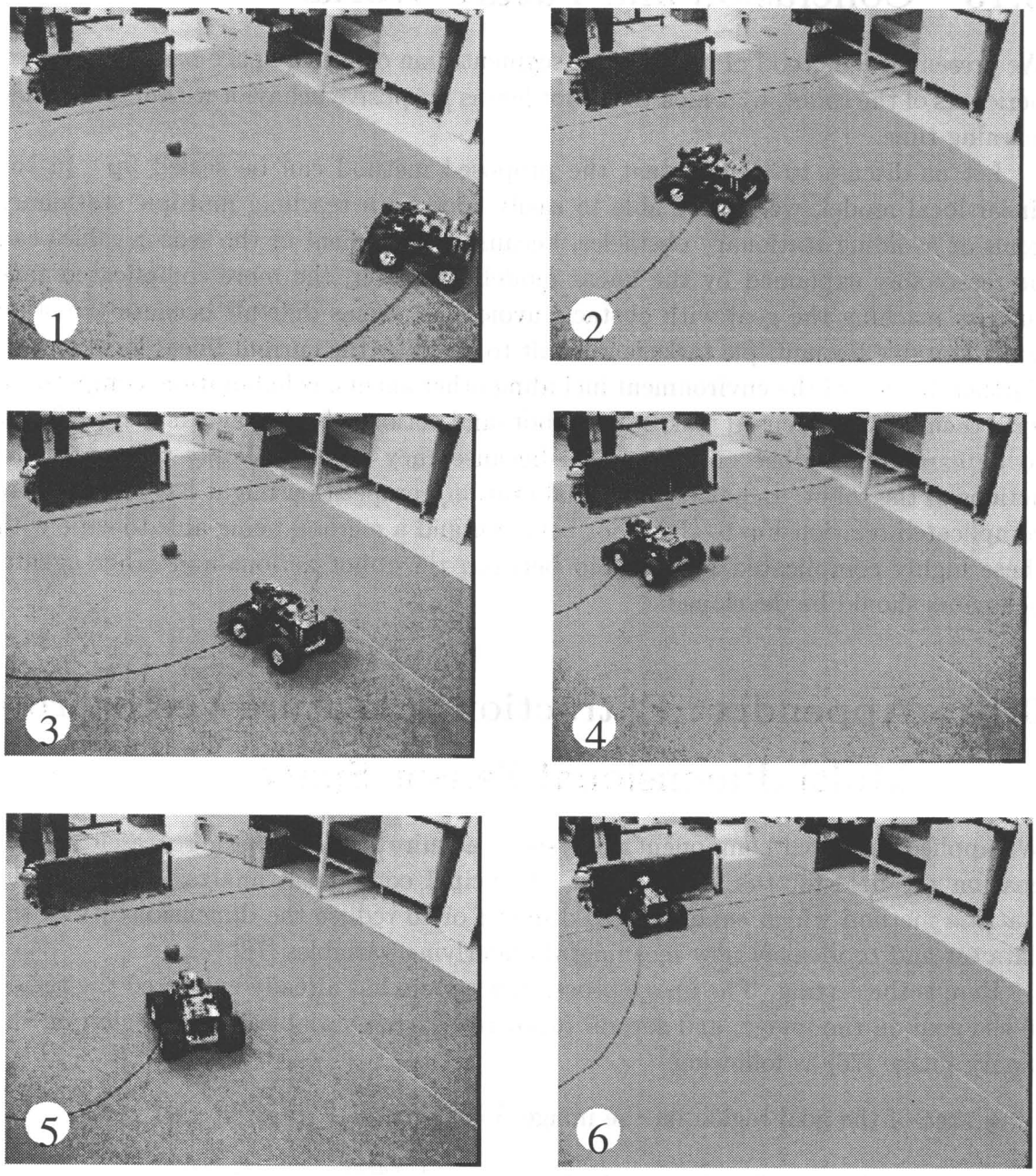


Figure 3.19: The robot succeeded in shooting a ball into the goal

3.13 Conclusion and Future Works

We presented a method of incremental segmentation of sensor space based on the experiences of the robot, by which the robot learns purposive behavior within reasonable learning time.

Let us discuss to what extent the proposed method can be scaled up. In the linear local model, we will be able to easily cope with reaching multiple stationary goals or avoiding stationary obstacles, because the gradient of the sensor values can be reasonably explained by the linear model. However, the more complicated task such as reaching the goal with obstacle avoidance, seems difficult because the state space suitable for multiple tasks is difficult to build by the current linear local model. Further, in case of the environment including other agents, collaboration/competition with them are the focused task to the robot, and actions of other agents seem difficult to be explained by the current model because they are not simply related to the actions of the robot. Behaviors for collaboration/competition might have much more complicated relationship to the robot behavior and a method being able to cope with these highly complicated relationship between the robot actions and other agents' behaviors should be developed.

3.14 Appendix : Extraction of Feature Vector from Multi-dimensional Sensor Space

We applied principal component analysis to acquire feature vector representing the goal on the field for the soccer robot. Principal component analysis is one of the statistic method which enables us to discover or to reduce the dimensionality of the data set and to identify new meaningful underlying variables [78].

Here is the setting. The image processing system has already extracted the region of the goal on the image, and several primitive feature variables of the region on the binary image [76] as following:

- area of the goal region on the image $\overline{S} = \int_S dx\, dy$
- center of the goal region on the image $\overline{x} = \frac{\int_S x\, dx\, dy}{\overline{S}}$, $\overline{y} = \frac{\int_S y\, dx\, dy}{\overline{S}}$
- moments of the goal region on the image
  $\int_S (x-\overline{x})^2 dx\, dy$, $\int_S (y-\overline{y})^2 dx\, dy$, $\int_S (x-\overline{x})(y-\overline{y}) dx\, dy$

We have sampled the data when we put the robot randomly on the field and the robot capture the any parts of the goal on the image. The number of data is 4,350.

Table 3.1 shows that the first principal component has large weights of area, the y axis of the center, the second moments of x, y axes of the goal region on the image, then this first principal component represents the size of the goal region of the image. The second component and third one have large weights of x axis of center and covariance moment around the center of the goal region of the image. The covariance moment around the center of the goal region is correspond to the inclination of the goal view. Therefore, the second component and third component vectors represent the x position and the inclination of the goal region on the image. Figure 3.20 shows the contributing rates.

Table 3.1: Weights of Goal Vectors by Principal Component Analysis

Principal Component	1	2	3	4	5	6
	Contribution Ratio (%)					
	52.0	20.0	14.0	10.0	2.9	0.9
variable	weight					
area of the region	0.540749	0.116863	0.024735	-0.227210	-0.218212	0.770766
x-axis of the center	0.088836	-0.617226	-0.770923	-0.128451	-0.001237	0.017783
y-axis of the center	0.390129	-0.275403	0.126652	0.826337	-0.262000	-0.066596
second moment around y-axis	0.517237	0.151763	-0.012555	-0.377643	-0.429306	-0.618351
second moment around x-axis	0.528275	0.078885	-0.012108	0.032007	0.833562	-0.136771
covariance moment around the center	0.029817	-0.707300	0.623474	-0.324660	0.067894	-0.010170

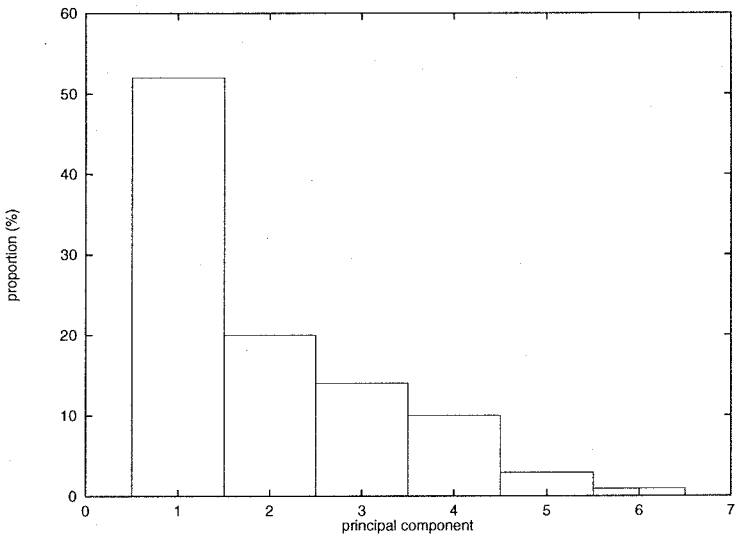


Figure 3.20: Contributing rate of principal component

Chapter 4

Behavior Acquisition by Multi-Layered Reinforcement Learning

4.1 Introduction

In order to realize an autonomous robot which acquires various behaviors by itself in real world, it is necessary to be able to manage a wide range of state and action variables according to situations, to keep the spaces as small as possible, and to learn/control behaviors based on the small state and action spaces.

It is almost impossible or impractical that robot acquires the various behaviors for the given tasks based on a huge monolithic state/ action space which consists of all sensors' information and actuators commands, because the computational resources are limited and learning time is not eternity from a practical viewpoint.

It is almost impossible or impractical that robot acquires the various behaviors for the given tasks based on a huge monolithic state/ action space which consists of all sensors' information and actuators commands, because the computational resources are limited and learning time is not eternity from a practical viewpoint.

Another approach to the problem of the curse of dimension and the perceptual aliasing is to adopt a hierarchical structure within leaning control system. That is, the system

1. prepares learning/control modules of one kind each of which deals with a subspace divided from a whole state/action space,

2. abstracts situations and behaviors based on the acquired learning/control modules, and

3. acquires higher level, new behaviors based on the state and action spaces constructed from already abstracted situations and behaviors.

This approach can suppress the explosion of the state and action spaces since the higher level learning/control system manages adequately small size spaces which are abstracted in the lower levels.

In this chapter, we propose a method by which a hierarchical structure for behavior learning is self-organized. The modules in the lower networks are organized as experts to move into different categories of sensor value regions and learn lower level behaviors using motor commands. In the meantime, the modules in the higher networks are organized as experts which learn higher level behavior using lower modules. Each module assigns its own goal state by itself. We apply the method to a simple soccer situation in the context of RoboCup, and show the experimental results.

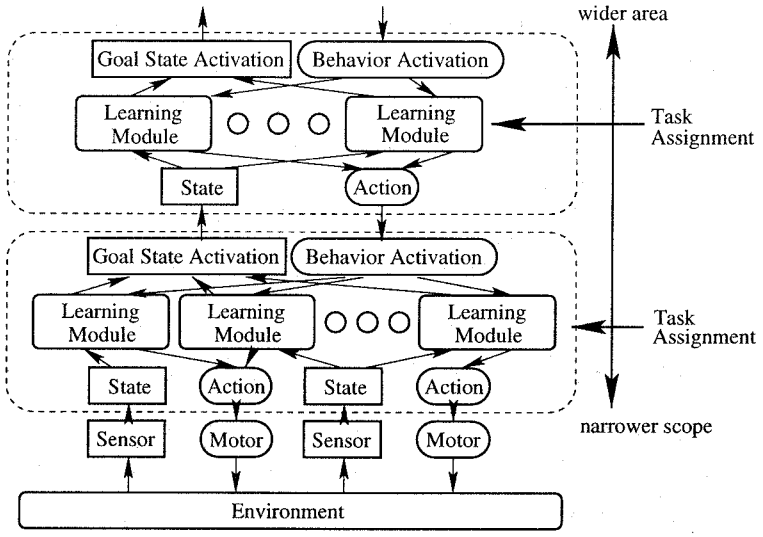
4.2 Multi-Layered Learning System

The robot prepares learning modules of a kind, makes a layer with the modules, and constructs a hierarchy with the layers. The hierarchy of the learning modules' layers seems to play a role of task decomposition. The lower learning modules explore small areas in the given environment, and learn lower level, fundamental behaviors. They learn behaviors with narrower scope and shorter time horizons, focusing on the more details. In contrast, the upper learning modules explore a large area, and learn higher level, more abstracted behaviors based on the learning modules at the lower layer. They have behaviors with broader scope, longer time horizons, and less concern for the details.

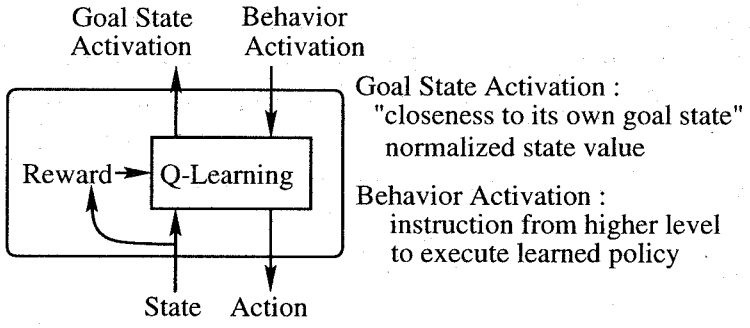
4.2.1 Architecture

The the proposed architecture of the multi-layered reinforcement learning system is shown in Figure 4.1, in which (a) and (b) indicate a hierarchical architecture with two levels, and individual learning module embedded in the layers.

Each module has its own goal state in its state space, and it learns the behavior to reach the goal, or maximize the sum of the discounted reward received over time, using the continuous Q-learning method [62]. The state and the action are constructed using sensory information and motor command, respectively at the bottom level.



(a) A whole system



(b) A behavior learning module

Figure 4.1: A hierarchical learning architecture

The input and output to/from the higher level are the goal state activation and the behavior activation, respectively, as shown in Figure 4.1(b). The goal state activation g is a normalized state value [1], and $g = 1$ when the situation is the goal state. When the module receives the behavior activation b from the higher modules, it calculates the optimal policy for its own goal, and sends action commands to the lower module. The action command at the bottom level is translated to an actual motor command, then the robot takes the action in the environment.

[1]The state value function estimates the sum of the discounted reward received over time when the robot takes the optimal policy, and is obtained by Q learning.

One basic idea is to use the goal state activations g of the lower modules as the representation of the situation for the higher modules. Intuitively, we can regard that the state value function represents how close the robot is to the goal if the module received reward only when it reached its goal, because the state value function estimate the sum of the discounted reward received over time when the robot takes the optimal policy. The state of the higher modules is constructed using the pattern of the goal state activations of the lower modules. In contrast, the actions of the higher level modules is constructed using the behavior activations to the lower modules.

4.2.2 Continuous Q learning

We use continuous Q learning [62] as a behavior learning method of the modules which is a modified version of the normal Q-learning. Here, we will briefly review the basics of continuous Q-learning.

First, we quantize the state (action) space arbitrary. Each quantized state (action) can be the representative state (action). The state (action) representations are given by a contribution value vector of the representative states $(w_1^s, \cdots, w_n^s)$ (actions $(w_1^a, \cdots, w_m^a)$). A contribution value indicates the closeness to the related representative state (action). The summation of the contribution values is one.

The Q-value when executing the representative action a_j at the representative state s_i is denoted by $Q_{i,j}$. A Q-value at any state and action pair is given by:

$$Q = \sum_{i=1}^{n} \sum_{j=1}^{m} w_i^s w_j^a Q_{i,j} \tag{4.1}$$

Given the representative state s_i, the optimal representative action is calculated by $\arg\max_j Q_{i,j}$. The optimal action contribution vector $\boldsymbol{a}^*$ for any state $\boldsymbol{s}$ is given by:

$$\boldsymbol{a}^* = \boldsymbol{w}^{a*} = \sum_{i=1}^{n} w_i^s e(\arg\max_j Q_{i,j}) \tag{4.2}$$

where $\boldsymbol{e}(k)$ denotes an M-dimensional vector of which k-th component is one and of which others are zeros. In order to obtain the optimal action based on eq.(4.2), $\max Q$ is calculated by:

$$\max Q = \sum_{i=1}^{n} \sum_{j=1}^{m} w_i^s w_j^{a*} Q_{i,j} \tag{4.3}$$

Then, the Q value when choosing an action $\boldsymbol{a}$ at the current state $\boldsymbol{s}$, and transiting the next state $\boldsymbol{s}'$ given reward r is updated by:

$$Q_{i,j} \leftarrow Q_{i,j} + \alpha_t w_i^s w_j^a (r + \gamma V(\boldsymbol{s}') - Q(\boldsymbol{s}, \boldsymbol{a})) \tag{4.4}$$

where $\max Q^{t'}$ denotes Q value when choosing the optimal action at the next state.

4.2.3 State and Action Space Construction

Learning modules at the bottom level construct the state/action space using the sensory information and the motor command of the robot. Learning modules at the higher levels construct the representative states and actions using the goal state activations and behavior activations of the lower modules, respectively. That is, the contribution vector of the representative states and actions at the higher modules is given by the normalized pattern of the goal state activations and behavior activations of the lower modules.

4.2.4 Self-distribution of Goal State

The basic idea for the distribution of learning modules is "to assign the goal state of each learning module in the state space uniformly". However, it seems difficult

① to find out how the state space does extend, or

② to define a distance function in the state space without robot's experiences.

These problems will occur especially among the layers which are higher than the bottom one.

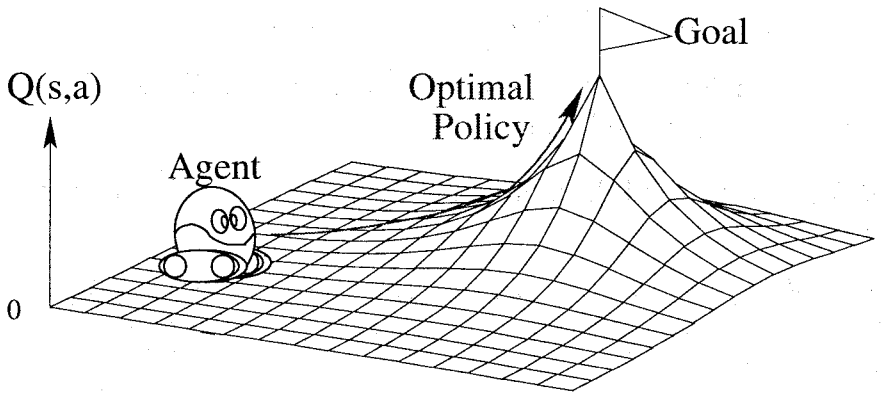


Figure 4.2: State Value V(s) represents how close the agent is to the goal.

Now, we can use the state value function as the distance function which estimates the distance to its own goal state, We can regard that $V(s)$ represents how close the robot is to the goal if the robot received reward only when it reach its goal.

Figure 4.3 shows the distribution procedure of each learning module's goal state in the state space uniformly. It shows a case of one dimension state space, however, the procedure is same way in the case of multi-dimension one. The vertical axis indicates a goal state activation of a learning module. Figure 4.3 (a) shows an example of the learning module distribution at the initial stage. There are three learning modules, and they are not distributed uniformly. The region where the goal state activations of other modules are low could be judged that there is no learning module of which goal state is near. Then, the learning system adds a new learning module in that place (Figure 4.3 (b)). When the density of learning module is high, the goal state activations of the learning modules are high. Then, the system moves a learning module's goal state to the region where the goal state activations of other learning modules are low (Figure 4.3 (c)). When the density of learning module is still high, the system deletes an learning module. As a result, the goal state of each learning module distributes in the state space uniformly after the learning (Figure 4.3 (d)).. The algorithm is as follows.

① $S^{neighbor} = \{s^{neighbor}$ such that contribution value is larger than a threshold $T\}$

② If $||S^{neighbor}|| = 0$, then exit, where $||\cdot||$ means the number of elements.

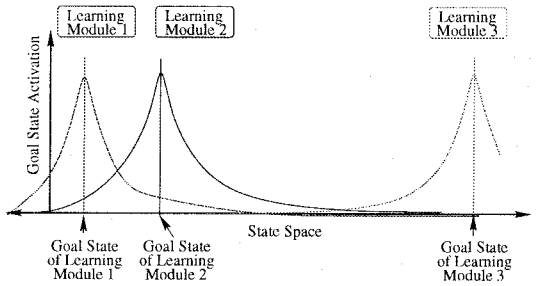
③ Search a learning module $module_{query}$ which has its goal state in the $S^{neighbor}$

④ Calculate the distribution of the maximum goal state activation $V_{\max}^{noquery}(s^{neighbor})$ of the learning modules which are NOT $module_{query}$

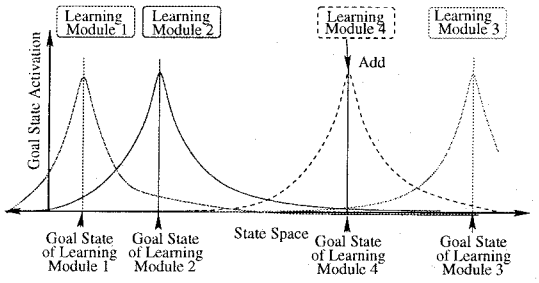
⑤ If $module_{query}$ doesn't exist, and maximum of the $V_{\max}^{noquery}(s^{neighbor})$ is low, then add a new learning modules and exit.

⑥ If $module_{query}$ exists and the minimum of $V_{\max}^{noquery}(s^{neighbor})$ is high, delete $module_{query}$ and exit.

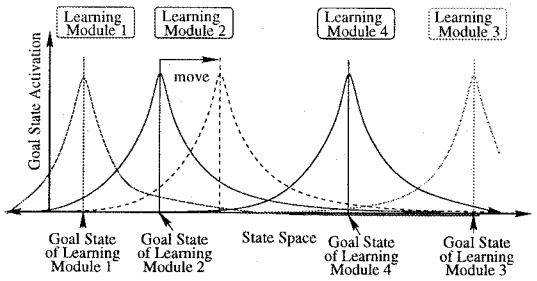
⑦ Search $s_{\min}^{neighbor}$ which is the state where the $V_{\max}^{noquery}(s^{neighbor})$ is minimum among $S^{neighbor}$. If $module_{query}$ exist and its goal state is not $s_{\min}^{neighbor}$, then move the goal state to $s_{\min}^{neighbor}$



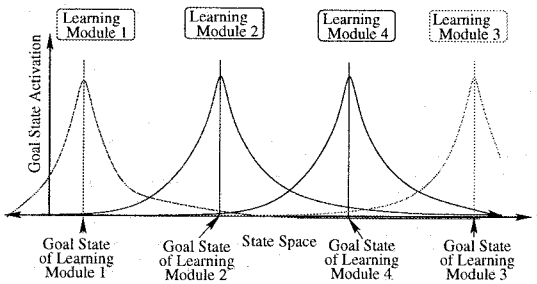
(a)Case 1



(b)Case 2

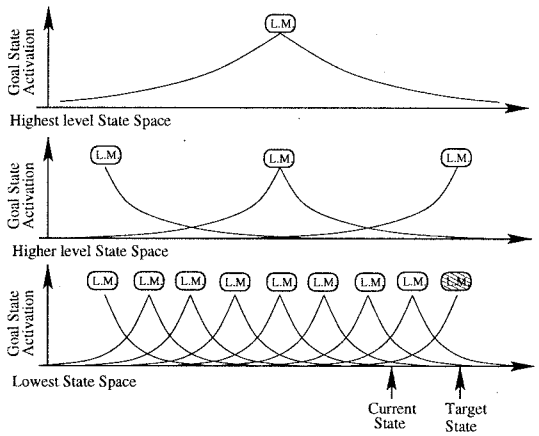


(c)Case 3

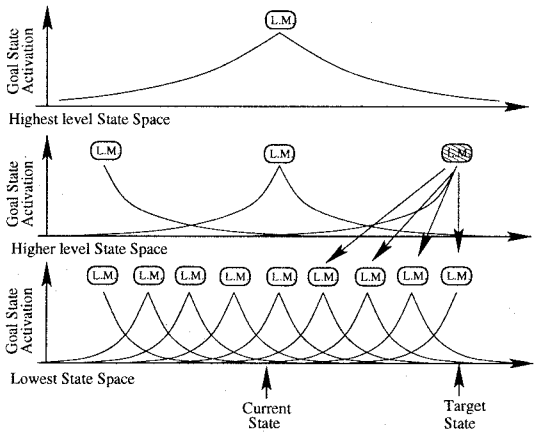


(d)Case 4

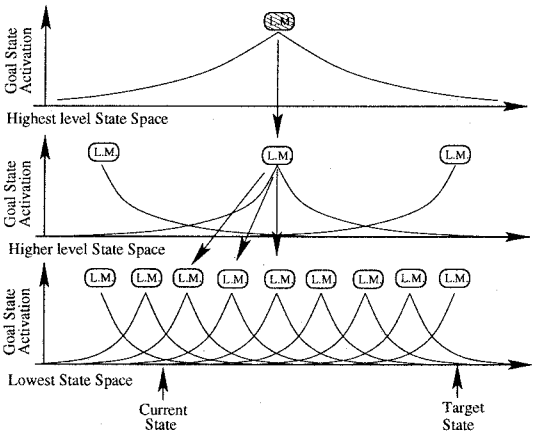
Figure 4.3: An example of the assignment of the goal state among learning modules



(a)Case 1



(b)Case 2



(b)Case 3

Figure 4.4: Strategy in the multi-layered control structure (L.M. stands for learning module).

4.2.5 Construction of Layer

We have described the distribution of the learning modules among each layer. The learning system makes multi-layer by superposing them. Because it assigns fewer learning modules than the number of states, fewer number of states is assigned at higher layer. Then fewer learning modules will assigned at the higher layer. The layering procedure stops when the number of learning module is one at the top layer.

4.2.6 Strategy in the Multi-Layered Learning System to Accomplish A Task

The target state is given to the multi-layered learning system in the state space at the bottom layer. First of all, the system searches the learning module which is at the target state. If the learning module can accomplish the given task, that is it can reach the target state using its policy, the system sets the behavior activation of the learning module. The system judges whether the learning module accomplish the given task or not by its Q value at the current situation. That is, if Q value is hight, then the module has a policy to reach the target state, while if Q value is low, the module has not experienced the situation, or the situation is very far from its goal state. Then, if Q value is higher than an threshold, the system judges that the module can accomplish the given task.

If the learning module $module_g^0$ which has the nearest goal state to the given target state s_{target}^0at the bottom layer cannot accomplish the given task, the system makes the state at the higher layer which is related to the module $module_g^0$ be the target state s_{target}^1, and searches the learning module $module_g^1$ which is nearest to the target state s_{target}^1. If this learning module $module_g^1$ can reach the target state s_{target}^1 from the current situation, then the system sets the behavior activation of the learning module. This learning module $module_g^1$ sends its command to the lower layer by setting the behavior activations of lower learning modules, then reaches its goal state (Figure 4.4 (b)). If the system reaches the region which the learning module $module_g^0$ at bottom layer can deal with, that is, the situation becomes in case 1 in Figure 4.4, it starts the learning module $module_g^0$, then move to the given target state in the same way as the first step.

If the learning module $module_g^1$ cannot deal with the current situation, the system does same way at upper layer (Figure 4.4 (c)). The multi-layered system sets behavior activation of only one learning module at each layer because of avoidance of conflict among learning modules' policies.

4.3 Experiments

4.3.1 Overview

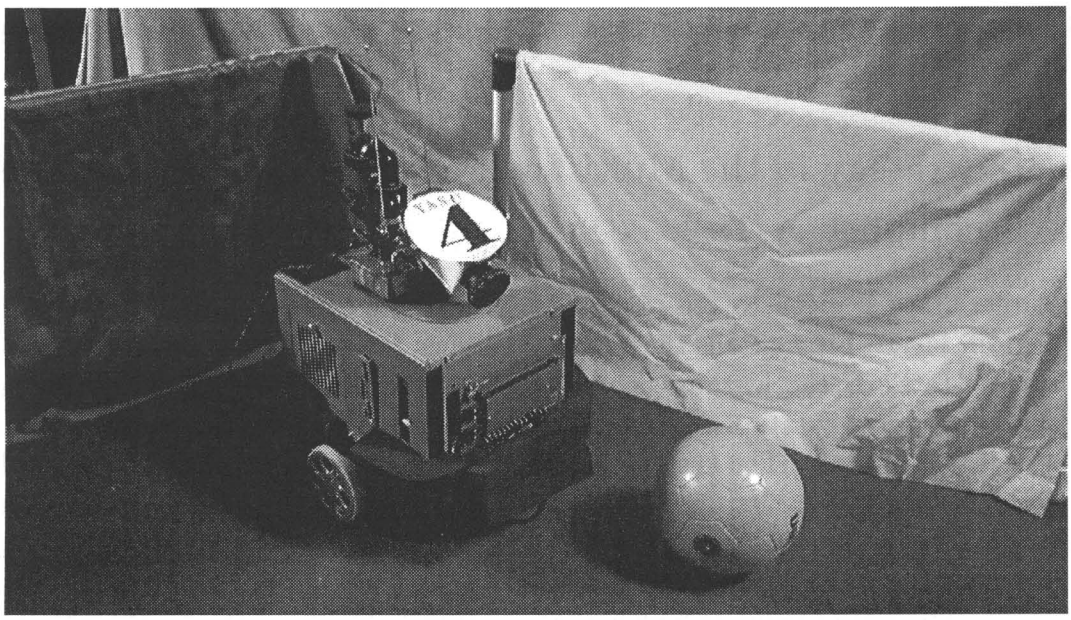


Figure 4.5: A mobile robot, a ball and goals

To evaluate the proposed method, we apply it to a simple navigation task. The target situation is given by reading the sensor information when the robot is at the target position.

Figure 4.5 shows a picture of a mobile robot we designed and built, a ball, and a goal. Figure 4.6 shows an overview of the robot system. It has two TV cameras. One has a wide-angle lens of which visual angles are 35 degrees and 30 degrees in horizontal and vertical directions, respectively. The camera is tilted down 23.5 degrees to capture the ball image as large as possible. Other has a omni-directional mirror and is mounted on the robot. The driving mechanism is PWS (Power Wheeled System), and the action space is constructed in terms of two torque values to be sent to two motors that drive two wheels. These parameters of the system are unknown to the robot, and it tries to estimate the mapping from sensory information to appropriate motor commands by the method. The environment consists of the ball, and the goal, and the mobile robot.

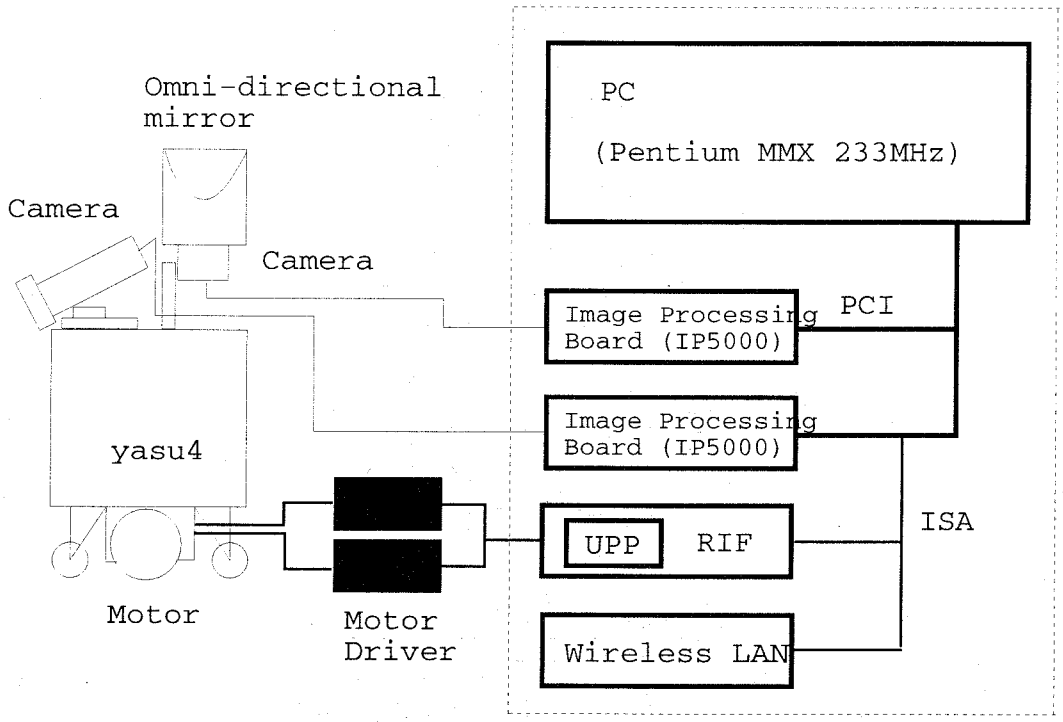


Figure 4.6: An overview of the robot system

In this experiment, the robot receives the information of only one goal, for the simplicity. The state space at the bottom layer is constructed in terms of the centroid of goal images of the two cameras and is tessellated both into 9 by 9 grids. And the action space is constructed in terms of two torque values to be sent to two motors corresponding to two wheels and is tessellated into 3 by 3 grids. Consequently, the number of representative state and action are $162(9 \times 9 \times 2)$ and $9(3 \times 3)$, respectively. The representative state and action at the upper layer is constructed by the learning modules at the lower layer which are automatically assigned.

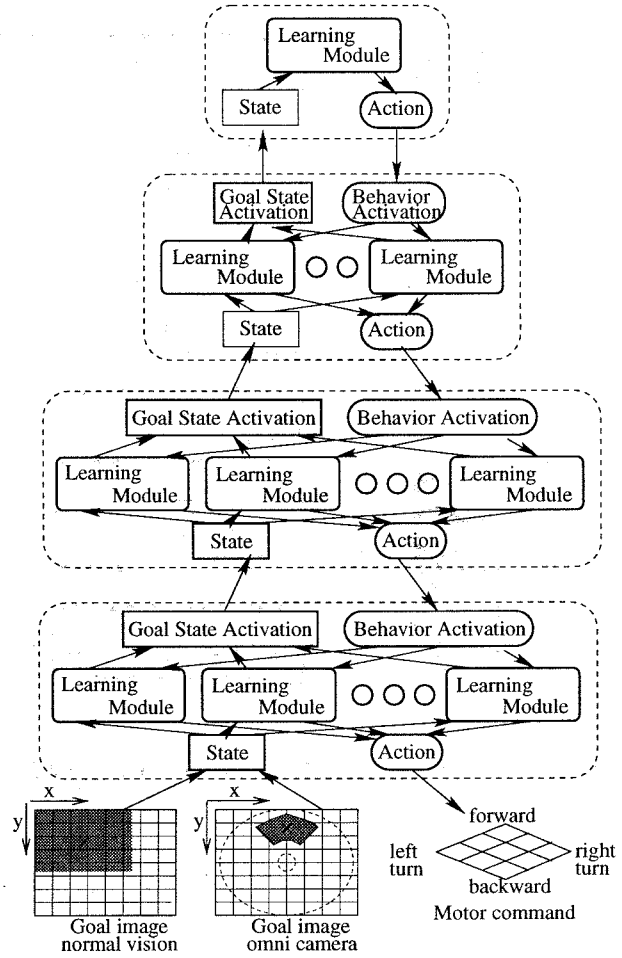


Figure 4.7: A hierarchical architecture of learning modules

4.3.2 Experiment Results

The experiment is constructed with two stages, one is the learning one and other is the task execution one using the learned result. First of all, the robot moved at random in the environment for about two hours. The system learned and constructed the four layers and one learning module exist at the top layer (Figure 4.7). We call each layer from the bottom, bottom, middle, upper, top layer. In this experiment, the system assigned 40 learning modules at the bottom layer, 15 modules at the middle layer and 4 modules at the upper layer. Figures 4.8 and 4.9 show the distribution of goal state activations of learning modules at the bottom layer in the state spaces of

wide-angle camera image and omni-directional mirror image, respectively. The x, y axes indicate the centroid of goal images. The numbers on the figures indicate the numbers of learning modules. The figures show that each learning module is assigned on the state space uniformly.

The task for the robot is reaching a specified position using this multi-layer learning structure. The robot was located far from the goal, and faced opposite direction to it as an initial position. The target position was locating in front of the goal and watching it. Figures 4.10 (a), (b), and (c) show the time development of the goal state and the behavior activations of learning modules at the bottom layer, the middle layer and the upper layer, respectively. We omitted the time development of the top learning module's activation because only one learning module existed at the top layer, and it have never been set behavior activation. The straight line segments on top of the figure indicate the development of the behavior activations. The numbers in the Figure 4.10 (a) indicate the numbers of learning modules at the bottom layer, and are correspond to the same numbers in the Figures 4.8 and 4.9. Figure 4.11 shows a rough sketch of the state transition and the commands to the lower layer on the multi-layer learning system. This figure corresponds to the Figures 4.10 (a), (b) and (c). The circles in the figure indicate the learning module, and the number in the circle indicates the number of the learning module. The up arrows indicate that the upper learning module recognizes the state which corresponds to the lower module as the goal state. The thin solid lines indicate the state transition while the robot accomplished the task. The down arrows indicate that the upper learning module set the behavior activation of the lower learning module. When the robot located at the initial position, the learning module 25 at the bottom layer, the learning module 10 at the middle layer, and the learning module 1 at the upper layer are near to their own goal states. When the robot located at the target position, the learning module 1 at the bottom layer, the learning module 7 at the middle layer and the learning module 0 at the upper layer are near to their own goal states. First of all, the system tried to activate the learning module 1 at the bottom layer. However, the module could not manage the current situation, then the system tried to activate the learning module 7 at the middle layer. But, the module could not handle the current situation, either, then the system activated the learning module 0 at the upper layer. The learning module 0 at the upper layer activated the learning module 15 at the middle layer, then this middle layer module activated the learning modules 27 and 13 at the bottom layer until about the 40 th step. Next, the learning module 7 at the middle layer became able to handle the situation, and activated the learning modules 30 and 26 at the bottom layer until about the 360 th step. Finally, the learning module 1 at the bottom layer became able to handle the situation, and the system reached the target position using this module.

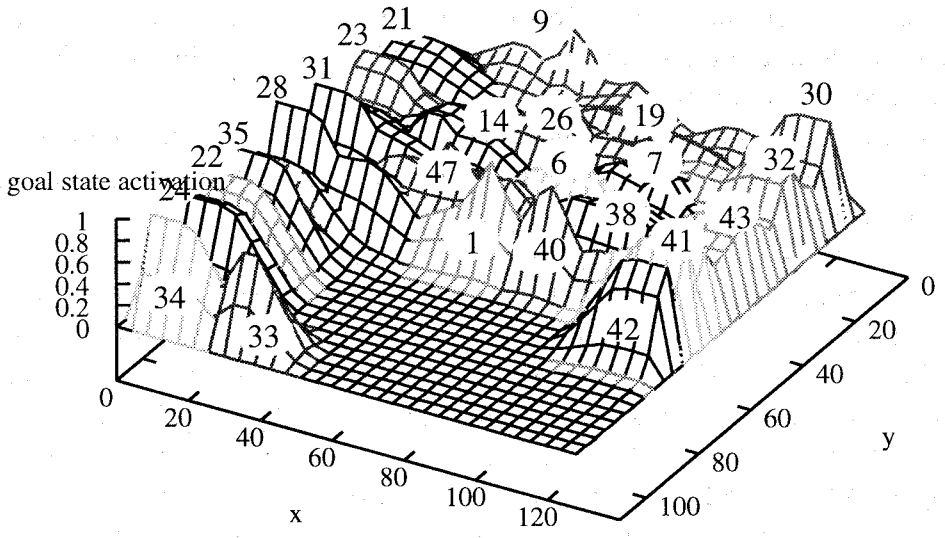


Figure 4.8: The distribution of learning modules at bottom layer on the normal camera image

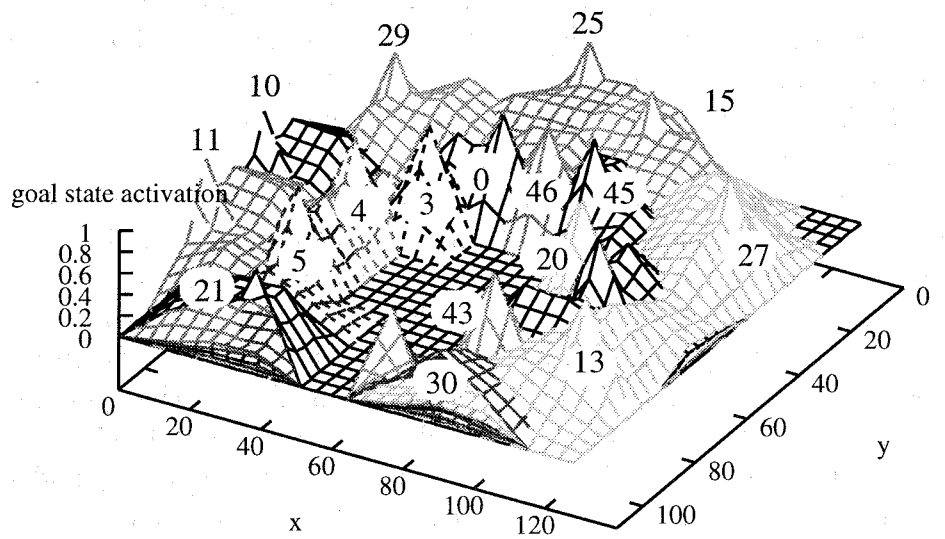
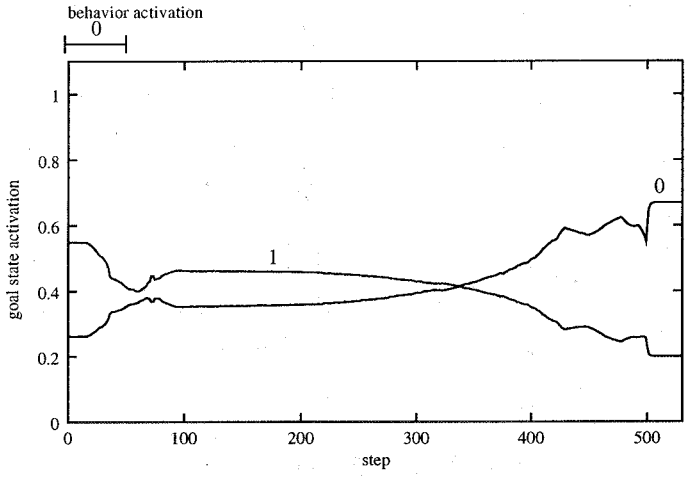
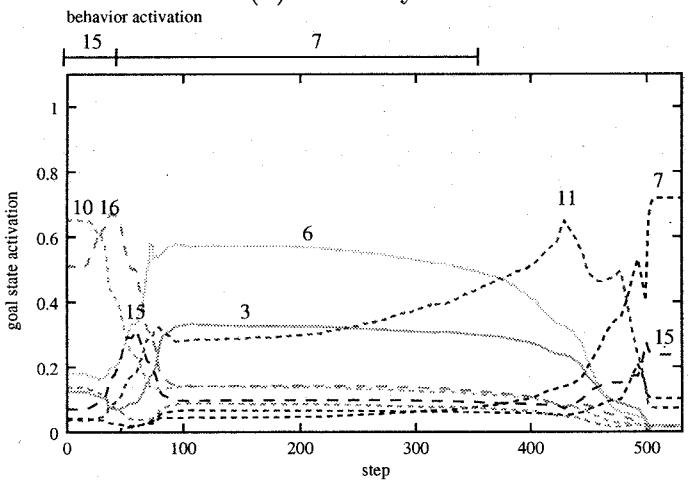


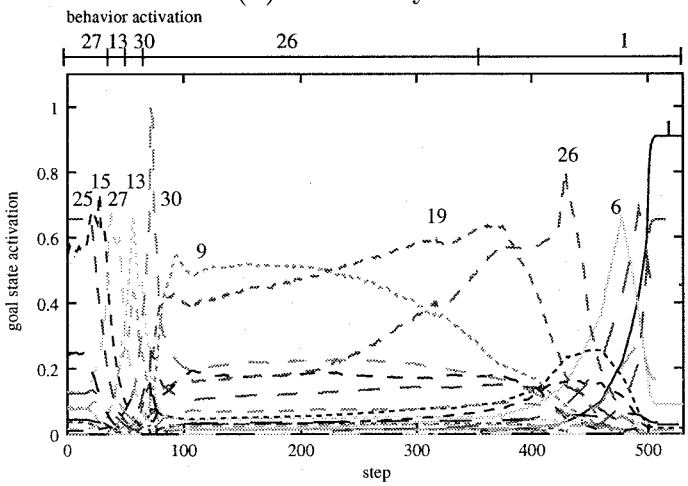
Figure 4.9: The distribution of learning modules at bottom layer on the omni-directional camera image



(a) third layer



(b) second layer



(b) first layer

Figure 4.10: A sequence of the goal state activation and behavior activation of learning modules at each layer

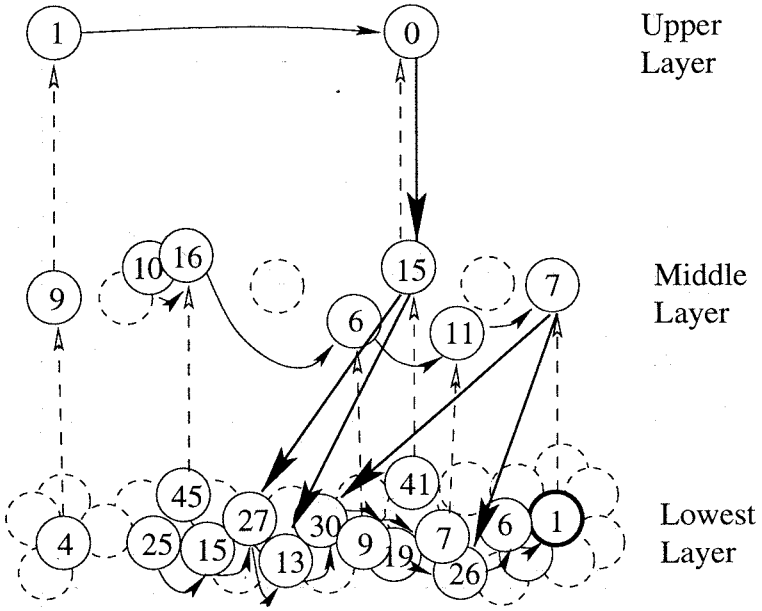


Figure 4.11: A rough sketch of the state transition on the multi-layer learning system

4.3.3 Appendix : sequence of learning module distribution

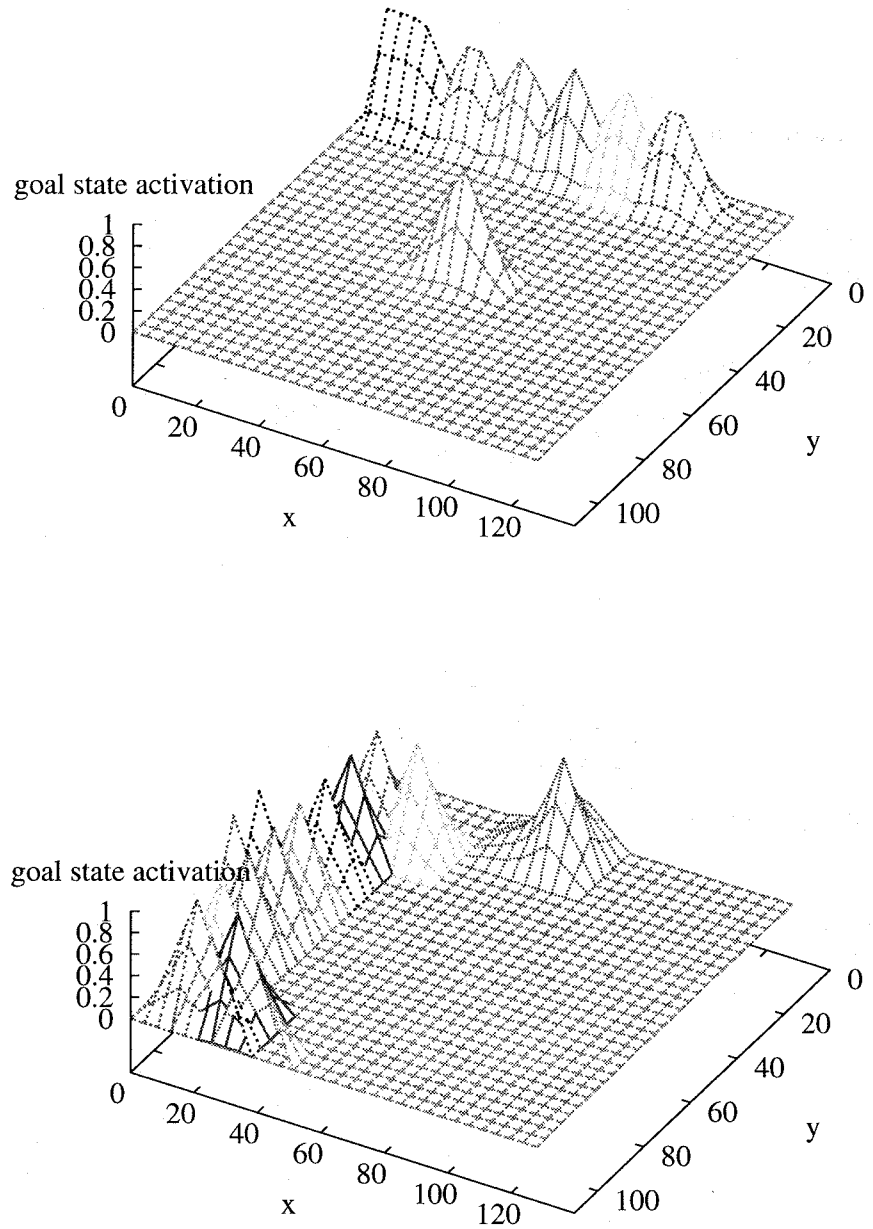


Figure 4.12: The distribution of learning modules at bottom layer (1000step): TOP:perspective camera image, BOTTOM:omni-directional camera image

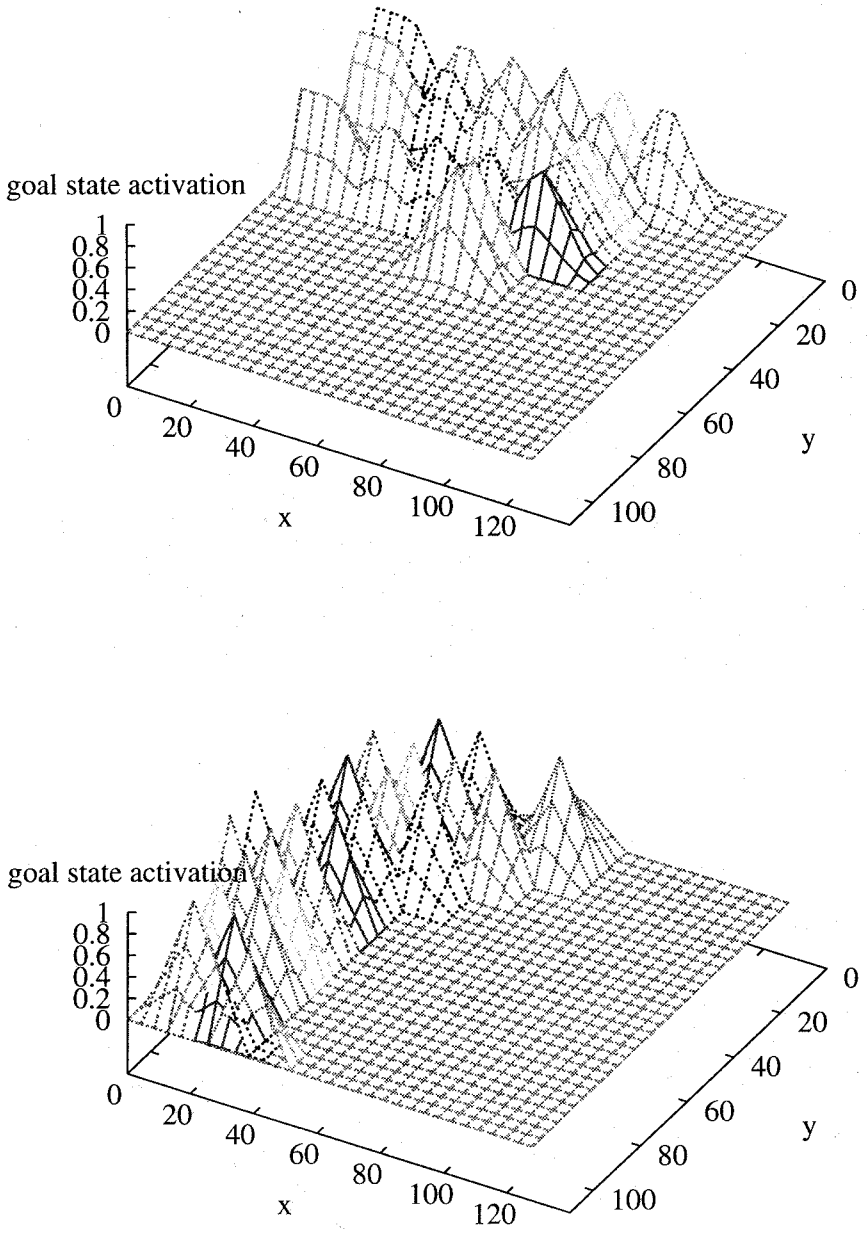


Figure 4.13: The distribution of learning modules at bottom layer (2000step): TOP:perspective camera image, BOTTOM:omni-directional camera image

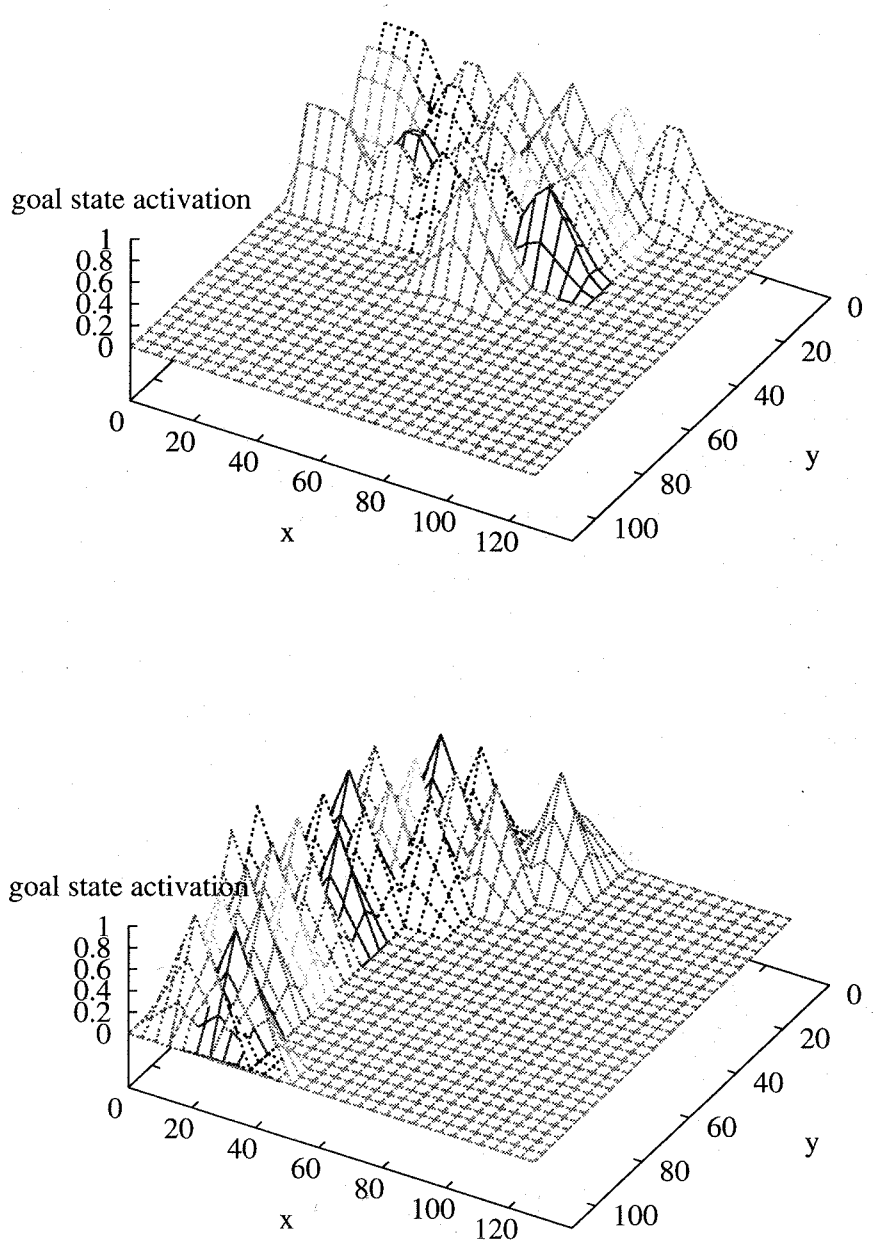


Figure 4.14: The distribution of learning modules at bottom layer (3000step): TOP:perspective camera image, BOTTOM:omni-directional camera image

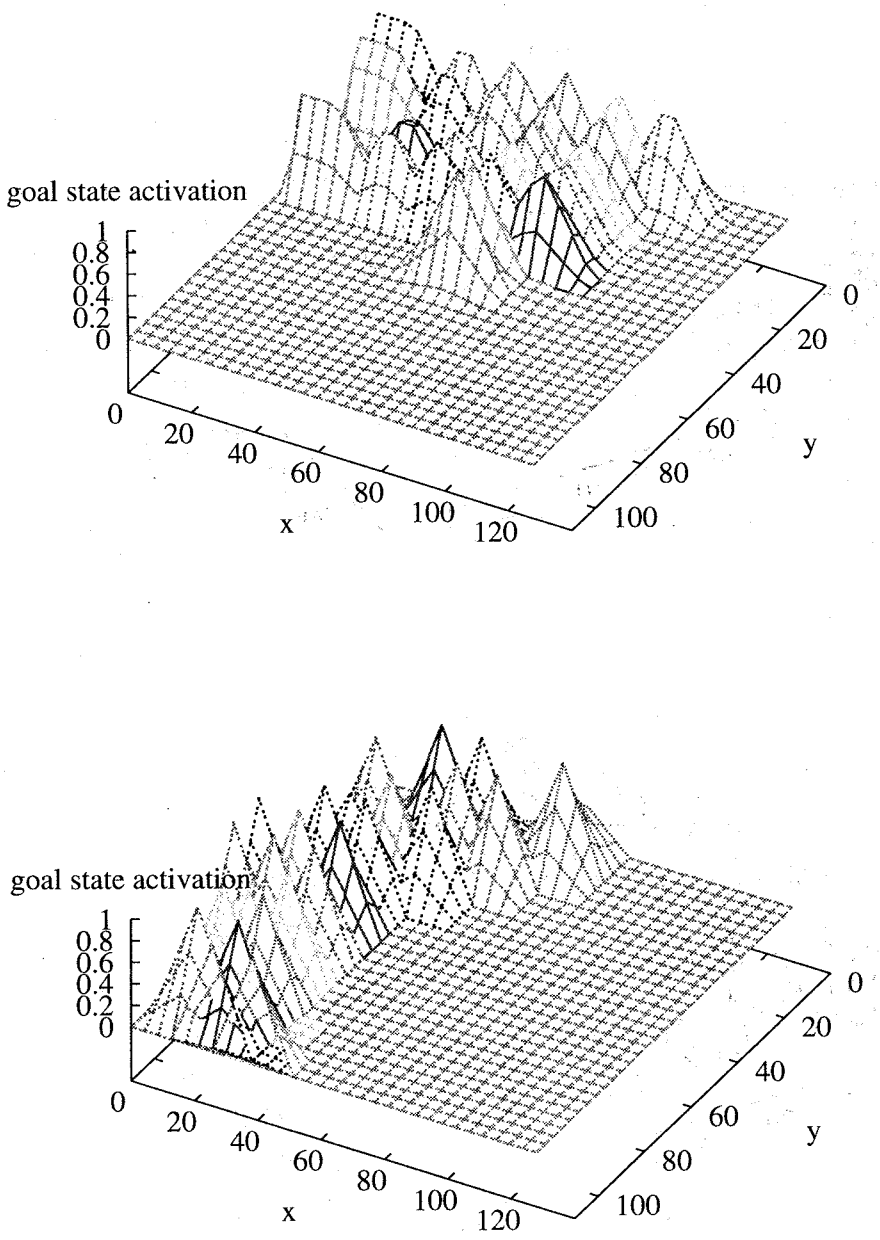


Figure 4.15: The distribution of learning modules at bottom layer (4000step): TOP:perspective camera image, BOTTOM:omni-directional camera image

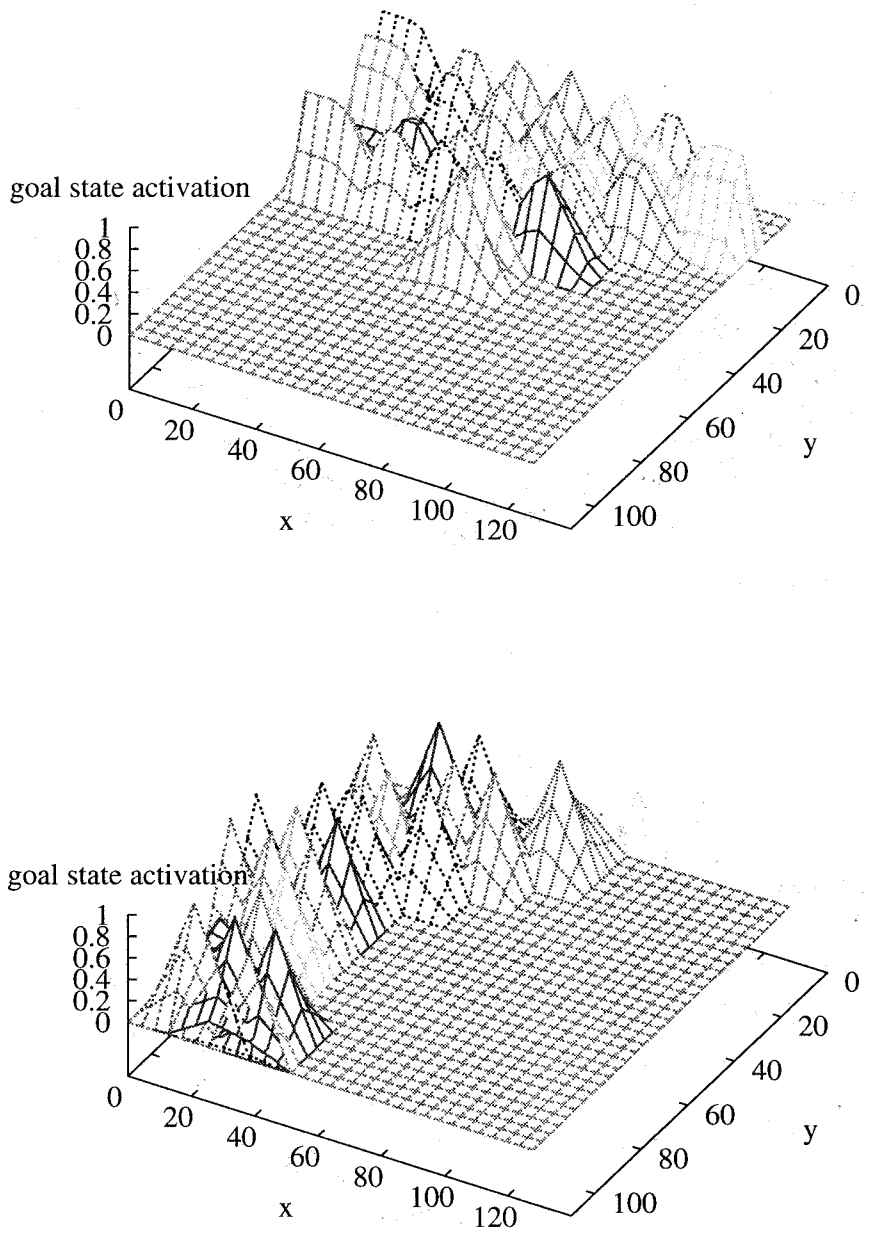


Figure 4.16: The distribution of learning modules at bottom layer (5000step): TOP:perspective camera image, BOTTOM:omni-directional camera image

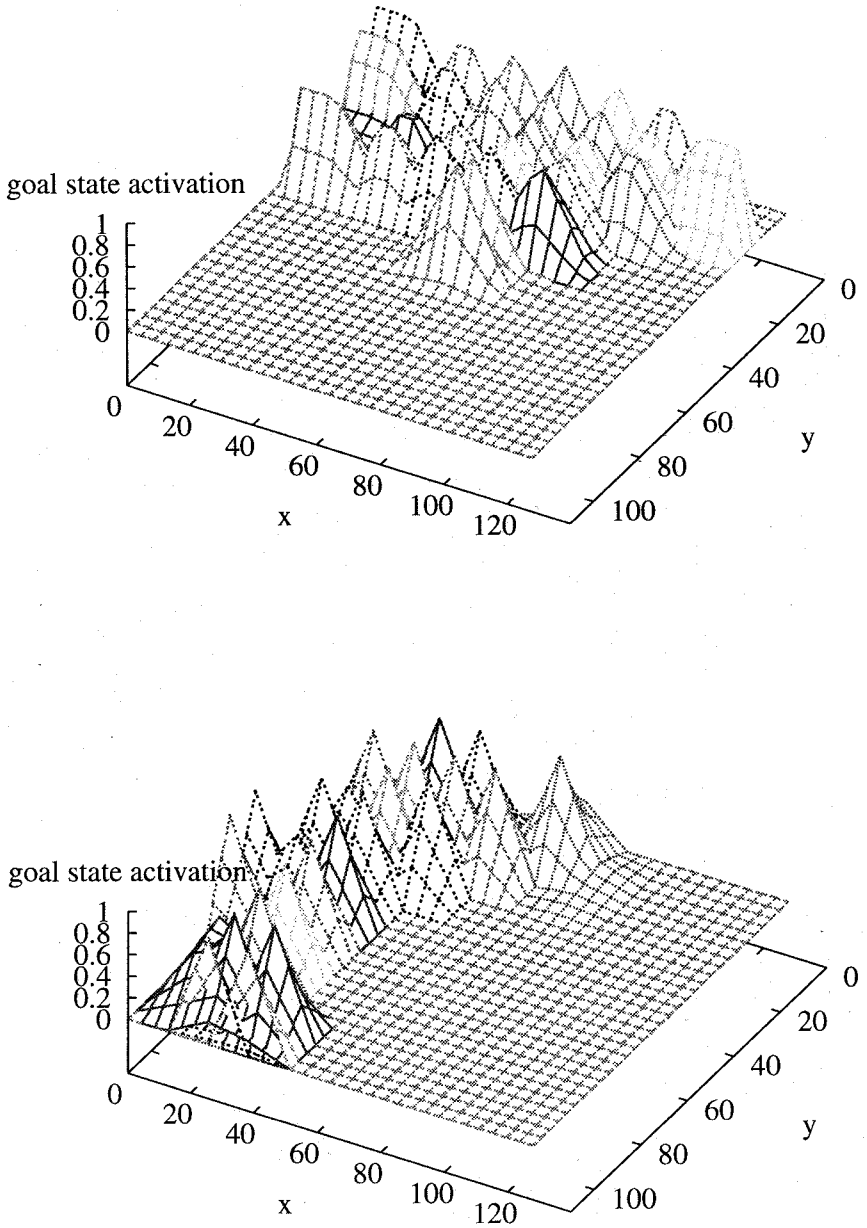


Figure 4.17: The distribution of learning modules at bottom layer (6000step): TOP:perspective camera image, BOTTOM:omni-directional camera image

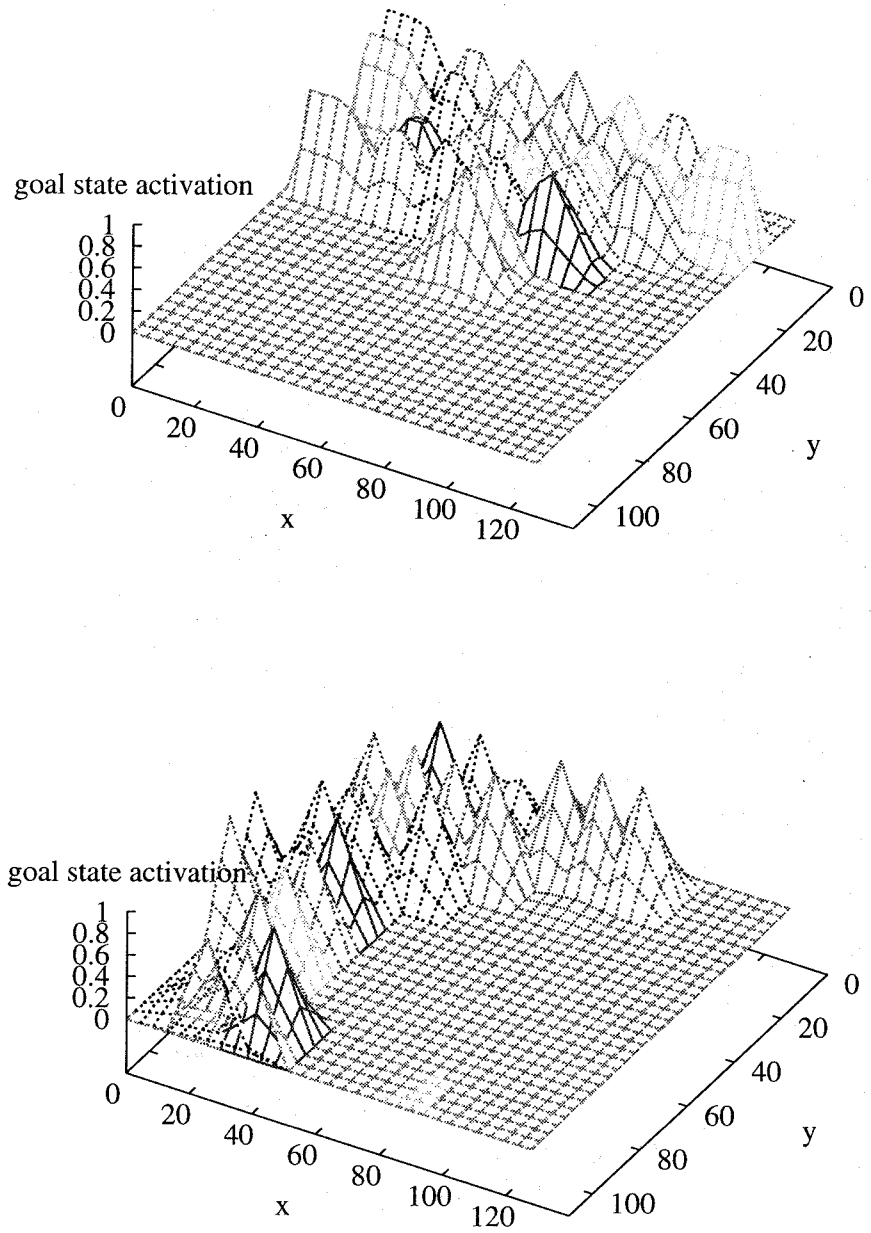


Figure 4.18: The distribution of learning modules at bottom layer (7000step): TOP:perspective camera image, BOTTOM:omni-directional camera image

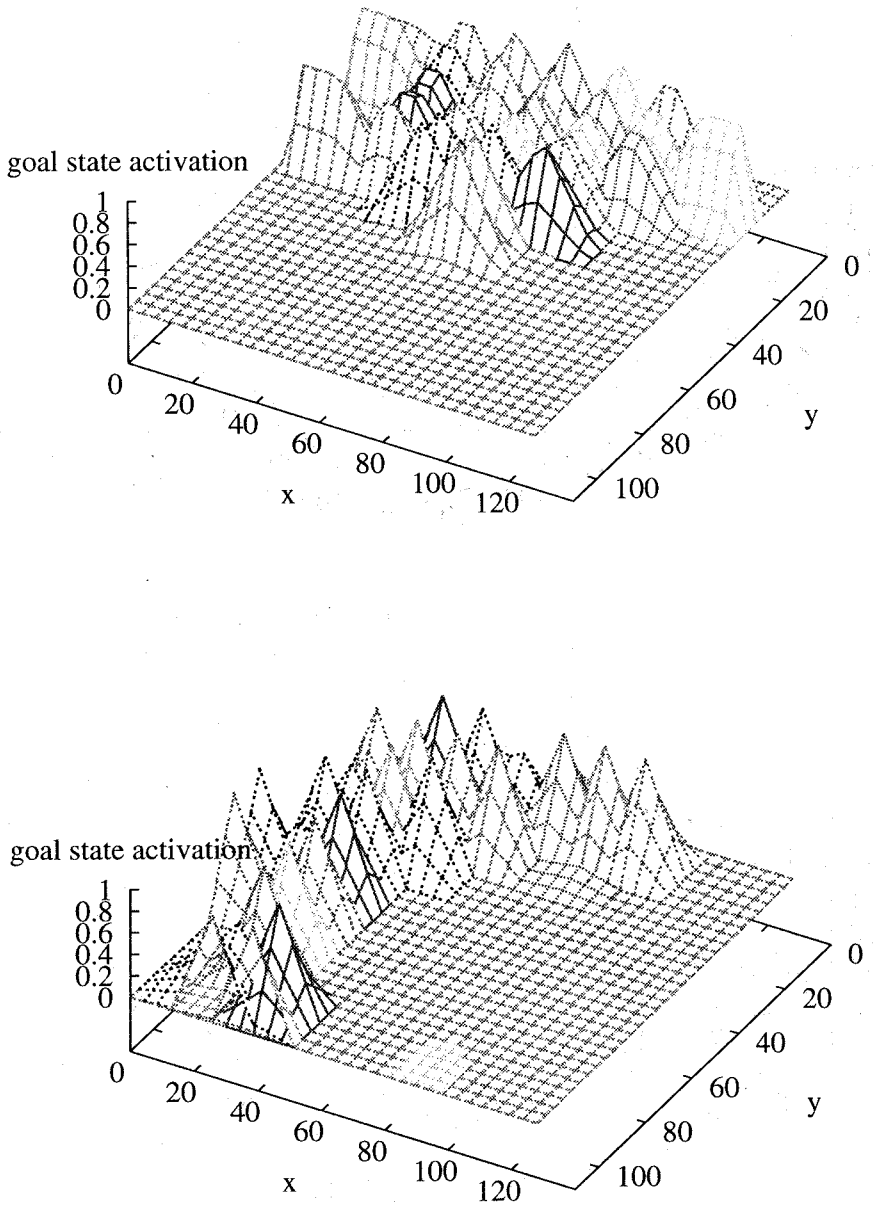


Figure 4.19: The distribution of learning modules at bottom layer (8000step): TOP:perspective camera image, BOTTOM:omni-directional camera image

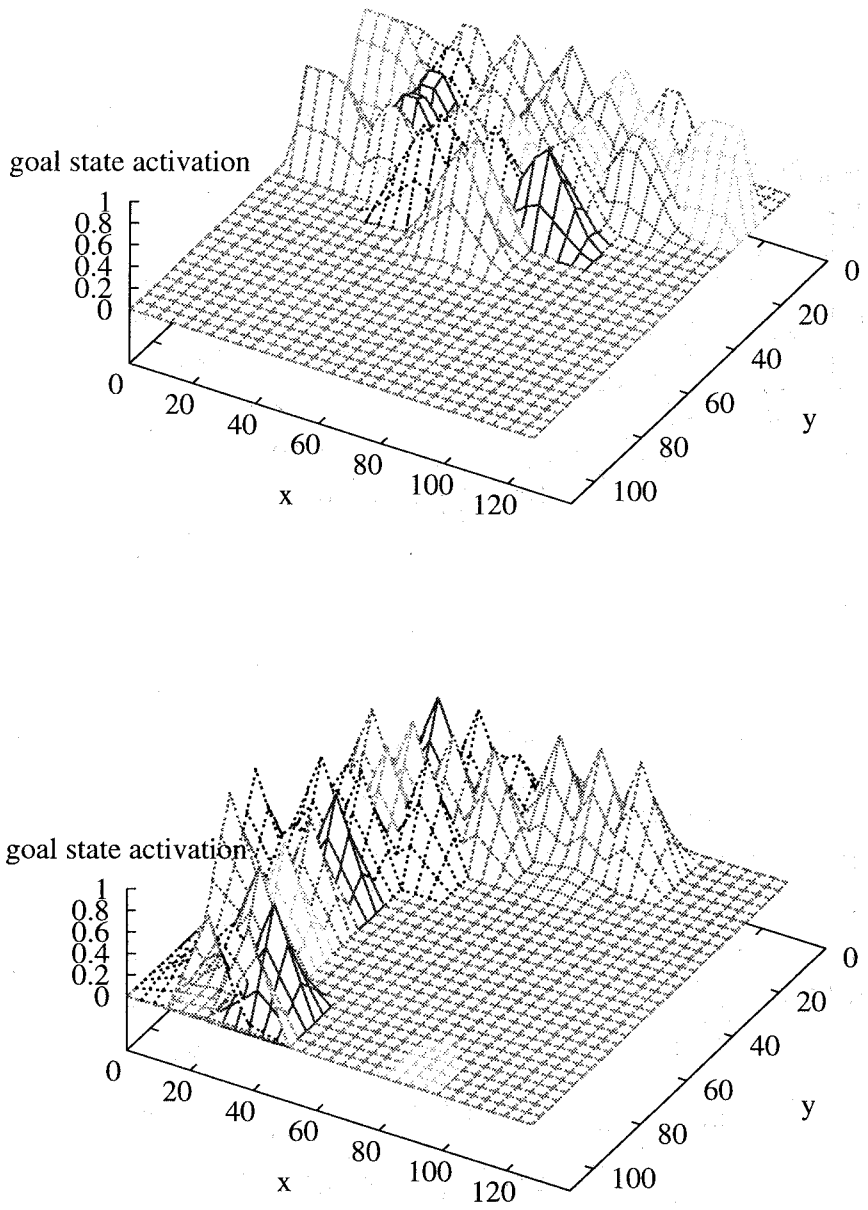


Figure 4.20: The distribution of learning modules at bottom layer (9000step): TOP:perspective camera image, BOTTOM:omni-directional camera image

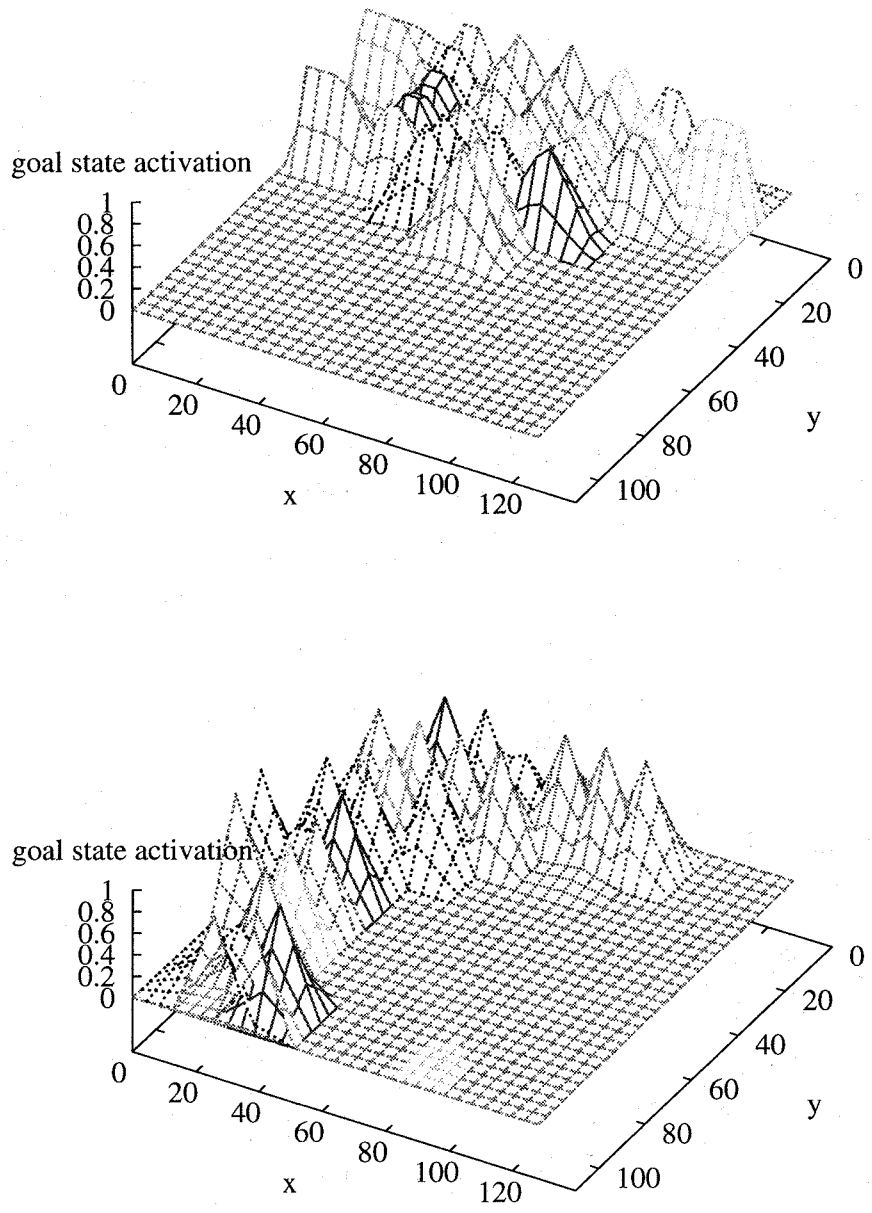


Figure 4.21: The distribution of learning modules at bottom layer (10000 step): TOP:perspective camera image, BOTTOM:omni-directional camera image

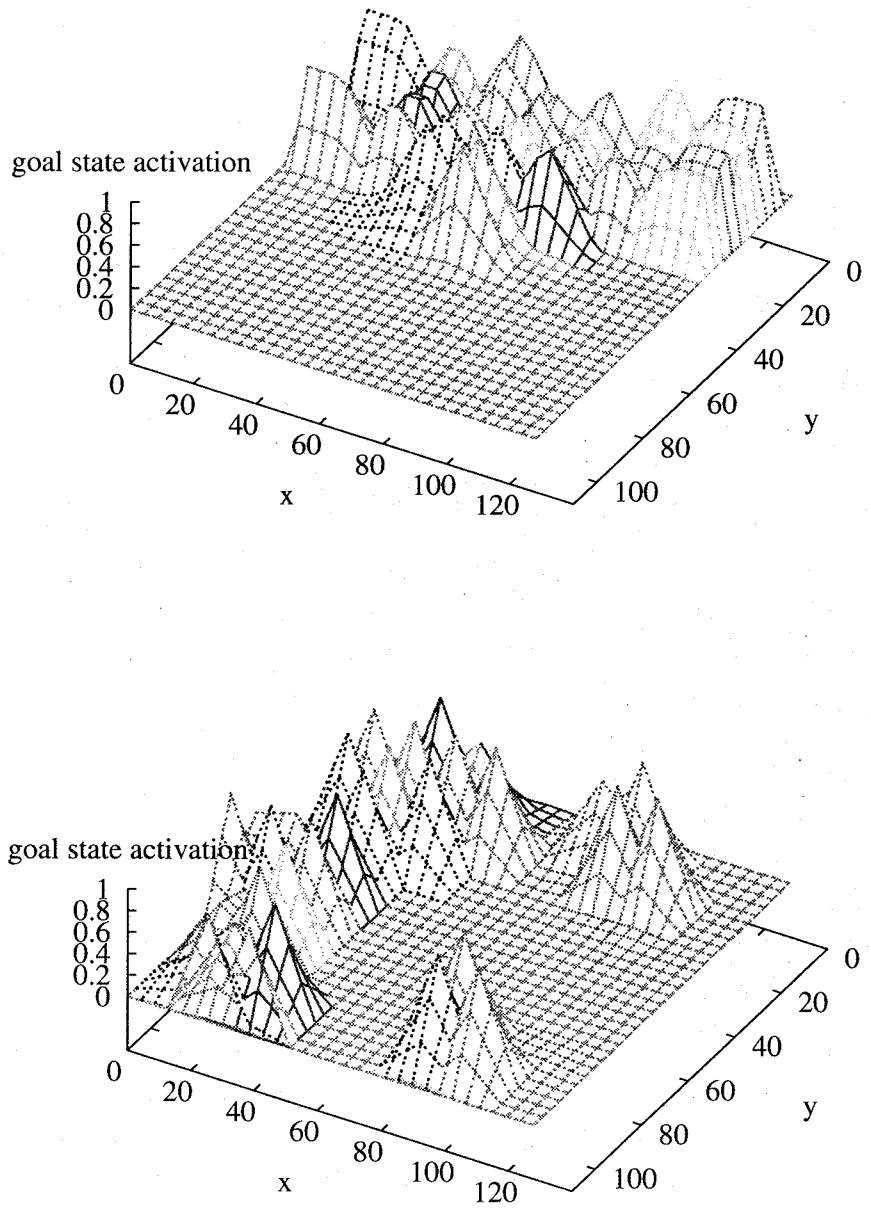


Figure 4.22: The distribution of learning modules at bottom layer (20000 step): TOP:perspective camera image, BOTTOM:omni-directional camera image

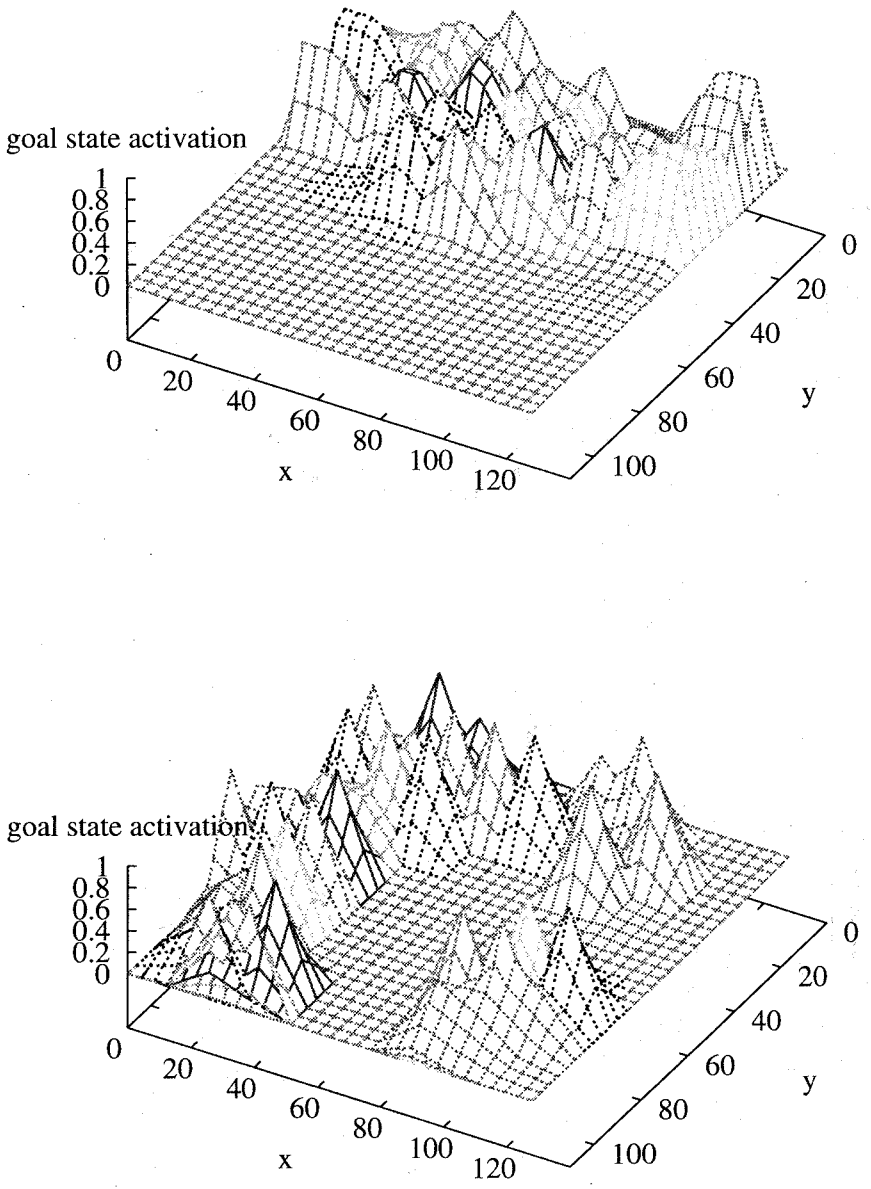


Figure 4.23: The distribution of learning modules at bottom layer (30000 step): TOP:perspective camera image, BOTTOM:omni-directional camera image

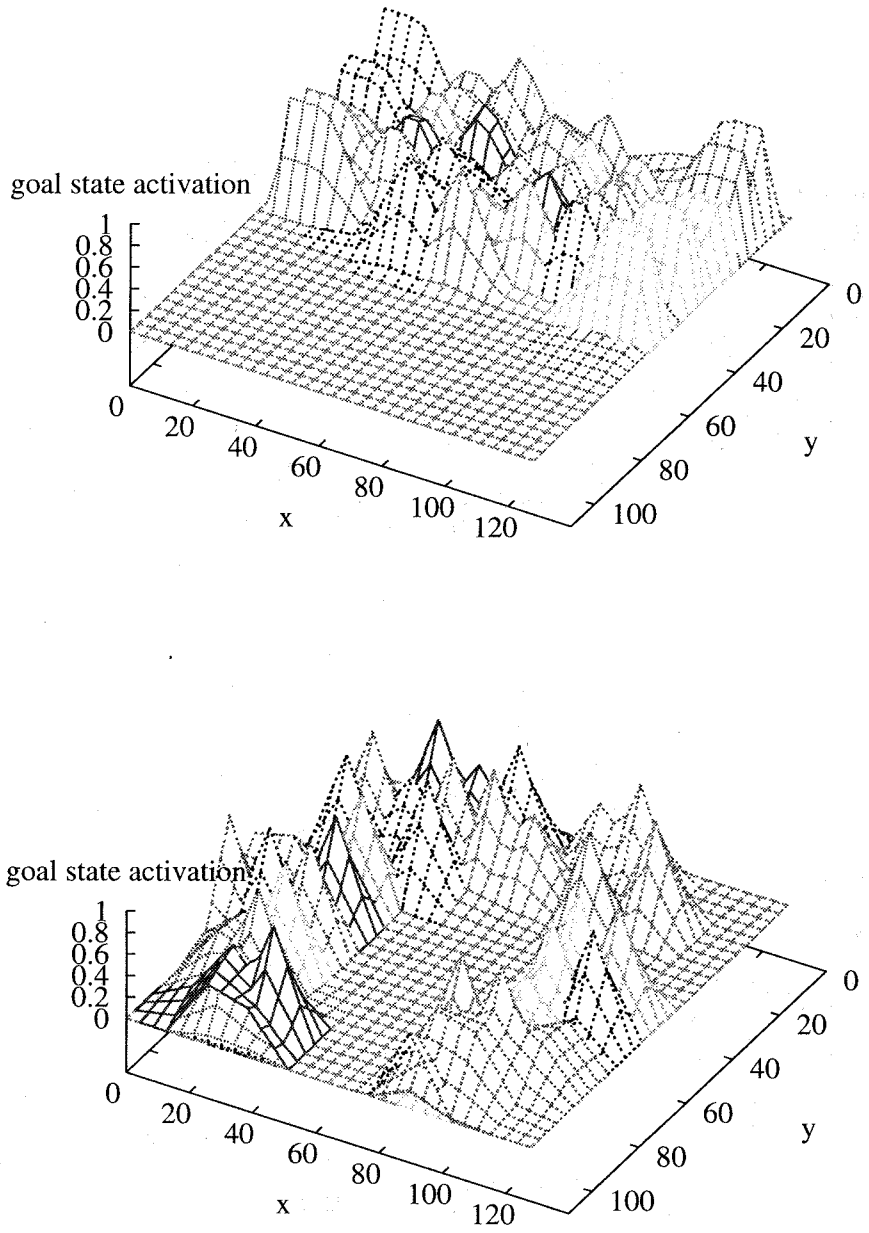


Figure 4.24: The distribution of learning modules at bottom layer (40000 step): TOP:perspective camera image, BOTTOM:omni-directional camera image

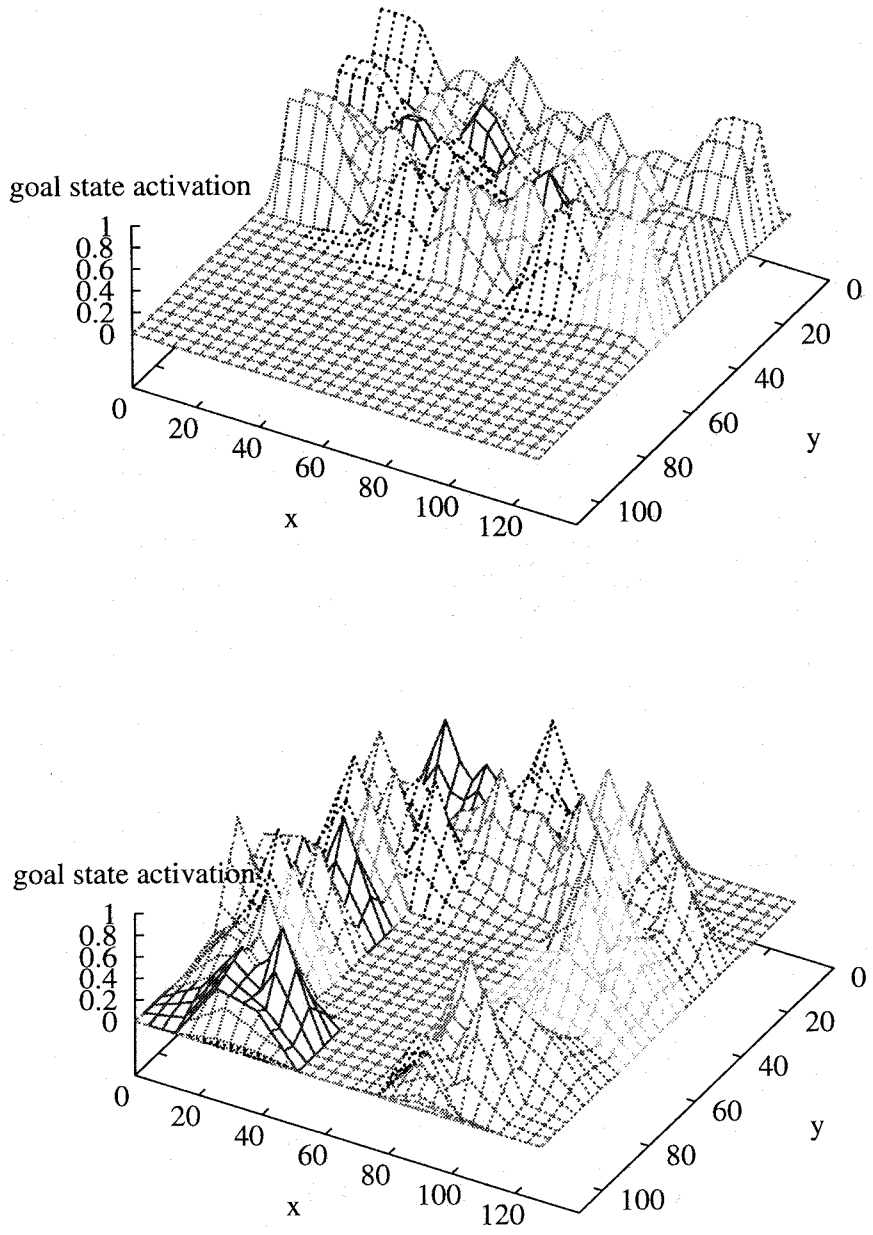


Figure 4.25: The distribution of learning modules at bottom layer (50000 step): TOP:perspective camera image, BOTTOM:omni-directional camera image

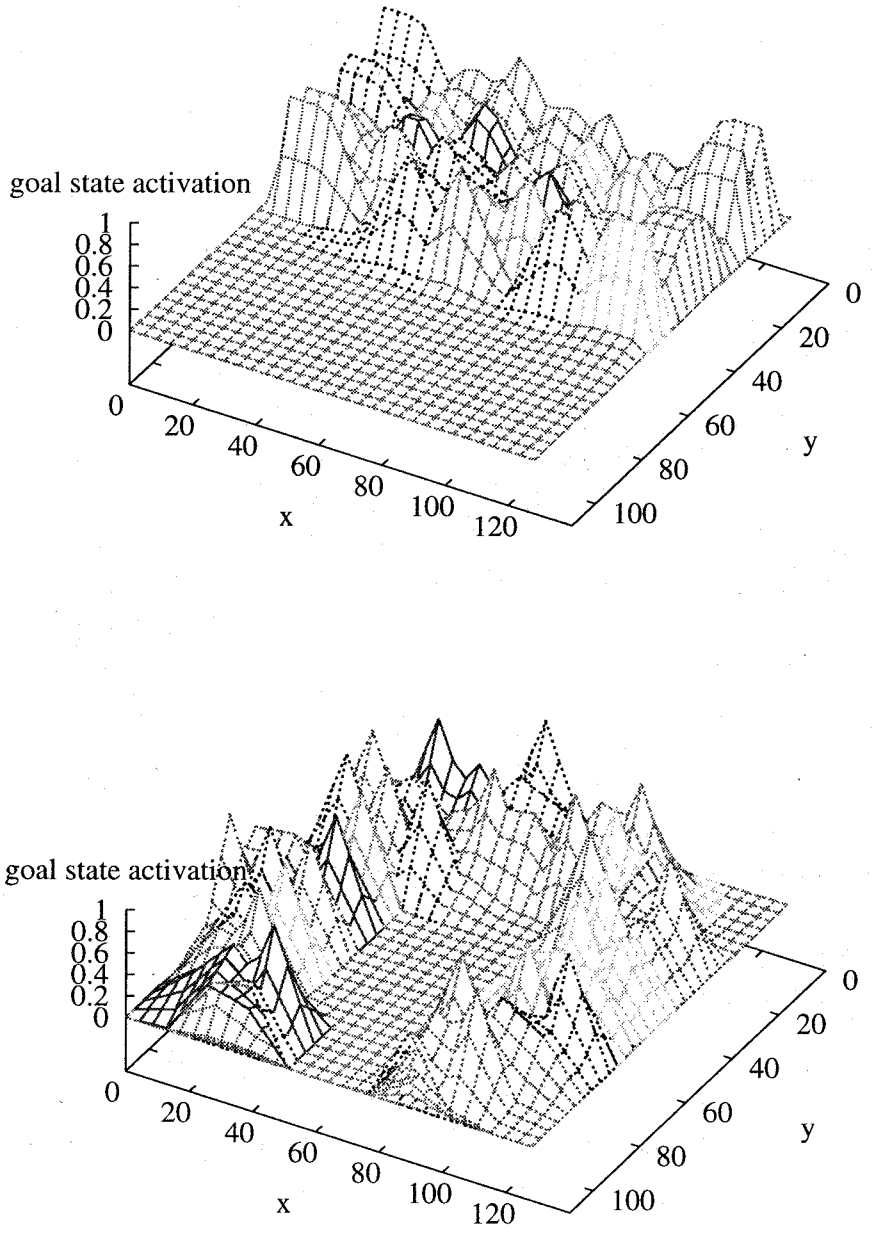


Figure 4.26: The distribution of learning modules at bottom layer (60000 step): TOP:perspective camera image, BOTTOM:omni-directional camera image

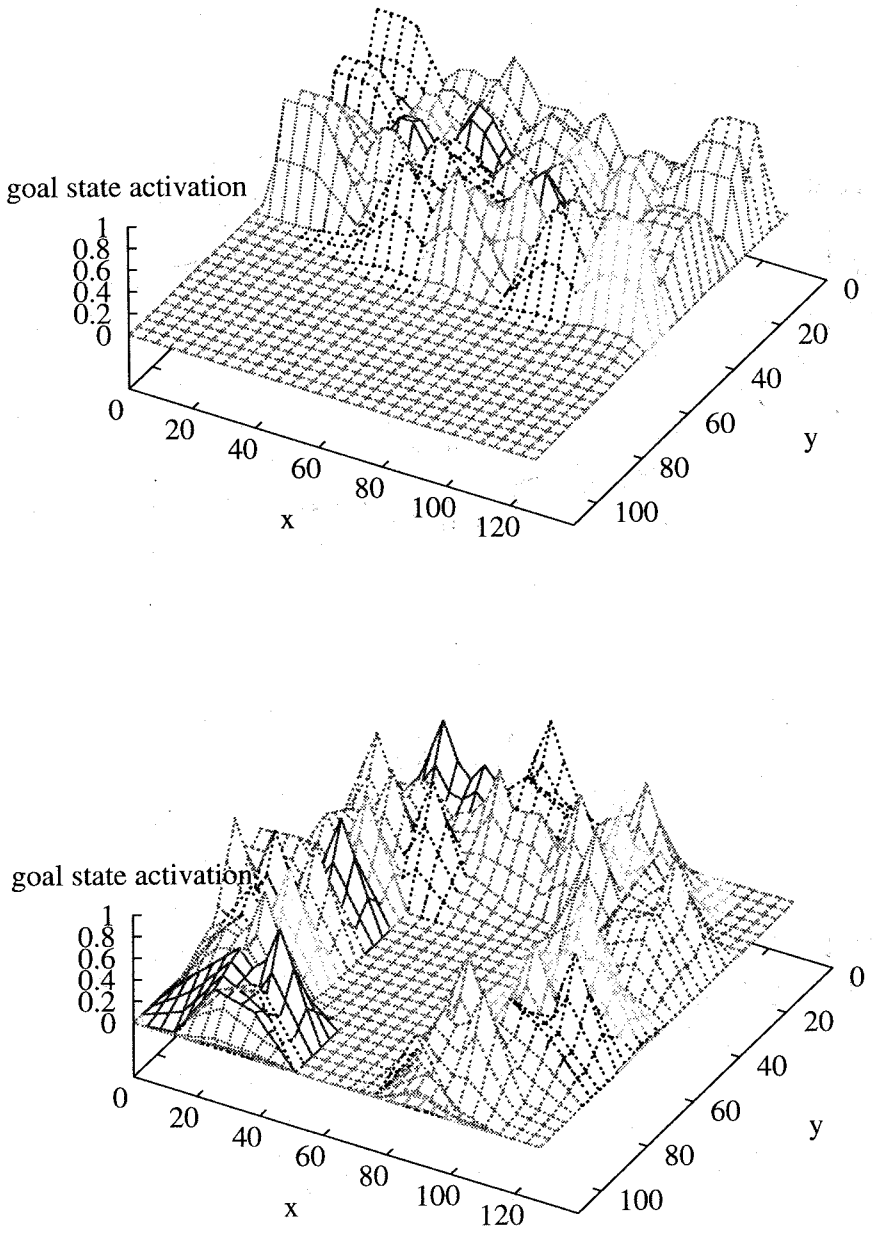


Figure 4.27: The distribution of learning modules at bottom layer (70000 step): TOP:perspective camera image, BOTTOM:omni-directional camera image

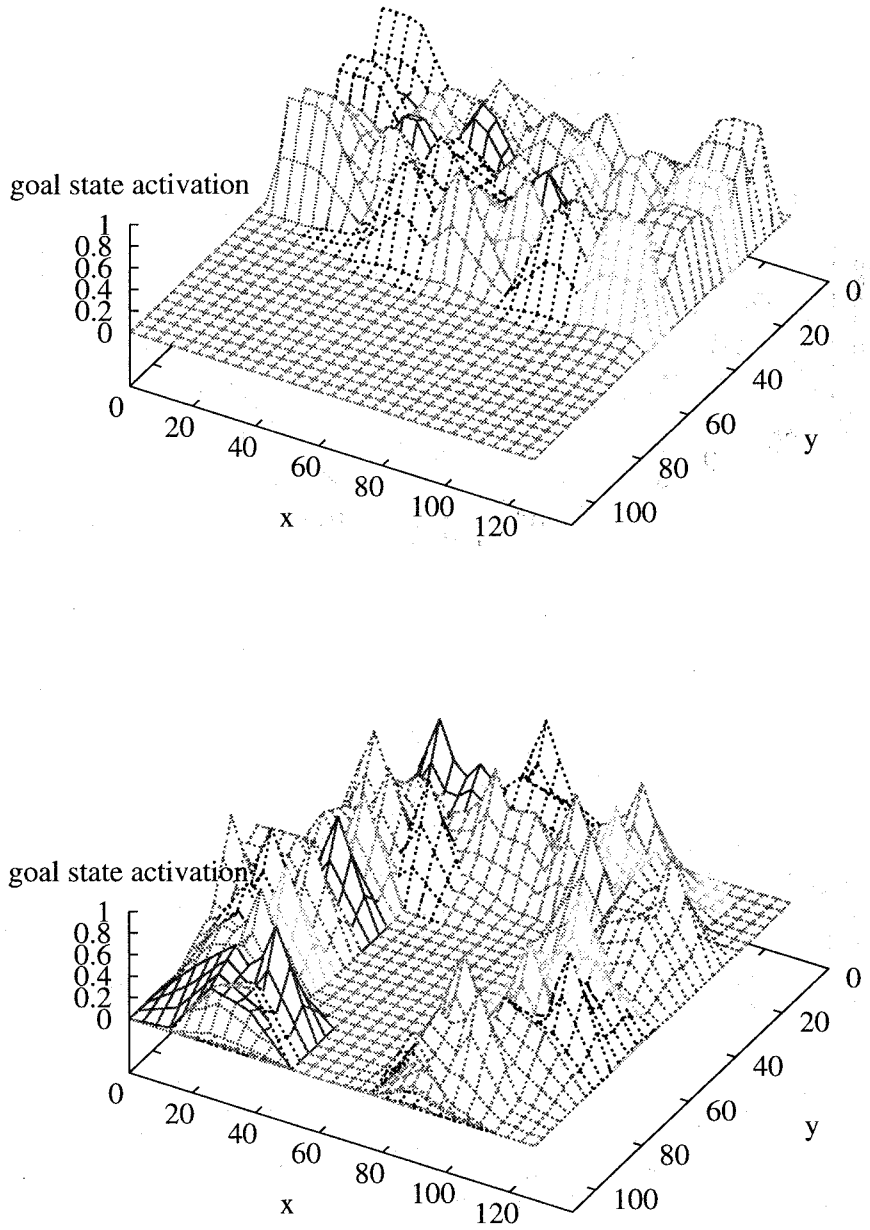


Figure 4.28: The distribution of learning modules at bottom layer (80000 step): TOP:perspective camera image, BOTTOM:omni-directional camera image

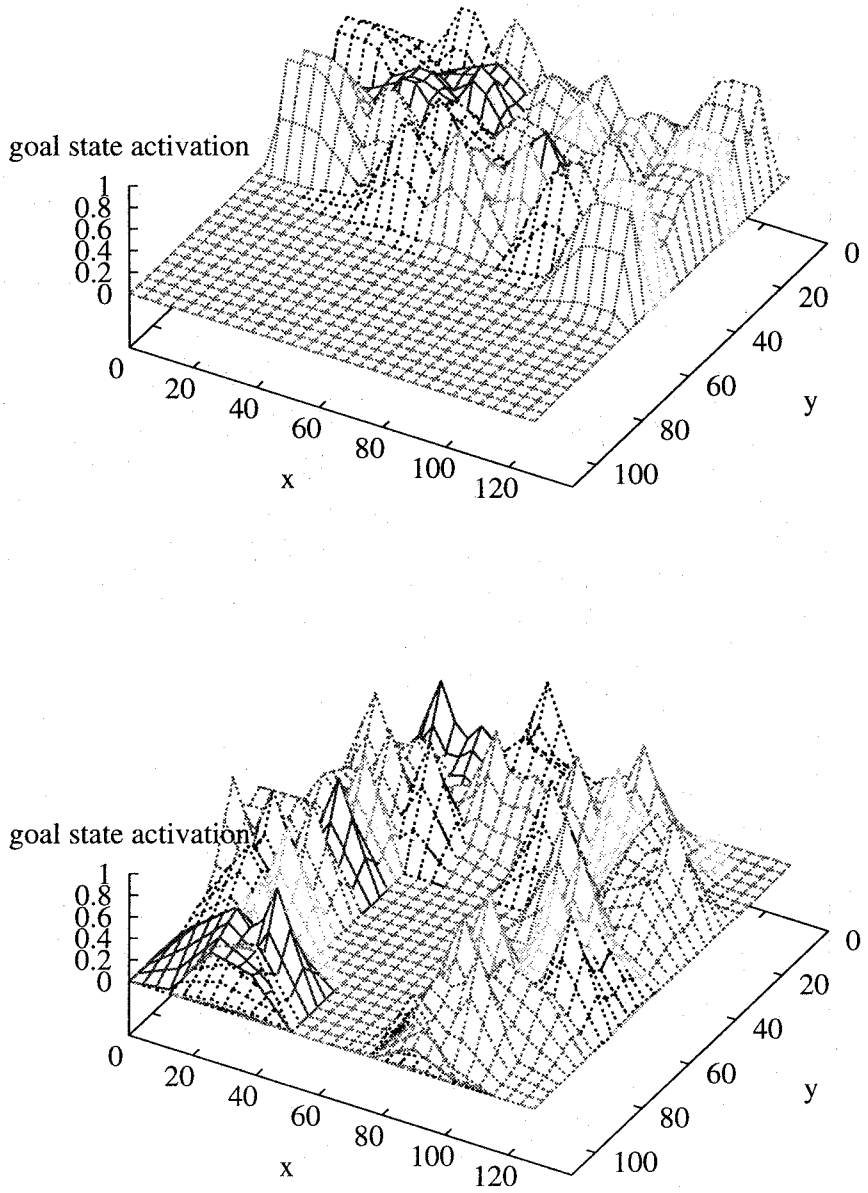


Figure 4.29: The distribution of learning modules at bottom layer (90000 step): TOP:perspective camera image, BOTTOM:omni-directional camera image

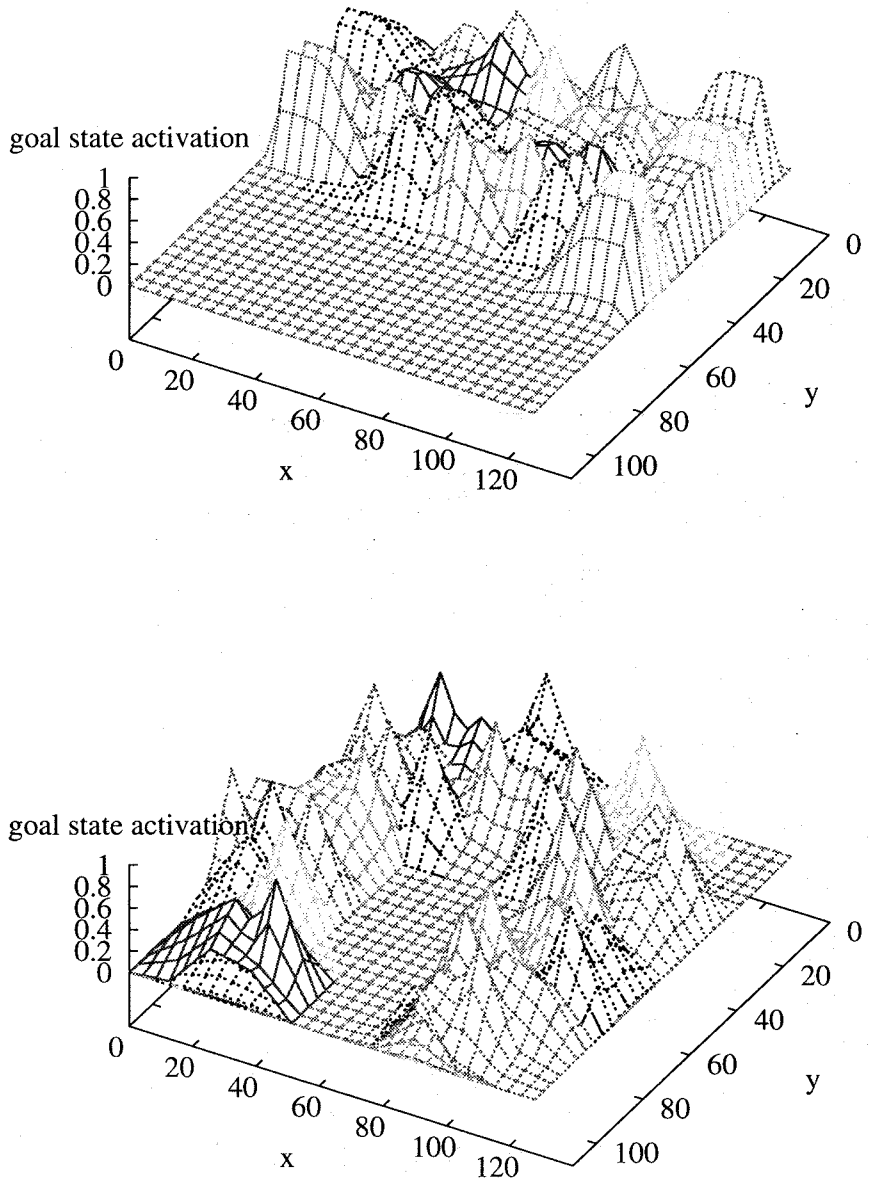


Figure 4.30: The distribution of learning modules at bottom layer (100000 step): TOP:perspective camera image, BOTTOM:omni-directional camera image

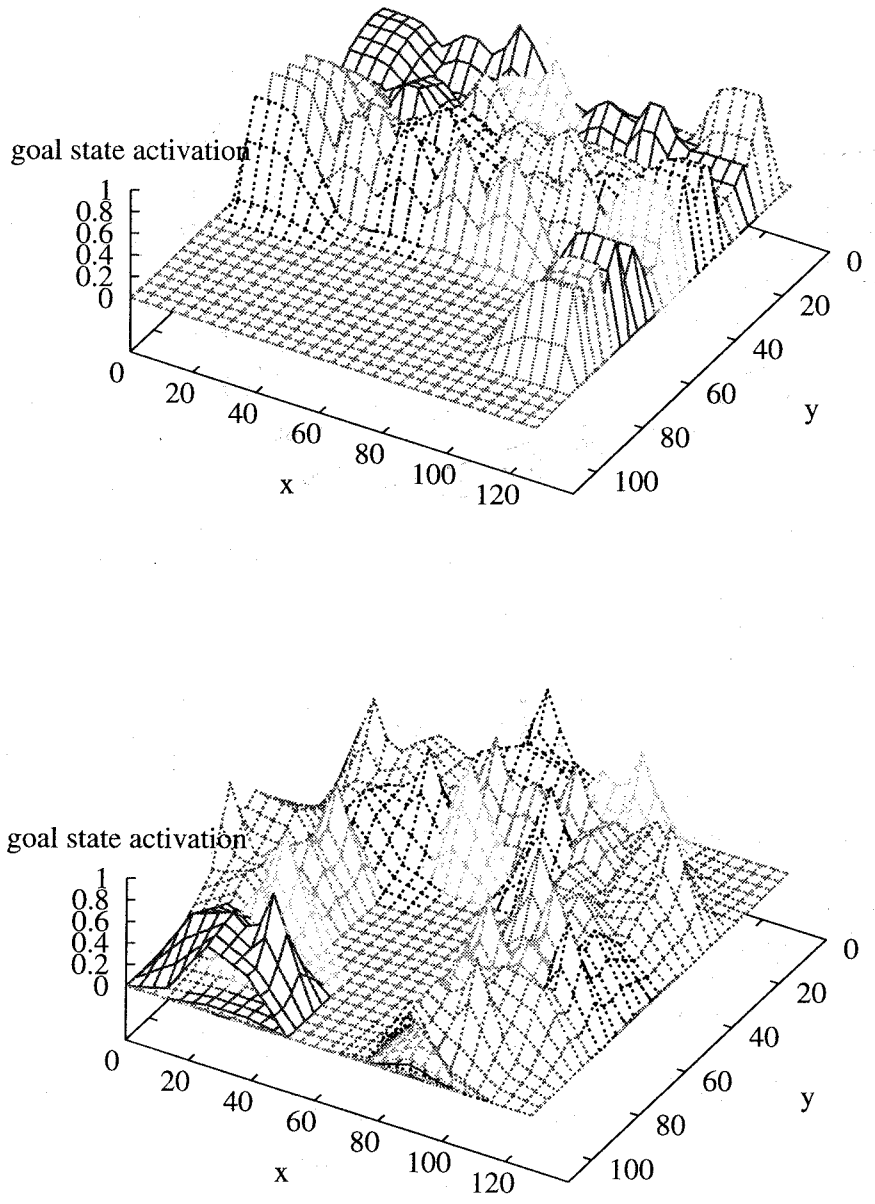


Figure 4.31: The distribution of learning modules at bottom layer (200000 step): TOP:perspective camera image, BOTTOM:omni-directional camera image

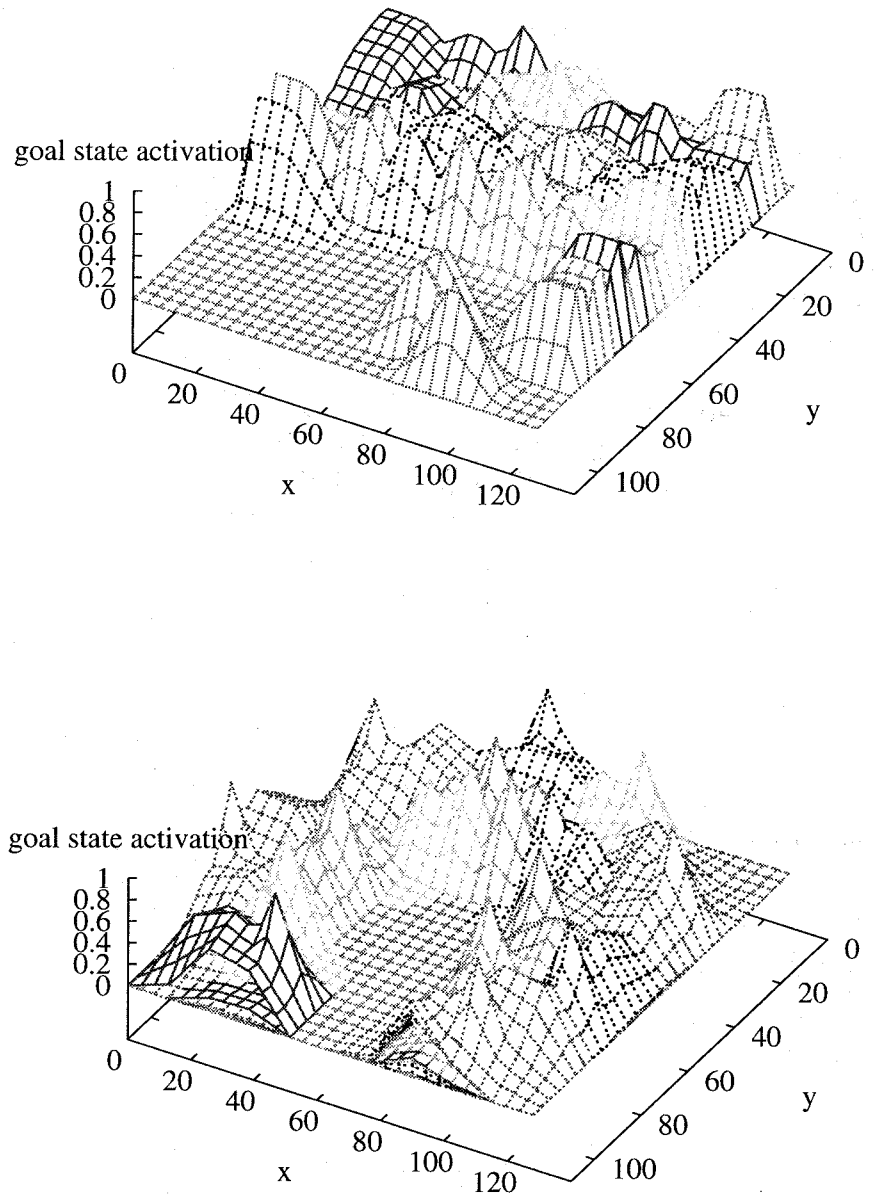


Figure 4.32: The distribution of learning modules at bottom layer (300000 step): TOP:perspective camera image, BOTTOM:omni-directional camera image

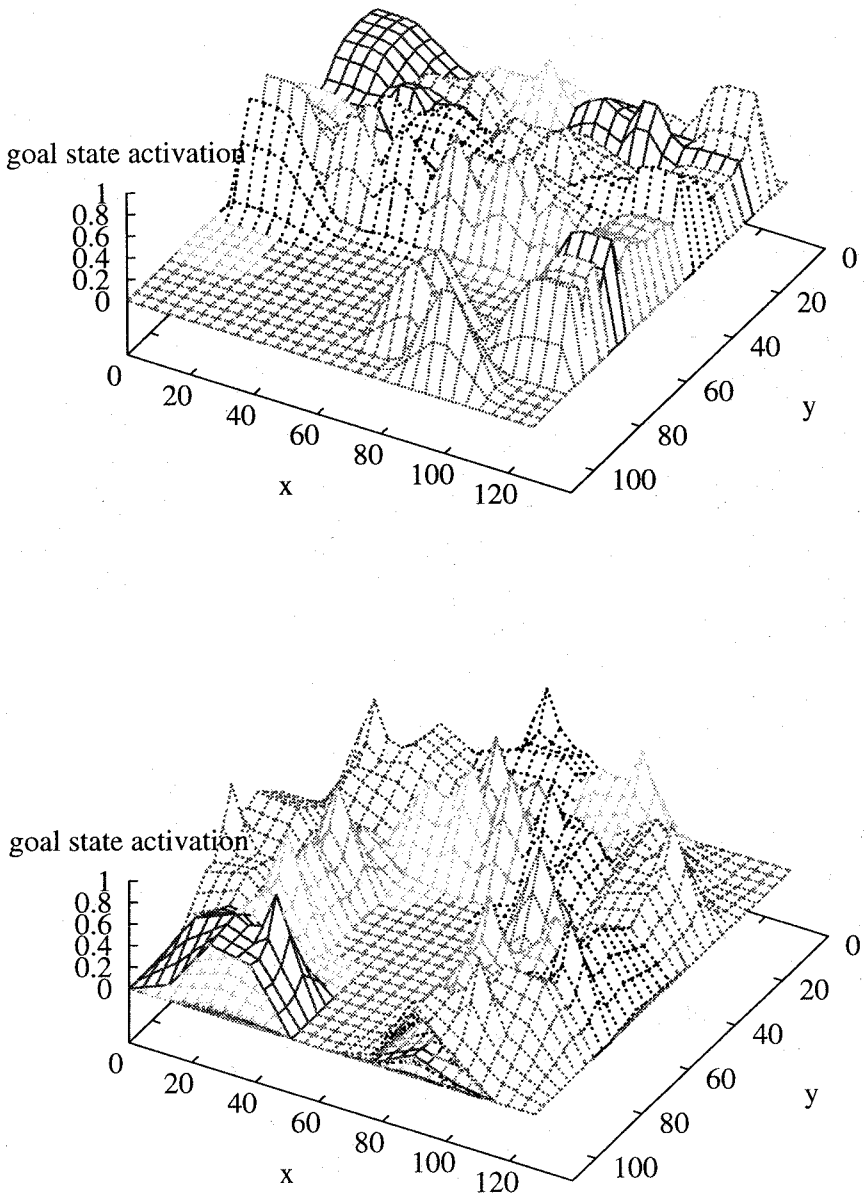


Figure 4.33: The distribution of learning modules at bottom layer (400000 step): TOP:perspective camera image, BOTTOM:omni-directional camera image

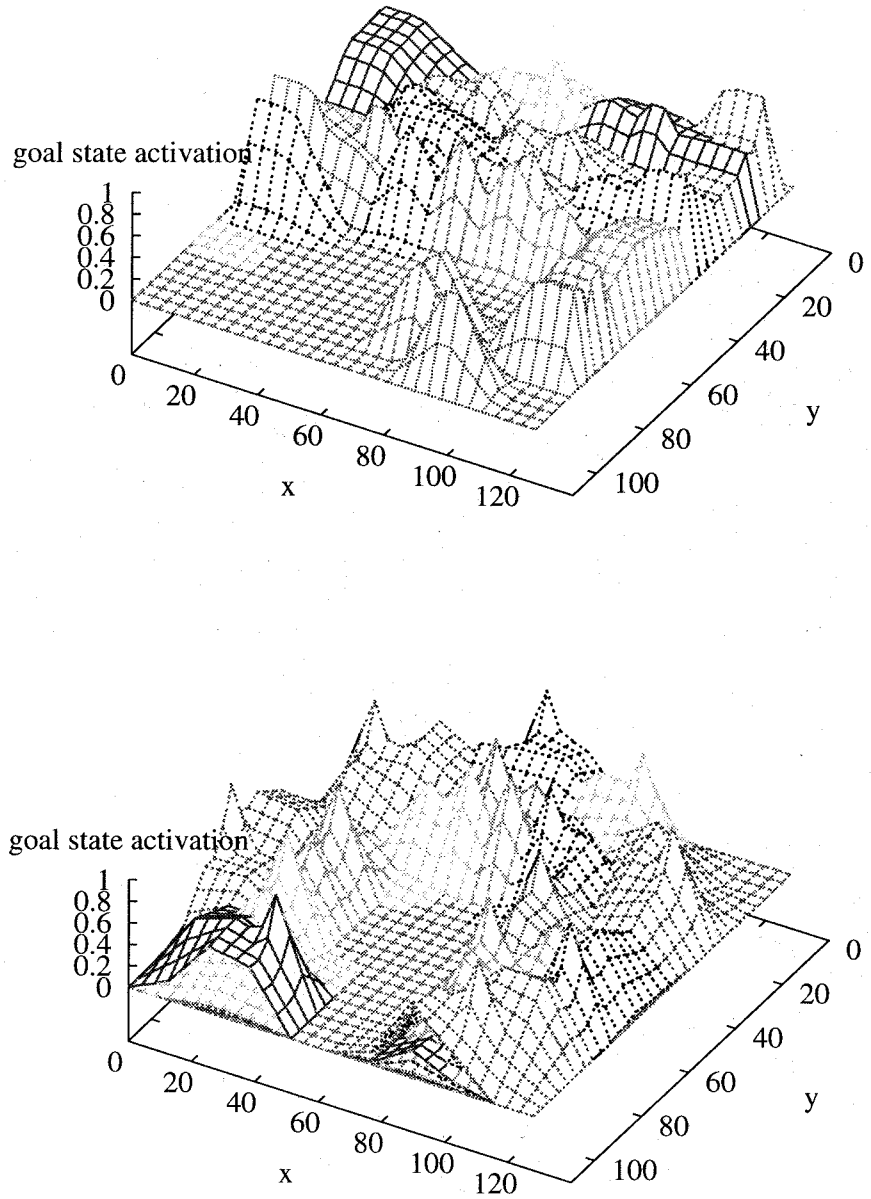


Figure 4.34: The distribution of learning modules at bottom layer (500000 step): TOP:perspective camera image, BOTTOM:omni-directional camera image

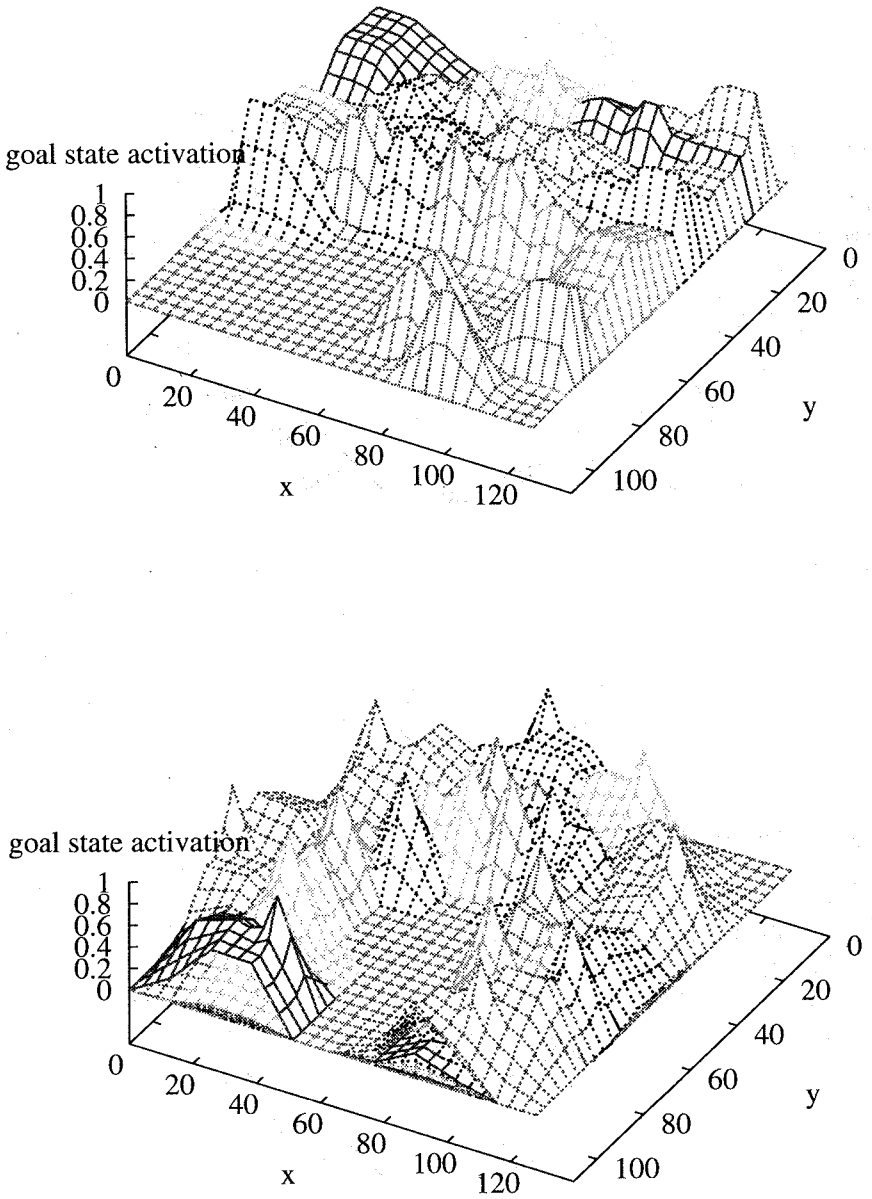


Figure 4.35: The distribution of learning modules at bottom layer (600000 step): TOP:perspective camera image, BOTTOM:omni-directional camera image

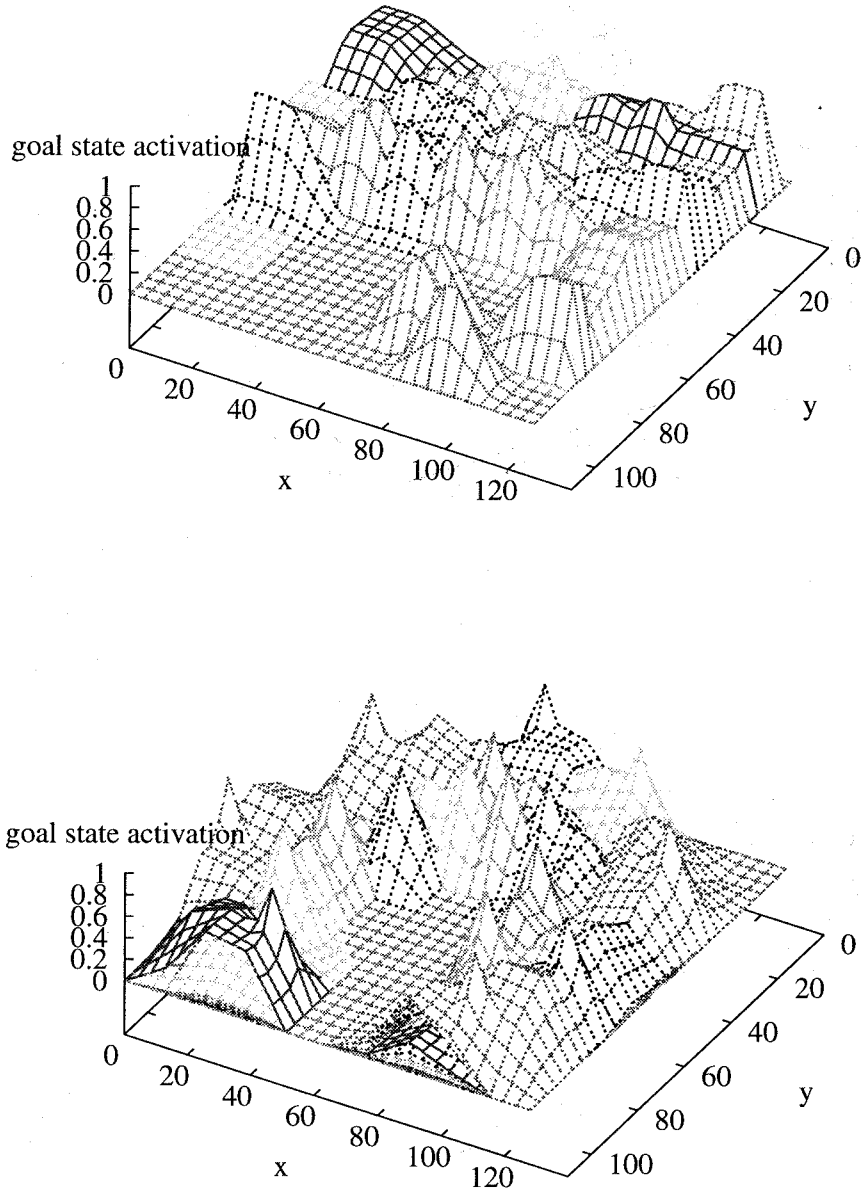


Figure 4.36: The distribution of learning modules at bottom layer (700000 step): TOP:perspective camera image, BOTTOM:omni-directional camera image

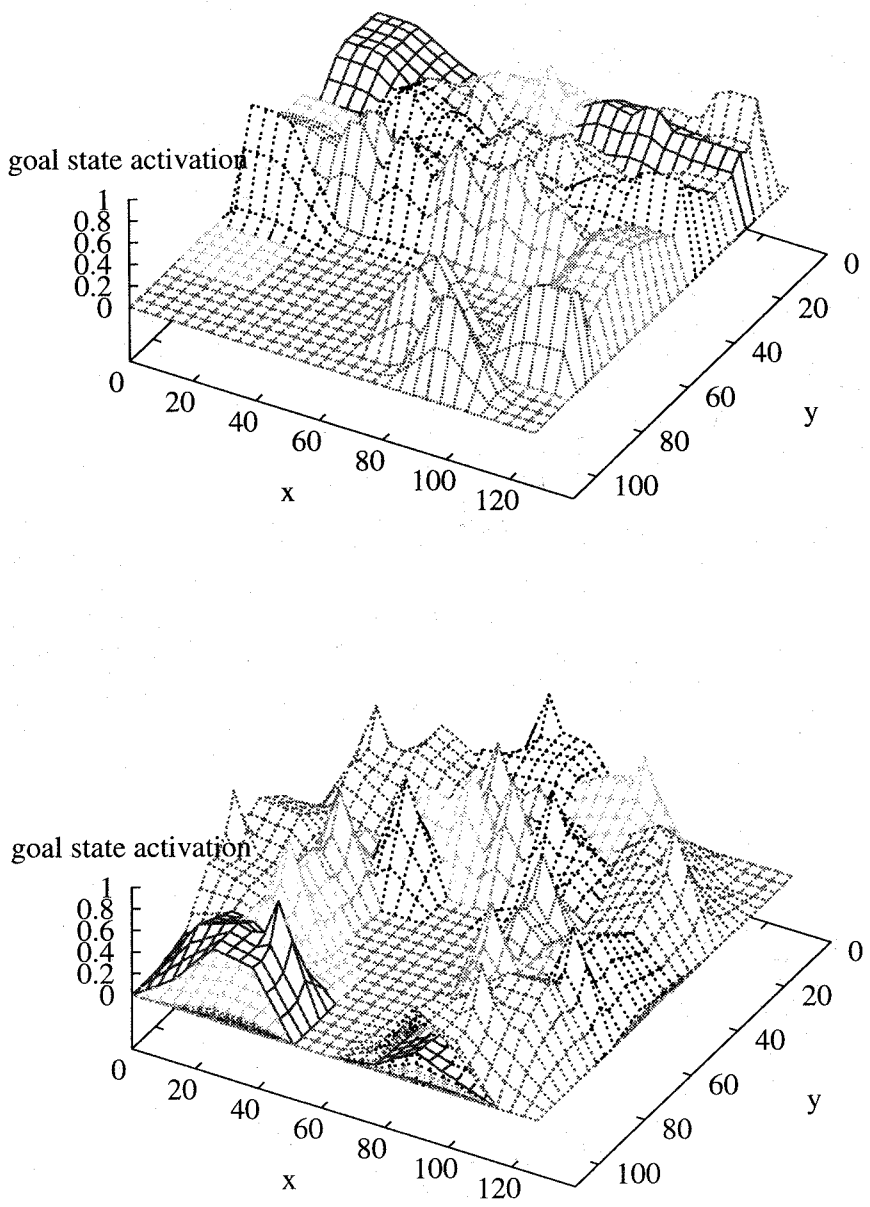


Figure 4.37: The distribution of learning modules at bottom layer (800000 step): TOP:perspective camera image, BOTTOM:omni-directional camera image

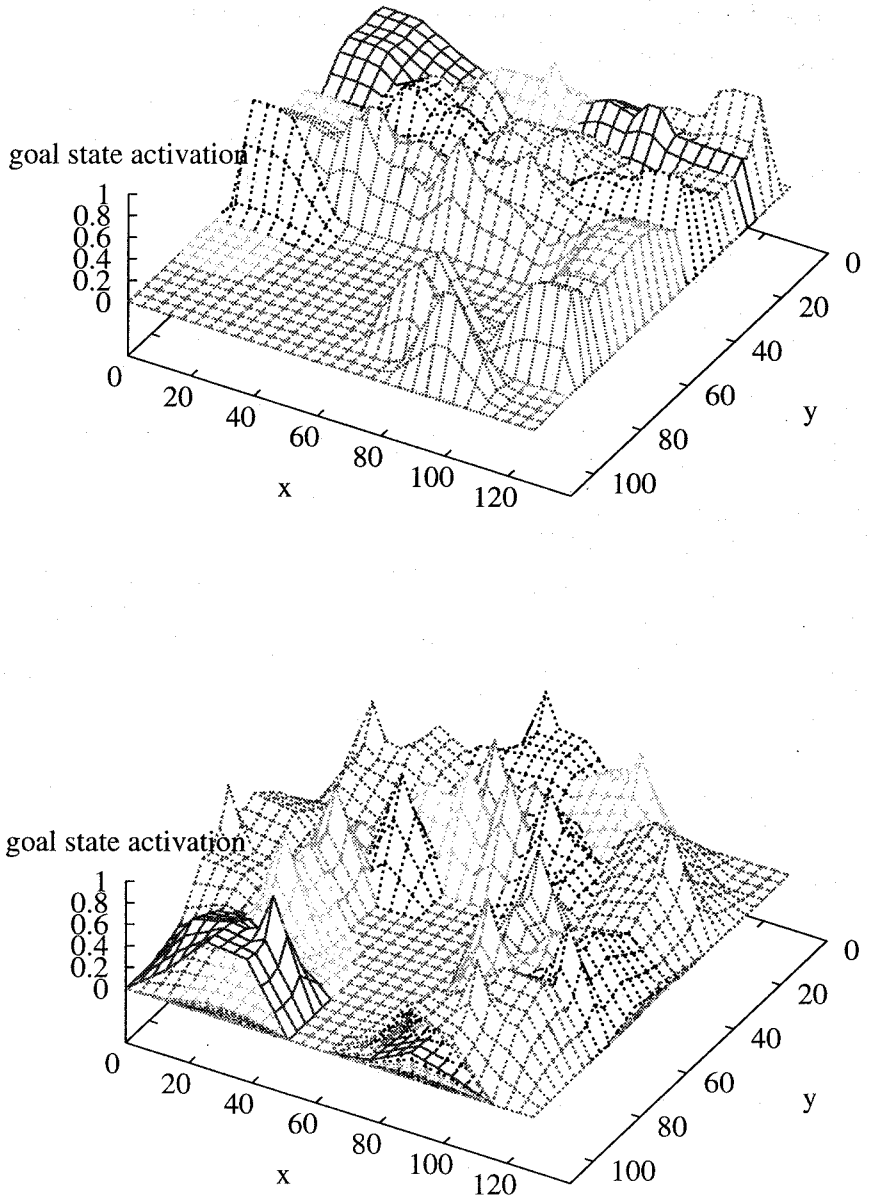


Figure 4.38: The distribution of learning modules at bottom layer (900000 step): TOP:perspective camera image, BOTTOM:omni-directional camera image

4.4 Conclusion

We proposed a novel method of multi-layered reinforcement learning system. The results show that the competitive learning enables to determine the subgoals or subtasks by the robot without human designers intervention, and that as a whole system the robot could show purposive behaviors.

The proposed approach should be able to reuse previously learned knowledge to the related tasks. And, this enables the robot to learn always new, more abstracted behaviors in newly encountered situations through all its life.

Chapter 5

State-Action Space Construction for Multi-Layered Learning System

5.1 Introduction

In this chapter, we propose an approach to the problem of decomposing the large state space at the bottom level into several subspaces and merge those subspaces at the higher level. This allows the system

- to acquire behaviors based on new state features by adding new learning modules layers based on them while it leaves the already learned hierarchical structure,

- to acquire behaviors based on state space constructed with a large number of state valuables by merging the modules at lower levels which acquire behaviors based on the subspaces, and

- to save computational resources because the number of state could be small by decomposing the whole state space into small subspaces.

We apply the method to a simple soccer situation in the context of RoboCup, show the experimental results, and give discussions.

5.2 Multi-Layered Learning System

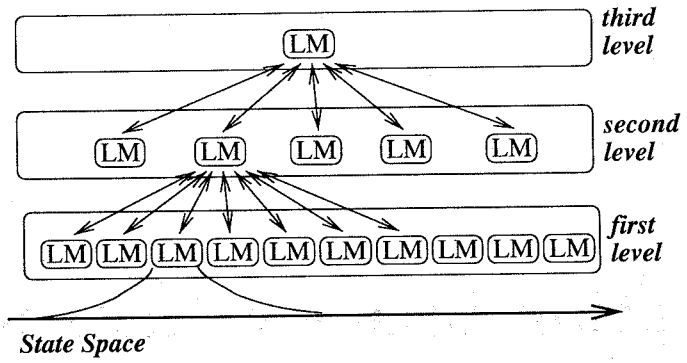


Figure 5.1: An overview of a hierarchical learning architecture (LM stands for learning module).

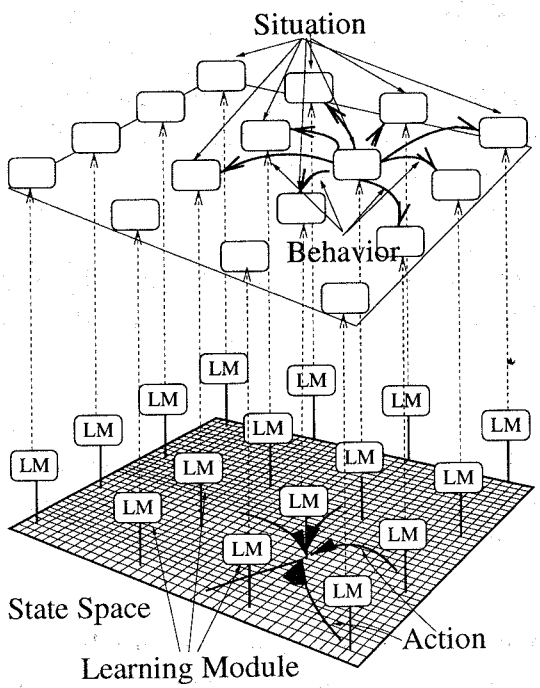


Figure 5.2: The relationships of situations and behaviors inside a layer and between different levels

The basic idea of multi-layered learning system is same as the previous chapter. The robot prepares learning modules of one kind, makes a layer with these modules, and constructs a hierarchy with the layers. The hierarchy of the learning module's layers can be regarded as a role of task decomposition. The lower learning modules explore small areas in the given environment, and learn lower level, fundamental behaviors. They learn behaviors with narrower scopes and shorter time horizons, focusing on the more details. On the other hand, the higher learning modules explore large areas, and learn higher level, more abstracted behaviors based on the learning modules at the lower layers.

This multi-layered learning system defines situations/behaviors based on the modules of lower layers, defines state and action spaces using them, and acquires new abstracted behaviors on them. Figure 5.2 shows a rough sketch of this idea. The system assigns learning modules on the state space of a certain layer. Each learning module acquires the behavior to reach its own goal specified on the state space. The another layer on it regards a region assigned to a lower learning module as a situation, and a motion to a close region as a behavior.

5.3 State-Action Space Construction based on Lower Learning Modules

When the higher layer constructs its state-action space based on situations and behaviors acquired by the modules of several lower layers, it should consider that the layers are independent from each other, or there is dependence between them. The layer might be basically independent from each other when the each layer's modules recognize a different object and learn behaviors for it. On the other hand, there might be dependence between the layers when modules on all layer recognize the same object in the environment and learn the behavior against it. For example, the system would regard that both layers are independent from each other if the modules on one layer acquire several navigation behaviors, and the module on the other layer acquire object manipulation behaviors; in the case of robot in the RoboCup field, one layer's modules could be experts for ball handling and the other layer's modules ones for navigation on the field. On the other hand, it will recognize that there is a certain relationship between the layers when the system captures a number of data which represent one certain object with different sensor devices. In such a case, the system can recognize the situation complementary using plural layers' outputs even if one layer loses the object on its own state spaces. Now, we propose "a multiplicative approach" for the former, and "a complementary approach" for the latter.

5.3.1 Multiplicative Approach

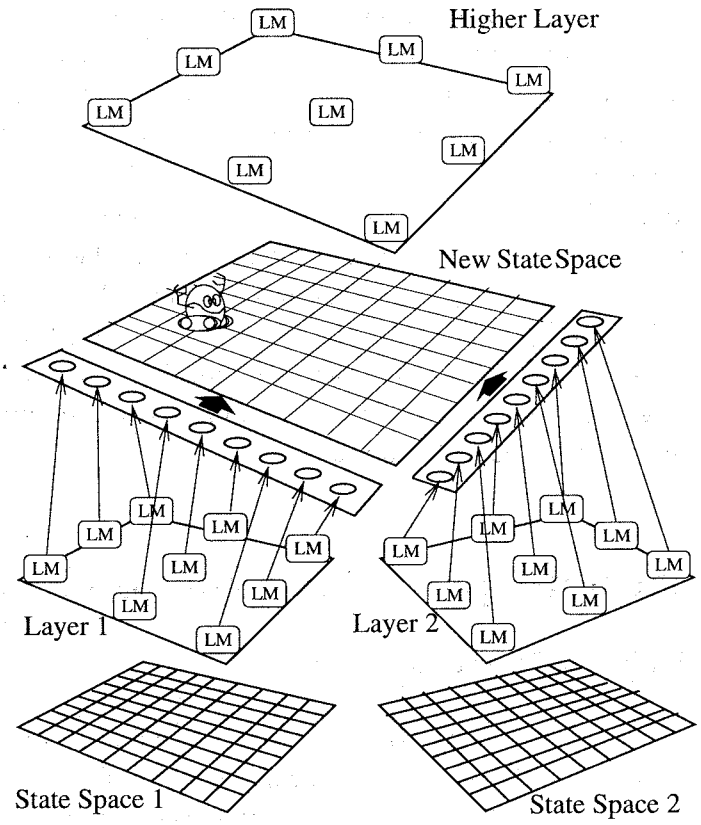


Figure 5.3: State-action space construction based on multiplicative approach

The state space is constructed as a direct product of module's activations of lower layers. This case occurs when the plural layers deal with different objects from each other. The higher modules recognize the lower module's outputs of layers as different one (In Figure 5.3, two layers are shown for the reader's understanding but generally it can be any number.). We design the state-action space for the modules managing two different layer's modules as shown in Figure 5.3. The system constructs an $n \times m$-dimensional state vector and an $n + m$-dimensional action vector if one lower layer has n modules and the other has m ones.

5.3.2 Complementary Approach

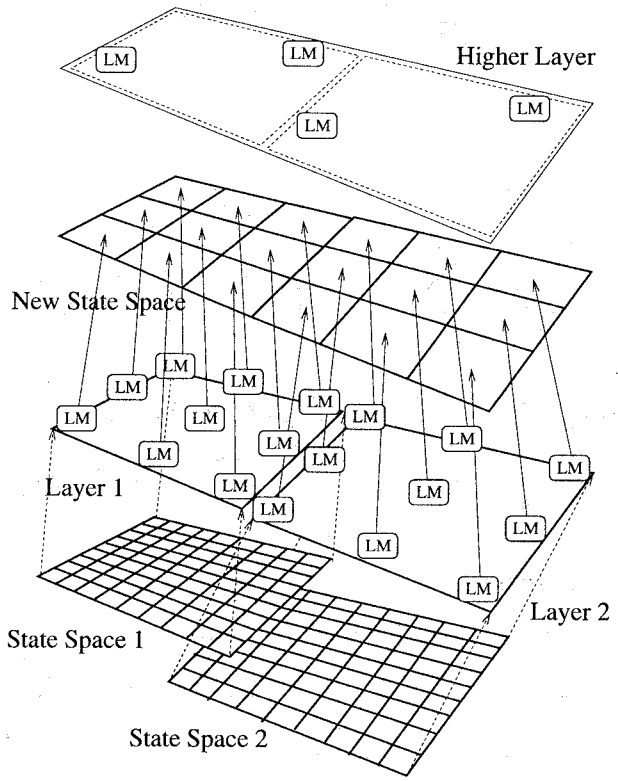
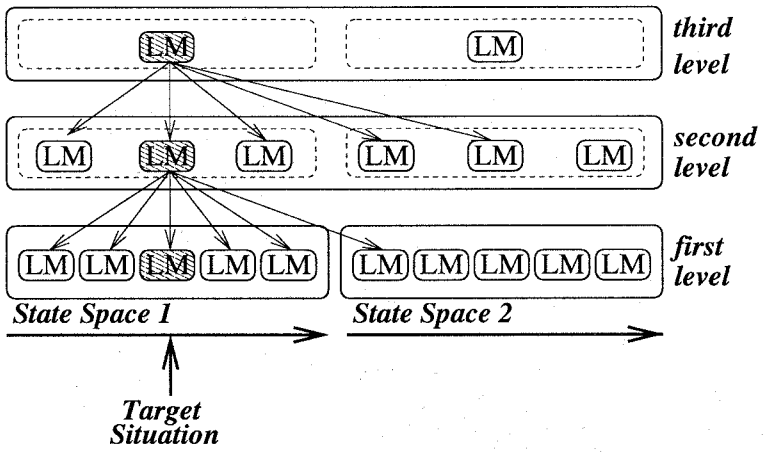


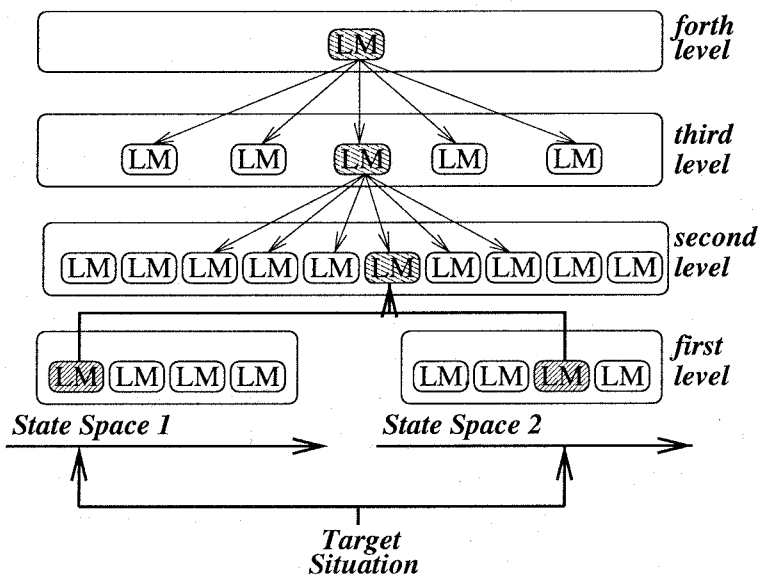
Figure 5.4: State-action space construction based on complementary approach

The state space is constructed in an additive manner (see Figure 5.4) when there is a correlation between activation patterns of lower layers. This approach seems to be an extension of the method described in the previous chapter. The output of one layer is assumed to be complementary to the other in this case; if the system acquires the output of one layer, it can predict the output of other one to some extent. The system does not need to recognize the outputs of two layers as different one, but as the same one output. We design the state-action space of modules which manages activations of two layers as shown in Figure 5.4. The system constructs an $n + m$-dimensional state vector and an $n + m$-dimensional action vector if one lower layer has n modules and the other m modules. This means that the modules which has its goal state on the output of one lower layer extrapolate the information from the other lower layer, if there is no active modules on the lower layer. This approach saves computational resources compared to the "multiplicative approach".

5.4 Strategy in the Multi-Layered Learning System to Accomplish A Task



(a) a target situation in one state space



(b) a target situation in two state spaces

Figure 5.5: The strategy in the multi-layered control structure

The basic idea of the strategy in the multi-layered system is same as describeed in the previous chapter. The target state is given to the multi-layered learning system in the state space at the bottom level. Figure 5.5(a) shows this situation. If the target situation is specified in one state space, the system executes the procedure described in the previous chapter. first of all, the system searches a module nearest to the target situation. If the module can accomplish the given task, it applies its optimal policy. Else, the system searches the module nearest to the target situation at the higher level. If the module can apply its policy, it sends behavior activation to its lower modules, else the system does the same way at the higher level.

① A target situation is specified in one state space at the bottom level.

② List up all layers which contain the space specified as the target situation and **constructed by the complementary approach**.

③ Find the lowest layer in the list.

④ Specify a target stat nearest to the target situation in the state space.

⑤ Find a module nearest to the target state.

⑥ Can the module executes its policy in the current situation?

YES : Activate the module to reach the target state.

NO : Find the target state on the higher layer in the list, then go to ④.

If the target situation is specified in two state spaces at the bottom level, the system searches the lowest layer which has modules managing them. Figure 5.5(b) shows this case. The target situation is given in the two different state spaces. The system searches a module nearest to the target situation at the second level though there are two activated modules at the first level because the given task is specified in the two state spaces and one layer at the second level manages them. If the module at the second level cannot handle the situation, the system searches a module at higher level as the proposed method in the previous chapter does.

① A target situation is specified in the state spaces at the bottom level.

② List up all layers which **contain all space specified as the target situation**.

③ Find the lowest layer in the list.

④ Specify a target state nearest to the target situation on the state space of the layer.

⑤ Find a module nearest to the target state.

⑥ Can the module executes its policy in the current situation?

YES : Activate the module to reach the target state.

NO : Find the target state on the higher layer in the list, then go to ④.

5.5 Simulation Experiments

5.5.1 An Overview

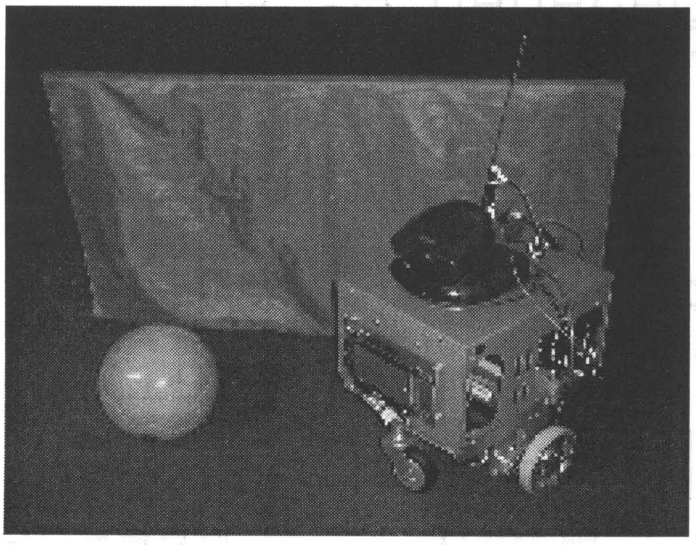


Figure 5.6: A mobile robot, a ball, and a goal

To evaluate the multiplicative approach, we apply it to a simple navigation task and shooting behavior in a computer simulation experiments. The environment consists of a ball, a goal, and a mobile robot. Figure 5.6 shows a picture of the mobile robot we designed and built, the ball, and the goal. The robot has a TV camera of which visual angles are 35 degrees and 30 degrees in horizontal and vertical directions, respectively. The camera is tilted down 23.5 degrees to capture the ball image as large as possible. The driving mechanism is a PWS (Power Wheeled System), and the action space is constructed in terms of two torque values to be sent to two motors that drive two wheels. These parameters of the system are unknown to the robot, and it tries to estimate the mapping from sensory information to appropriate motor commands by the method.

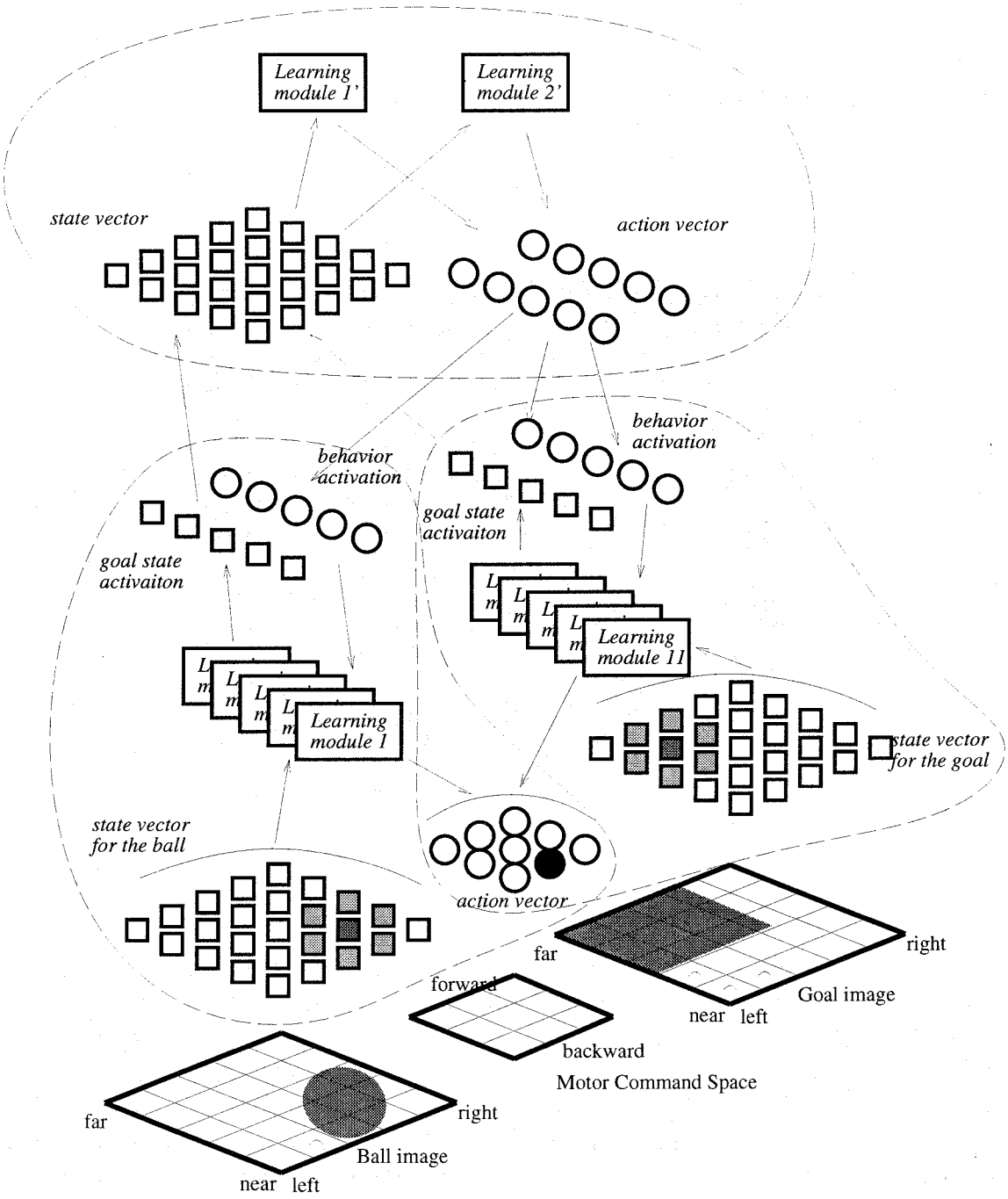


Figure 5.7: A hierarchical architecture of learning modules

Here, we introduce a two-level learning architecture, with one higher layer and two lower ones. The lower layers have 20 modules (10 for the ball and 10 for the goal), and the higher layer has 2 modules (Figure 5.7). The ball and the goal state spaces are constructed in terms of the centroid of the ball and the goal images, and tessellated both into 9 by 9 grids. The number of representative states for each module at the lower layer is 83; the combination of 9 by 9 of the centroid of the image, and "lost-into-left' and "lost-into-right" when the robot cannot capture the image of the ball or the goal. The number of states for each module at the higher layer is 100; the combination of 10 lower modules for the ball, and 10 lower modules for the goal. We assign tasks (reaching the ball and the goal, lose the ball and goal into the left side, and losing the ball and the goal into the right side) to several modules for stability of assignment system of other modules.

For the comparison between our proposed method and a standard monolithic learning method, we prepare two learning modules, one for reaching a ball and other for reaching a goal. The state space of the monolithic learning module is constructed as a combination of all ball situations and goal ones. The number of states is 6,889; the combination of 83 states for ball situations by 83 states for goal situations.

5.5.2 Results (1: Navigation)

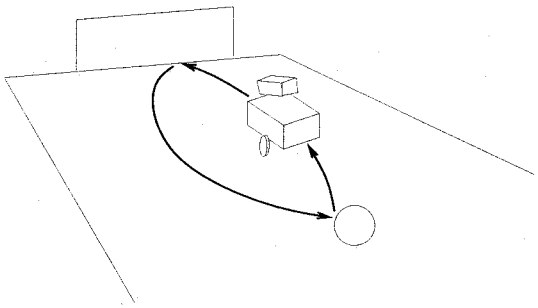


Figure 5.8: A simulation environment

The tasks for the robot are reaching a ball and reaching a goal. At first, we prepare a simple initial controller which forces the robot to make a round trip between a ball and a goal. The initial controller drives the robot as following: rotate the body until the the object comes to the center on the image, and drives it forward. At the beginning of the learning, the robot follows the initial controller. After having some experiences and learning the policy, it behaves by itself based on the learned policy.

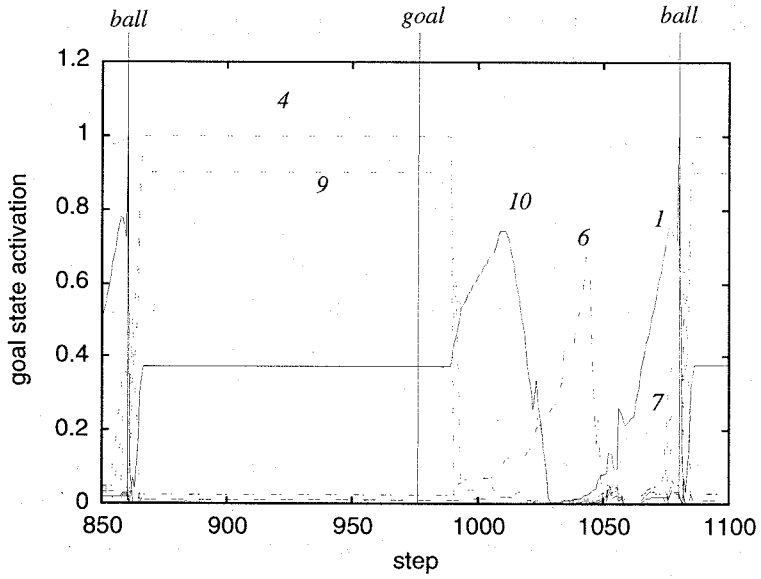


Figure 5.9: Goal state activation of modules at lower layer (ball) (navigation)

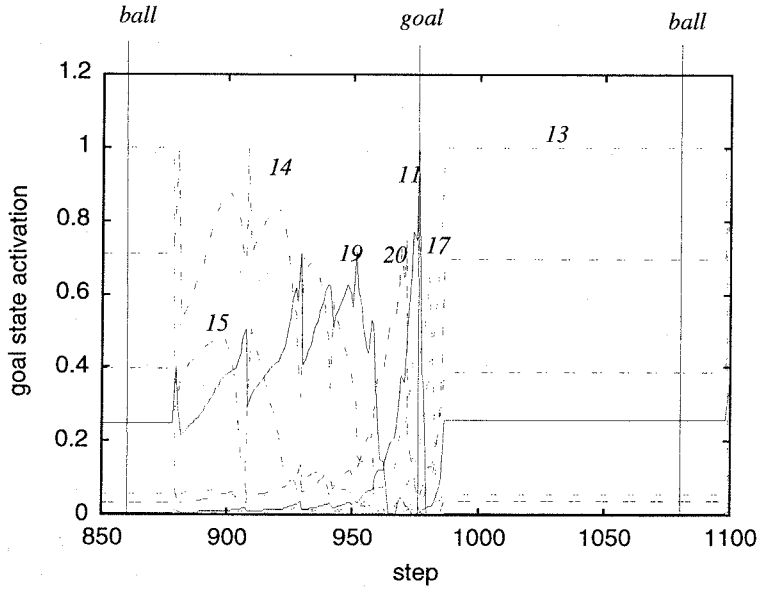


Figure 5.10: Goal state activation of modules at lower layer (goal) (navigation)

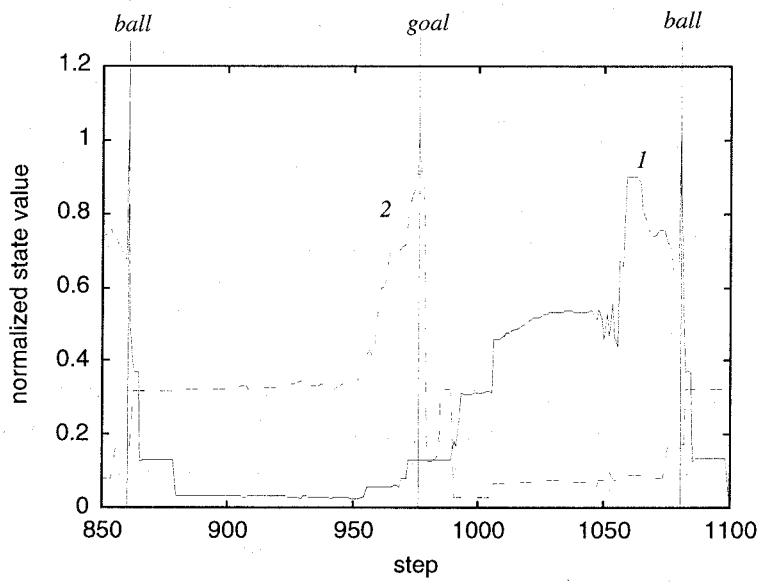


Figure 5.11: Goal state activation of modules at higher layer (navigation)

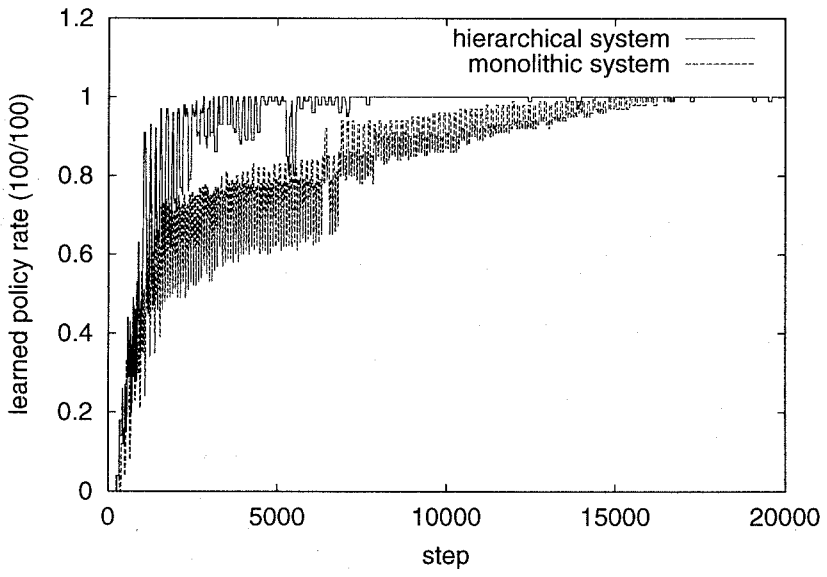


Figure 5.12: Learned policy following rate (navigation)

Figures 5.9 and 5.10 show the time development of the goal state activations of the learning modules at the lower layers of the ball navigation and the goal one, respectively. The robot goes from the ball to the goal in the period during the 860 th and the 975 th steps, and goes to the ball again in the period during the 975 th and the 1,080 th steps. We can see each learning module assigned its own goal state and has high activation when the situation is near to the goal state. For example, module 1 assigned its goal state to the situation nearest to the ball, and module 11 assigned its goal state to the situation nearest to the goal. The others assigned their goal states to other situations.

Figure 5.11 shows the time development of the goal state activations of both learning modules at the higher layer. Figure 5.12 shows the time development of the learned policy following rate. The learned policy following rate means the rate of action which follows the learned policy in the last 100 steps. The robot used the initial controller at the beginning of the learning, and it switched its controller from the initial one to the learned one. The hierarchical learning system obtained the valid policy much faster than the monolithic system.

5.5.3 Result (2: Shooting Behavior)

Next, the task for the robot is to shoot a ball into the goal. Figures 5.13 and 5.14 show the time development of the normalized state values of learning modules at the lower layer. When the simulation started, the robot went to the ball and kicked the ball at the 58th step. Then, it followed the ball after that and shot the ball into the goal at the 105th step.

Figure 5.15 shows the time development of the following rates of the learned policy . There are three lines in the figure; they are the rates of hierarchical system with experience, hierarchical system without experience, and monolithic system, respectively. The experience means that the hierarchical system reused the lower modules which have been obtained in the learning time of the simple navigation task **5.5.2**. The figure shows that the hierarchical learning system obtained the valid policy much faster than the monolithic system, and the system obtained the valid policy faster than the other hierarchical system without experiences.

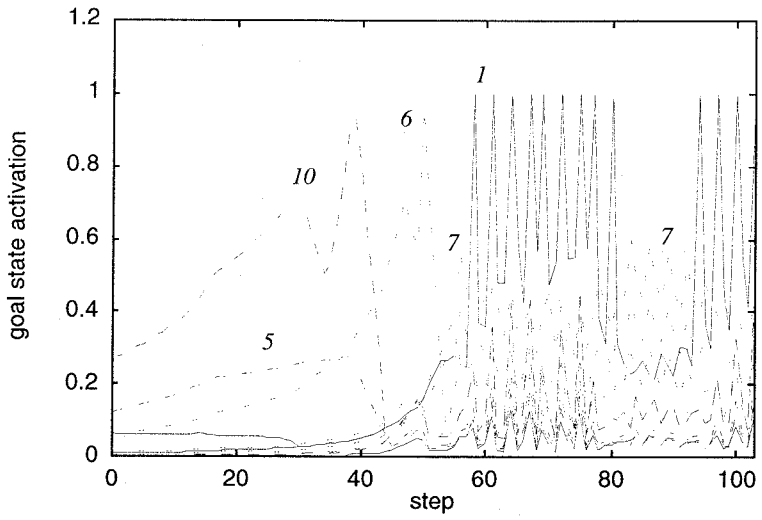


Figure 5.13: Goal state activation of modules at lower layer (ball) (shooting)

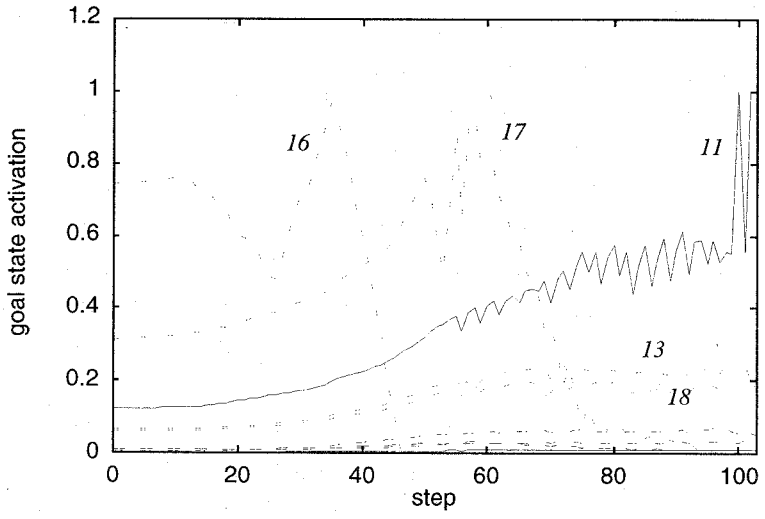


Figure 5.14: Goal state activation of modules at lower layer (goal) (shooting)

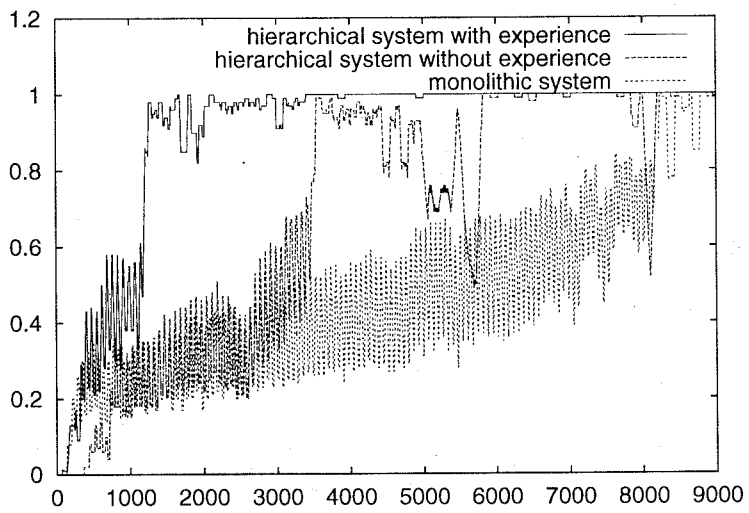


Figure 5.15: The following rate of the learned policy (shooting)

5.6 Real Robot Experiments

5.6.1 An Overview

To evaluate the proposed method, we apply it to a task of shooting a ball into a goal. Figure 5.16 shows a picture of the mobile robot we designed and built, the ball, and the goal. Figure 5.17 shows an overview of the robot system. It has two TV cameras; One with wide-angle lens of which visual angles are 35 degrees and 30 degrees in horizontal and vertical directions, respectively. This camera is tilted down 23.5 degrees to capture the ball image as large as possible. The other with omni-directional mirror is mounted on the robot. The driving mechanism is a PWS (Power Wheeled System), and the action space is constructed in terms of two torque values to be sent to two motors that drive two wheels. These camera and kinematic parameters of the system are unknown to the robot, and it tries to estimate the mapping from sensory information to appropriate motor commands by the method. The environment consists of a ball, a goal, and the mobile robot. The target situation is given by reading the sensor information when the robot pushes the ball into the goal; the robot captures the ball and the goal at the center bottom of the perspective camera image.

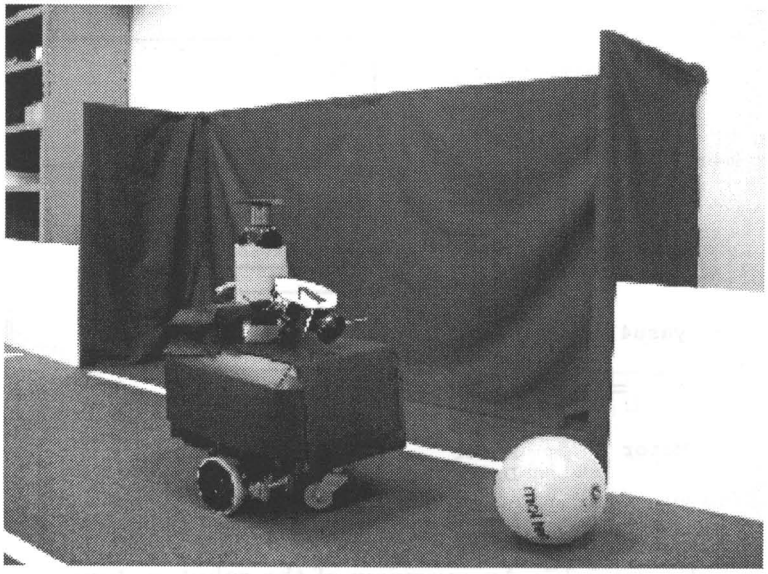


Figure 5.16: A mobile robot, a ball and a goal

The state spaces at the bottom layers are constructed in terms of the centroids of a ball and a goal images of the two cameras and the perspective image and omni-directional one are tessellated into 11 by 21 grids and 15 by 15 grids, respectively. The action space is constructed in terms of two torque values and is tessellated into 5 by 5 grids. The representative state and action at the higher layer is constructed by the learning modules automatically assigned at the lower layer.

We construct the hierarchical structure as shown in Figure 5.18. At the lowest level, there are four learning layers, and each of them deals with its own logical sensory space (ball positions on the perspective camera image and omni one, and goal position on both images). At the second level, there are three learning layers in which one adopts the multiplicative approach and the others adopt the complementary approach. The multiplicative approach of the "*ball pers.*$\times$*goal pers.*" layer deals with lower modules of "*ball pers.*" and "*goal pers.*" layers. The arrows in the figure indicate the flows from the goal state activations to the state vectors. The arrows from the action vectors to behavior activations are eliminated. At the third level, the system has three learning layer in which one adopts the multiplicative approach and the others adopt the complementary approach, again. At the levels higher than third layer, the learning layer is constructed as described in the previous chapter.

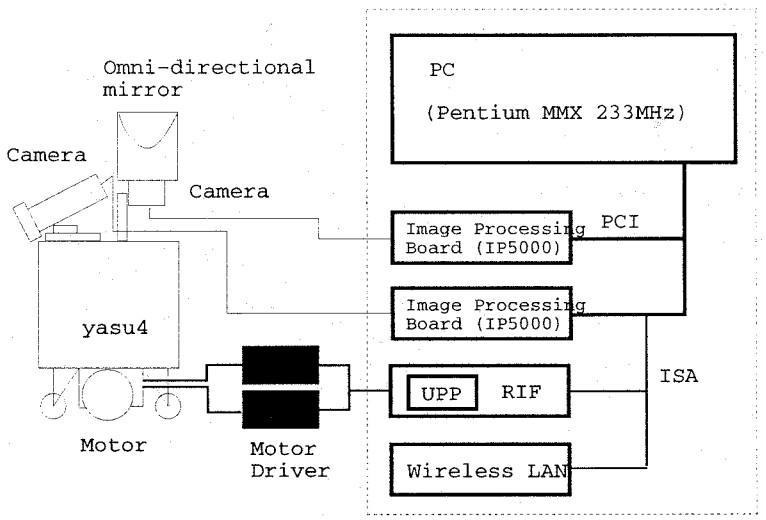


Figure 5.17: An overview of the robot system

5.6.2 Results

The experiment is constructed with two stages, one is the learning stage and other the task execution one using the learned policies. First of all, the robot moved at random in the environment for about two hours.

After the learning stage, we let our robot do a couple of tasks. One is shooting a ball into the goal using this multi-layer learning structure. The target situation is given by reading the sensor information when the robot pushes the ball into the goal; the robot captures the ball and goal at center bottom in the perspective camera image. As an initial position, the robot is located far from the goal, faced opposite direction to it . The ball was located between the robot and the goal.

Figure 5.20 shows the time development of the goal state and behavior activations of learning modules at the first, the second, and the third levels while the robot shoots the ball into the goal. The arrows on the top of each series indicate the behavior activations, and the others indicate the goal state activation. Figure 5.21 shows the sequence of the behavior activation of learning modules and the commands to the lower layer modules. The down arrows indicate that the higher learning modules fire the behavior activations of the lower learning modules.

When the robot located at the initial position, the module 8 in the "ball omni" layer and the module 12 in the "goal omni" layer at the first level has high goal state activations. "ball pers." layer and "goal pers." layer at the first level have no

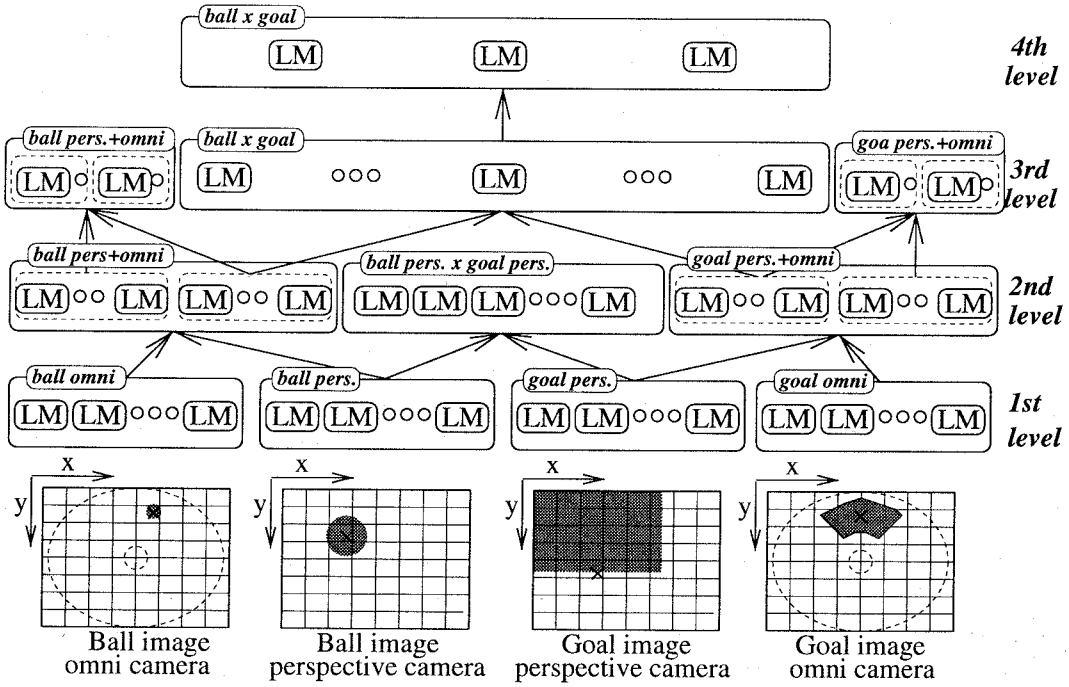


Figure 5.18: A hierarchy architecture of learning modules

activated modules because the robot does not capture the ball or the goal with the perspective camera. When the robot located at the target position, the module 0 at all layers are near to their own goal states. The robot turns its body until 20 steps in order to capture the ball and goal with the perspective camera, and it dribbles the ball, then finally shoots it into the goal.

Figure 5.21 shows the transition of the activated modules and the commands to the lower layer on the multi-layer learning system. First of all, the system tried to activate the module 0 of "*ball pers.*×*goal pers.*" layer at the second level, however, the module could not manage the current situation because the robot doesn't capture the ball and the goal with the perspective camera. Then, the system tried to activate the module 0 of "*ball*×*goal*" layer at the third level, and this module activates the module 1 of "*goal pers.+omni*" layer and the module 0 of "*ball pers.+omni*" layer at the second level, sequentially. These modules at the second level activate adequate modules at the first level. When the module 0 of "*ball pers.*×*goal pers.*" layer at the second level is able to handle the situation, the module takes over all control of the robot. Sometimes the module 0 of "*ball*×*goal*" layer at the third level is activated when the module "0" of "*ball pers.*×*goal pers.*" layer at the second level cannot

handle the situation because the robot bumped the ball and the situation changed drastically. Finally, the module 0 of "*ball pers.*×*goal pers.*" layer at the second level leads the robot to the target situation in which the robot is capturing the ball and the goal at center bottom of the perspective camera image.

5.6.3 Discussion

We verify the three advantages of multi-layered learning system which are described in **5.1.**.

① The system has a capability of behavior acquisition based on new state features by adding new learning modules layers based on them while it leaves the already learned hierarchical structure when it encounters new state features

We prepared the hierarchical structure as shown in Figure 5.18, here. In the case the robot perceives only a ball in the environment, the system trains the learning modules on the left side of the hierarchical structure shown in Figure 5.18; the "*ball pers.*" and "*ball omni*" layers at the first level, the "*ball pers.+omni*" layer at the second level, and the "*ball pers.+omni*" layer at the third level. Then, the system acquires a behavior of the ball reaching.

On the other hand, in the case the robot perceives goals in the environment, the system adds and trains the learning modules on the right side of the hierarchical structure shown in Figure 5.18; the "*goal pers.*" and "*goal omni*" layers at the first level, the "*goal pers.+omni*" layer at the second level, and the "*goal pers.+omni*" layer at the third level. Then, the system acquires several behaviors of the navigation based on the goals.

② The system acquires behaviors based on state space constructed with a large number of state valuables by merging the modules at the lower levels which acquire behaviors based on subspaces.

While there is no module which handles both the ball and the goals at the fist level, "*ball pers.*×*goal pers*" layer at the second level, "*ball*×*goal*" layer at the third level, and "*ball*×*goal*" layer at the fourth level can handle the both simultaneously, and acquired the shooting behavior through its experiments.

In the **5.5**, we have shown that the system with already acquired ball and goal reaching behavior learns the shooting behavior much more efficiently than the system which begins the learning from a scratch.

③ The system saves computational resources because the number of state could be small by decomposing the whole state space into small

subspaces

Table 5.1: Required memory size for the proposed layered learning system

level	layer	# of states A	# of actions B	Q table size $C = A \times B$	# of modules D	memory size $C \times D$
1	ball pers.	231	25	5,775	25	144,375
	ball omni.	225	25	5,625	18	101,250
	goal pers.	231	25	5,775	24	138,600
	ball omni.	225	25	5,625	23	129,375
2	ball pers.+omni.	43	43	1,849	9	16,641
	goal pers.+omni.	47	47	2,209	9	19,881
	ball pers. × goal pers.	600	49	29,400	65	1,911,000
3	ball pers.+omni.	9	9	81	1	81
	goal pers.+omni.	9	9	81	1	81
	ball × goal	81	18	1,458	15	21,870
4	ball × goal	15	15	225	8	1,800
				sum total	193	2,465,073

Table 5.2: Required memory size for the layered learning system with monolithic state and action spaces

level	layer	# of states A	# of actions B	Q table size $C = A \times B$	# of modules D	memory size $C \times D$
1	full state-action	207,936	25	5,198,400	10^4	10^{11}
2	full state-action	10^4	10^4	10^8	10^3	10^{11}
3	full state-action	10^3	10^3	10^6	10^2	10^8
...	full state-action	...	...	...	...	...
				total	10^4	10^{11}

We show that the proposed multi-layered system needs much less memory resources than the conventional monolithic one. Table 5.1 shows the number of states, the number of action, the size of Q-table, the number of assigned modules, and the requisite memory size for the layer when our method is applied. Table 5.2 shows the expected data for the conventional monolithic system. The data indicateds approximated numbers because it is difficult to implement the monolithic system to the real robot.

Our proposed method needs only several hundreds orders of states, and several hundred thousands order of memory for all learning modules at the first level because the state space is divided into several subspaces. On the other hand, the system

with the monolithic state-action space needs hundred thousands order of states, and the one learning modules needs several millions oder of computational memory. We assume that the system would assign about 1/10 learning modules than the number of states. The table shows that both systems need much computational resources at the second level. The total orders of memory size in this experiments are 10^6 and 10^{11} for proposed method and the conventional approach, respectively. The required computational resources reduced drastically.

5.7 Discussions

We proposed a mechanism which constructs multiple configurations of learning modules at higher layers using a number of groups of modules at lower layers. We applied the method to a simple soccer situation in the context of RoboCup, showed the experimental results.

The current system needs the learning layer construction by the programmer. Further, the current method has focused on the state space hierarchy, but the idea of hierarchy construction seems applicable to the action space hierarchy, too. These are our future works.

When the system applies the multiplicative approach to merge two state spaces of which state transitions may depend on each other, an interference problem would occur. For example, there are two lower layers which are basically independent of each other. The modules at the higher layer with state space constructed by multiplicative approach with these two layers, activates one module from both lower layers. If the higher modules activates one module of a certain lower layer, the behavior of the activated module might influence the situations of the other lower layer. The higher modules would not be able to acquire purposive behaviors if the interference occurs frequently. If the resolution of state and action spaces constructed by multiplicative approach are fine, the reinforcement learning method can find the indirect route to the target situation.

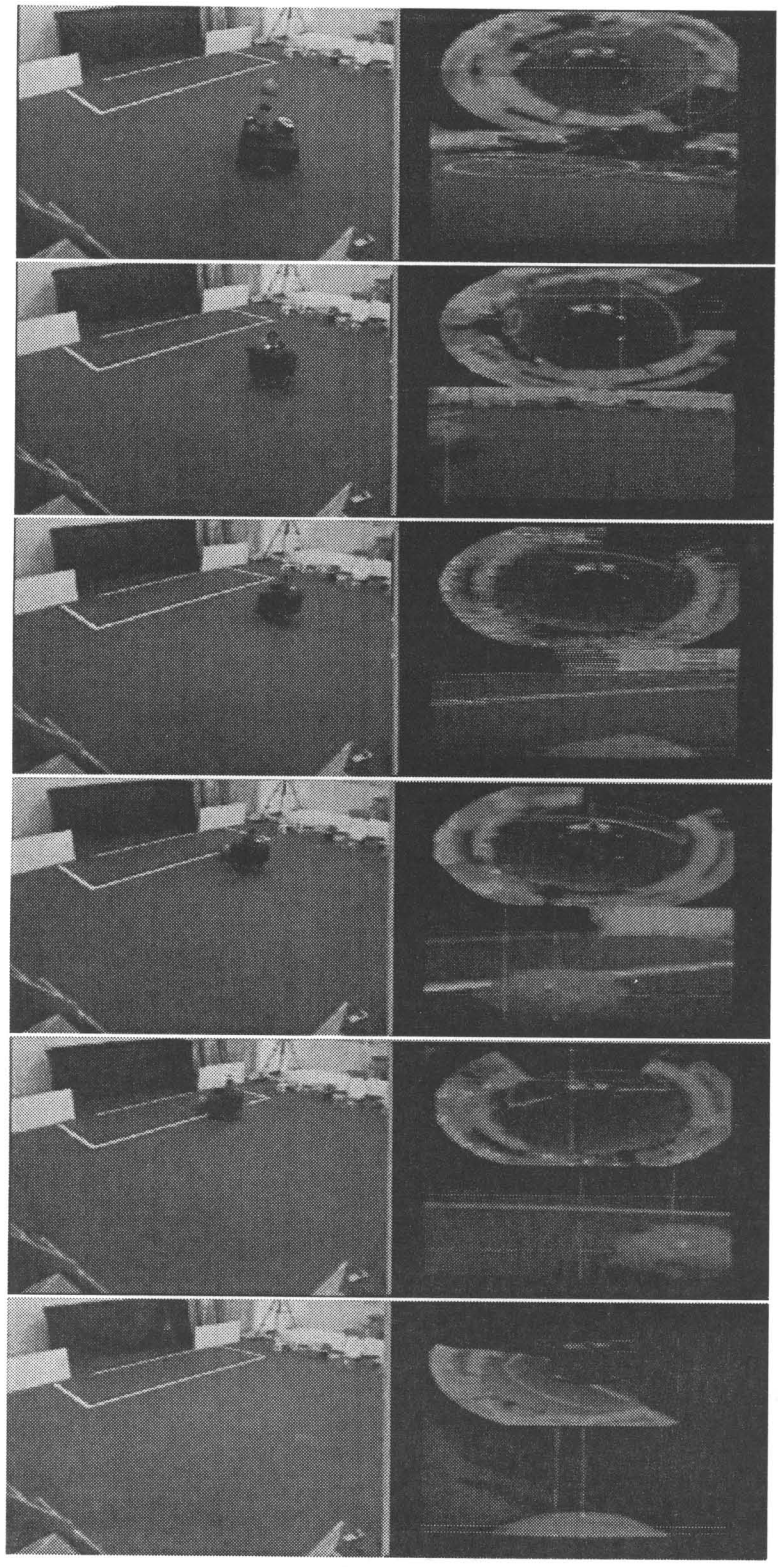
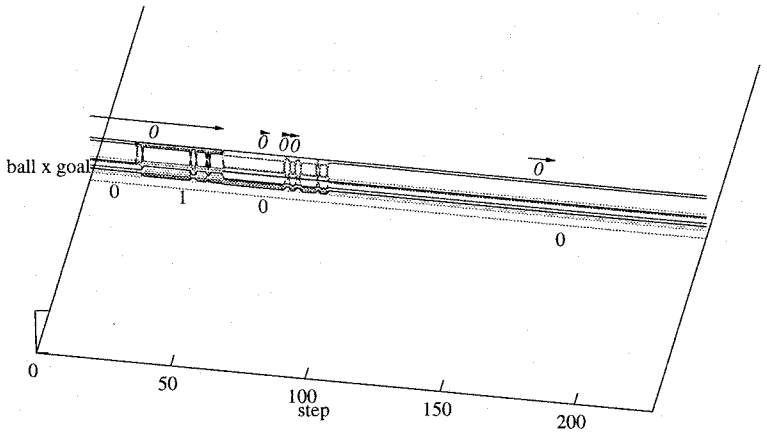
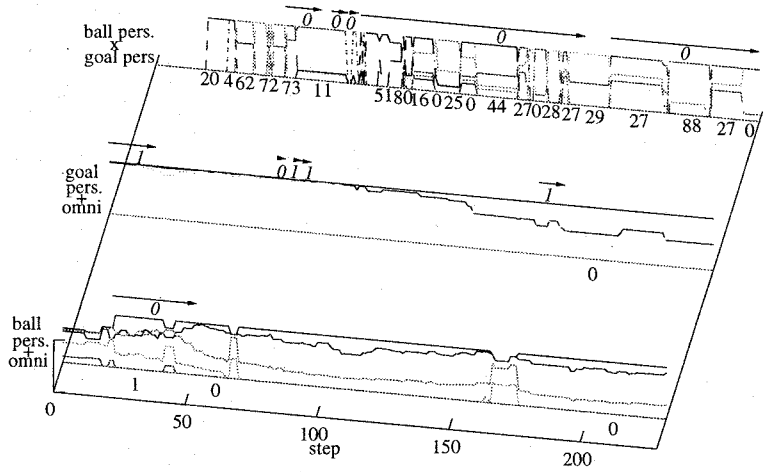


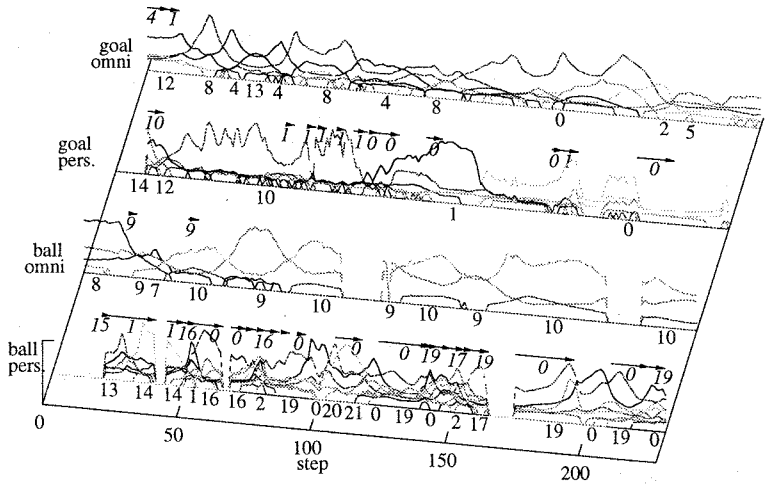
Figure 5.19: A sequence of a shooting behavior and its camera images



(a) third layer



(b) second layer



(b) first layer

Figure 5.20: A sequence of the goal state activation and behavior activation of learning modules

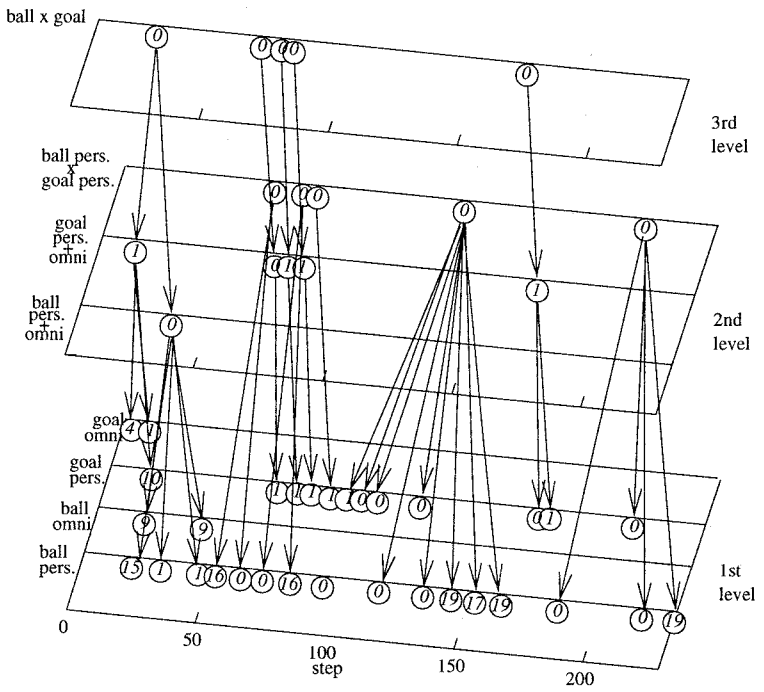


Figure 5.21: A sequence of the behavior activation of learning modules and the commands to the lower layer modules

Chapter 6

Conclusions and Future Works

The purpose of our study is to develop methods for an autonomous robot with learning capability of knowledge and behaviors through the interaction between the robot and the environment within a reasonable learning time with human designer's help as little as possible. We put focus on reinforcement learning as a method for behavior learning with little or no a priori knowledge and higher capability of reactive and adaptive behaviors.

However, the reinforcement learning methods have one critical disadvantage, that is, lack of the scalability. The motivation of our study is to extend the scalability in applying reinforcement learning methods to real autonomous robots.

We found that there are two major approaches to solve lack of the scalability; one is keeping the state and action space small enough, and the other is an introduction of hierarchy and multi-module architecture. Both approaches have the same essential issue of the state and action space construction.

The objective of this dissertation is to show ideas on the state and action space construction in single and multi-layered learning systems and the concrete models of the hierarchical control system for autonomous robot, and verify them through the real robot experiments.

In chapter 3, we proposed a method by which a robot learns purposive behavior within less learning time by incrementally segmenting the sensor space based on the experiences of the robot. The incremental segmentation is performed by constructing local models in the state space, which is based on the function approximation of the sensor outputs to reduce the learning time and the reinforcement signal to emerge a purposive behavior. We applied the idea to a soccer robot which tries to shoot a ball into a goal. The experiments with computer simulations and a real robot were shown. As a result, our real robot has learned a shooting behavior within less than

one hour training by incrementally segmenting the state space.

In chapter 4, we proposed a method by which a hierarchical structure for behavior learning is self-organized. The modules in the lower networks are organized as experts to move into different categories of sensor output regions and learn lower level behaviors using motor commands. In the meantime, the modules in the higher networks are organized as experts which learn higher level behavior using lower modules. Each module assigns its own goal state by itself. We applied the method to a simple soccer situation in the context of RoboCup, and showed the experimental results.

In chapter 5, we proposed an approach to the problem of decomposing the large state space at the bottom level into several subspaces and merging those subspaces at the higher level. This allows the system

- to acquire behaviors based on new state features by adding new learning modules layers based on them while it leaves the already learned hierarchical structure,
- to acquire behaviors based on state space constructed with a large number of state valuables by merging the modules at the lower levels which acquire behaviors based on subspaces, and
- to save computational resources because the number of state could be small by decomposing the whole state space into small subspaces.

We apply the method to a simple soccer situation in the context of RoboCup, and showed the experimental results.

As future works, there are a number of approaches to extend our current methods.

Integration with forward model We have adopted a direct reinforcement learning (Q-learning) method with no explicit model through our works to keep our points on the research simple. However, it seems obvious that the world model will help the learning system to acquire the optimal policy, and there are a number of studies which utilized this model. Sutton [58, 57, 59, 56] has proposed DYNA-architectures which integrate world model learning and execution-time planning. Atkeson and Santamaria [8] has shown that a model-based reinforcement learning is more data efficient and found better policies than a direct reinforcement learning. Generally speaking, the model-based learning system shows good performance though it needs rather computational resources.

Advanced mechanism for self-organization of hierarchy We proposed two approaches of state and action space construction; "multiplicative approach" and "complementary approach". However, we still needs a mechanism which enable the system to select layers to be combined, to judge which approach is

suitable, in order to develop various kinds of purposive behaviors. The following mechanism is considered to be adopted. One of the traditional stochastic methods identifies whether the layers are independent from each other or not. If the layers are independent from each other, the system apply "multiplicative approach". Else, it applies "complementary approach".

Implementation for Multi-modal sensor devices and actuators We have developed a series of vision based mobile robots. Their sensing devices are based on only vision system with a perspective camera and a camera with an omni-directional mirror, and the robots have wheel based locomotion devices. They can load some kinds of alternative sensing devices, such as touch sensors, proximity sensors (infra-red sensor, or ultra sonic sensor), relatively longer distance sensor (laser range sensor), and so on. It can also attach alternative actuators, such as object manipulators and ball kicking devices. There are obviously interesting and challenging issues in integration of multi-modal sensory devices and many DOFs to generate various kinds of behaviors.

Introduction of a Top-down Approach We have developed a method of bottom up approach. We may include a top-down approach in our method. The modules of higher layer can advice the system to create lower layer with more fine state space in the case the policy acquisitions of the higher layer would not work because of the interference between lower layer modules mentioned in **5.7**.

Bibliography

[1] J. S. Albus. Data storage in the cerebellar model articulation controller (cmac). *Journal of Dynamic Systems, Measurement, and Control, Trans. ASME*, Vol. 97, No. 3, pp. 227–233, 1975.

[2] J. S. Albus. A new approach to manipulator control: The cerebbellar model articulation controller (cmac). *Journal of Dynamic Systems, Measurement, and Control, Trans. ASME*, Vol. 97, No. 3, pp. 220–227, 1975.

[3] James S. Albus. The engineering of mind. In *Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior (From Animals to Animats 4)*, pp. 23–32. MIT Press, 1996.

[4] M. Asada, S. Noda, S. Tawaratumida, and K. Hosoda. Purposive behavior acquisition for a real robot by vision-based reinforcement learning. *Machine Learning*, Vol. 23, pp. 279–303, 1996.

[5] Minoru Asada, Shoichi Noda, and Koh Hosoda. Non-physical intervention in robot learning based on lfe method. In *Proc. of Machine Learning Conference Workshop on Learning from Examples vs. Programming by Demonstration*, 1995.

[6] Minoru Asada, Shoichi Noda, and Koh Hosoda. Action-based sensor space categorization for robot learning. In *Proceedings of the 1996 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Vol. 3, pp. 1502–1509, 1996.

[7] Minoru Asada, Shoichi Noda, Sukoya Tawaratsumida, and Koh Hosoda. Vision-based reinforcement learning for purposive behavior acquisition. In *Proceedings of 1995 IEEE International Conference on Robotics and Automation*, Vol. 1, pp. 146–153, 1995.

[8] Chirstopher G. Atkeson and Juan Carlos Santamaria. A comparison of direct and model-based reinforcement learning. In *International Conference on Robotics and Automation*, 1997.

[9] Justin Boyan and Andrew Moore. Generalization in reinforcement learning: Safely approximating the value function. In *Proceedings of Neural Information Processings Systems 7*. Morgan Kaufmann, January 1995.

[10] R. A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, Vol. 2, No. 1, pp. 14–23, 1986.

[11] Rodney Allen Brooks. *Cambrian Intelligence*. MIT Press, July 1999.

[12] M. Carreas, J. Batlle, and P. Ridao. Hybrid coordination of reinforcement learning-based behaviors for auv control. In *Proceedings of the 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1410–1415, Maui, Hawai, USA, Oct. 2001.

[13] David Chapman and Leslie Pach Kaelling. Input generalization in delayed reinforcement learning: An algorithm and performance comparisons. In *AAAI'91*, pp. 726–731, 1991.

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

[15] Jonalthan H. Connell and Sridhar Mahadevan. *ROBOT LEARNING*. Kluwer Academic Publishers, 1993.

[16] Jonathan H. Connell and Sridhar Mahadevan. Rapid task learning for real robots. In *ROBOT LEARNING*, chapter 5, pp. 105–140. Kluwer Academic Publishers, 1993.

[17] Peter Dayan and Geoffrey E. Hinton. Feudal reinforcement learning. In Stephen José Hanson, Jack D. Cowan, and C. Lee Giles, editors, *Advances in Neural Information Processing Systems*, Vol. 5, pp. 271–278. Morgan Kaufmann, San Mateo, CA, 1993.

[18] Bruce L. Digney. Emergent hierarchical control structures: Learning reactive/hierarchical relationships in reinforcement environments. In Pattie Maes, Maja J. Mataric, Jean-Arcady Meyer, Jordan Pollack, and Stewart W. Wilson, editors, *From animals to animats 4: Proceedings of The fourth conference on the Simulation of Adaptive Behavior: SAB 96*, pp. 363–372. The MIT Press, 1996.

[19] Bruce L. Digney. Learning hierarchical control structures for multiple tasks and changing environments. In Rolf Pfeifer, Bruce Blumberg, Jean-Arcady Meyer, and Stewart W. Wilson, editors, *From animals to animats 5: Proceedings of The*

fifth conference on the Simulation of Adaptive Behavior: SAB 98, pp. 321–330. The MIT Press, 1998.

[20] Kenji Doya. Temporal difference learning in continuous time and space. In D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, editors, *Advances in Neural Information Processing System 8*, pp. 1073–1079. MIT Press, Cambridge, MA, 1996.

[21] Kenji Doya. Efficient nonlinear control with actor-tutor architecture. In M. C. Mozer, M. I. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems 9*, pp. 1012–1018. MIT Press, Cambridge, MA, USA, 1997.

[22] Kenji Doya, Kazuyuki Samejima, Ken ichi Katagiri, and Mitsuo Kawato. Multiple model-based reinforcement learning. Technical report, Kawato Dynamic Brain Project Technical Report, KDB-TR-08, Japan Science and Technology Corporatio, June 2000.

[23] Artur Dubrawski and Patrick Reignier. Learning to categorize perceptual space of a mobile robot using fuzzy-art neural network. In *Proceedings of the IEEE/RSJ/GI International Conference on Intelligent Robots and Systems*, Vol. 2, pp. 1272–1277, September 1994.

[24] Teruo Fujii, Yoshikazu Arai, Hajime Asama, and Isao Endo. Multilayered reinforcement learning for complicated collision avoidance problems. In *Proceedings of the 1998 IEEE International Conference on Robotics and Automation*, pp. 2186–2198, 1998.

[25] Masahiko Haruno, Daniel M. Wolpert, and Mitsuo Kawato. Multiple paired forward-inverse models for human motor learning and control. *Advances in Neural Information Processing Systems*, Vol. 11, pp. 31–37, 1999. MIT Press, Cambridge, Massachusetts.

[26] Masahiko Haruno, Daniel M. Wolpert, and Mitsuo Kawato. Mosaic model for sensorimotor learning and control. *Neural Computation*, Vol. 13, pp. 2201–2220, 2001.

[27] Yasuhisa Hasegawa and Toshio Fukuda. Learning method for hierarchical behavior controller. In *Proceedings of the 1999 IEEE International Conference on Robotics and Automation*, pp. 2799–2804, 1999.

[28] Yasuhisa HASEGAWA, Hiroaki TANAHASHI, and Toshio FUKUDA. Behavior coordination of brachation robot based on bahavior phase shift. In *Proceedings of*

the 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems, Vol. CD-ROM, pp. 526–531, 2001.

[29] Long ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, Vol. 8, pp. 293–321, 1992.

[30] Leslie Pack Kaelbling. Hierarchical learning in stochastic domains: Preliminary results. In *Proceedings of the Tenth International Conference on Machine Learning*, 1993.

[31] Leslie Pack Kaelbling. Learning to achieve goals. In *Proceedings of the Thirteenth International Joint Conference on Artificial Inteligence*, 1993.

[32] Ben J. A. Kröse and Joris W. M. van Dam. Adaptive state space quantization for reinforcement learning of collision-free navigation. In *Proceedings of the 1992 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Vol. 2, pp. 1327–1331, 1992.

[33] Sridhar Mahadevan and Jonathan Connel. Scaling reinforcement learning to robotics by exploiting the subsumption architecture. In *Proceedings Eighth International Workshop on Machine Learning*, 1991.

[34] Sridhar Mahadevan and Jonathan Connell. Automatic programming of behavior-based robots using reinforcement learning. In *AAAI-91*, Vol. 2, pp. 768–773, 1991.

[35] Andrew Kachites McCallum. Learning to use selective attention and short-term memory in sequential tasks. In *Proceedings of the fourth International Conference on Simulation of Adaptive Behavior (From animals to animats 4)*, pp. 315–324, 1996.

[36] R. Andrew McCallum. Reduced training time for reinforcement learning with hidden state. In *The Proceedings of the Eleventh International Machine Learning Workshop (Robot Learning)*, 1994.

[37] R. Andrew McCallum. Instance-based utile distinctions for reinforcement learning with hidden state. In Armand Prieditis and Stuart Russell, editors, *The Twelfth Internationa Conference on Machine Learning*, pp. 387–395, Tahoe City, California, July 1995.

[38] Takahashi Minato and Minoru Asada. Environmental change adaptation for mobile robot navigation. In *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1859–1864, 1998.

[39] A. Moore. Variable resolution dynamic programiming: efficiently learning action maps in multivariate real-values state spaces. In *Proceedings Eighth International Workshop on Machine Learning*, 1991.

[40] Andrew W. Moore and Christopher G. Atkeson. The parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces. *Machine Learning*, Vol. 21, pp. 199–233, 1995.

[41] Jun Morimoto and Kenji Doya. Hierarchical reinforcement learning of low-dimensional subgoals and high-dimensional trajectories. In *The 5th International Conference on Neural Information Processing*, Vol. 2, pp. 850–853, 1998.

[42] Rémi Munos and Jocelyn Parinel. Reinforcement learning with dynamic covering of state-action space : Partitioning q-learning. In *Proceddings of the Third International Conference on Simulation of Adaptive Behavior (From Animals to Animats 3)*, pp. 354–363, 1994.

[43] N. Ono and K. Fukumoto. Multi-agent reinforcement learning: A modular approach. In *Proceedings of the 2nd International Conference on Multi-agent Systems (ICMAS-96)*. AAAI Pres, 1996.

[44] N. Ono and K. Fukumoto. Learning to coordinate in a continuous environment. In *Distributed Artificial Intelligence Meets Machine Learning: Learning in Multi-agent Environments*, pp. 73–81. Springer-Verlag, g. weiss edition, 1997.

[45] N. Ono and K. Fukumoto. A modular approach to multi-agent reinforcement learning. In *Distributed Artificial Intelligence Meets Machine Learning: Learning in Multi-agent Environments*, pp. 25–39. Springer-Verlag, g. weiss edition, 1997.

[46] Fuminori Saito and Toshio Fukuda. Learning architecture for real robot systems { extension of connectionist Q-learning for continuous robot control domain }. In *Proceedings of IEEE Int. Conf. on Robotics and Automation*, Vol. 1, pp. 27–32, 1994.

[47] Fuminori Saito and Toshio Fukuda. Two-link-robot brachiation with connectionist q-learning. In *Proceedings of the third international conference on simulation of adaptive behavior (From animals to animats 3)*, pp. 309–314. The MIT Press, 1994.

[48] Juan C. Santamaria and Ashwin Ram Richard S. Sutton. Experiments with reinforcement learning in problems with continuous state and action spaces. *Adaptive Behavior*, Vol. 6, No. 2, pp. 163–217, 1998.

[49] Ritsuko Sato, Hiroshi Ishiguro, and Toru Ishida. Robot oriented state space construction based on sensor data analysis. In *8th SICE Symposium on Decentralized Autonomous Systems*, pp. 111–119, Tokyo, January 1996.

[50] Satinder P. Singh. The effeicient learnig of multiple task sequences. In *Neural Information Processing Systems 4*, pp. 251–258, 1992.

[51] Satinder P. Singh. Reinforcement learning with a hierarchy of abstract models. In *National Conference on Artificial Intelligence*, pp. 202–207, 1992.

[52] Satinder Pal Singh. Scaling reinforcement learning algorithms by learning variable temporal resolution models. In *Proceedings of the Ninth International Conference on Machine Learning*, pp. 406–415, 1992.

[53] Satinder Pal Singh. Transfer of learning by composing solutions of elemental sequential tasks. *Machine Learning*, Vol. 8, pp. 323–339, 1992.

[54] Peter Stone and Mamuela Veloso. Layered approach to learning client behaviors in the robocup soccer server. *Applied Artificial Intelligence*, Vol. 12, No. 2-3, 1998.

[55] Peter Stone and Manuela Veloso. Team-partitioned, opaque-transition reinforcement learning. In M. Asada and H. Kitano, editors, *RoboCup-98: Robo Soccer World Cup II*, pp. 261–272. Springer Verlag, Berlin, 1999.

[56] Richard S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the Seventh International Conference on Machine Learning*, pp. 216–224. Morgan Kaufmann, 1990.

[57] Richard S. Sutton. Dyna, an integrated architecture for learning, planning and reacting. In *Working Notes of the 1991 AAAI Spring Symposium on Integrated Intelligent Architectures*, pp. 151–155, 1991.

[58] Richard S. Sutton. Integrated modeling and control based on reinforcement learning and dynamic programming. *Advances in Neural Information Processing Systems 3*, pp. 471–478, 1991.

[59] Richard S. Sutton. Planning by incremental dynamic programming. In *Proceedings of the Eighth International Workshop on Machine Learning*, pp. 353–357. Morgan Kaufmann, 1991.

[60] Richard S. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Proceeding of Neural Information Processing Systems(NIPS 95)*, 1995.

[61] Richard S. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in Neural Information Processing Systems 8*, pp. 1038–1044, 1996.

[62] Yasutake Takahashi, Masanori Takeda, and Minoru Asada. Continuous valued q-learning for vision-guided behavior acquisition. In *Proceeding of the 1999 IEEE International Conference on Multisenso Fusion and Integration for Intelligent Systems*, pp. 255–260, 1999.

[63] Ming Tan. Cost-sensitive reinforcement learning for adaptive classification and control. In *Cost-Sensitive Reinforcement Learning for Adaptive Classification and Control*, pp. 774–780. AAAI Press, 1991.

[64] Ming Tan. Learning a cost-sensitive internal representation for reinforcement learning. In *Proceedings Eighth International Workshop on Machine Learning*, pp. 358–362, 1991.

[65] Fumihide Tanaka and Masayuki Yamamura. An approach to lifelong reinforcement learning through multiple environments. In *6th European Workshop on Learning Robots*, pp. 93–99, 1997.

[66] J. Tani and S. Nolfi. Self-organization of modules and their hierarchy in robot learning problems: A dynamical systems approach. Technical report, Sony CSL Technical Report, SCSL-TR-97-008, 1997.

[67] Jun Tani and Stefano Nolfi. Self-organization of modules and their hierarchy in robot learning problems: A dynamical systems approach. Technical report, Technical Report: SCSL-TR-97-008, 1997.

[68] Chen K. Tham and Richard W. Prager. A modular Q-learning architecture for manipulator task decomposition. In *Proc. 11th International Conference on Machine Learning*, pp. 309–317. Morgan Kaufmann, 1994.

[69] Sebastian Thrun. A lifelong learning perspective for mobile robot control. In *In Proceedings of the IEEE/RSJ/GI International Conference on Intelligent Robots and Systems*, Vol. 1, pp. 23–30, 1994.

[70] Eiji Uchibe, Minoru Asada, and Koh Hosoda. Behavior coordination for a mobile robot using modular reinforcement learning. In *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems 1996 (IROS '96)*, pp. 1329–1336, 1996.

[71] Steven D. Whitehead. Complexity and cooperation in q-learning. In *Proceedings Eighth International Workshop on Machine Learning (ML91)*, pp. 363–367, 1991.

[72] Steven D. Whitehead and Dana H. Ballard. Active perception and reinforcement learning. In *Proceedings of the 7-th conference on Machine Learning*, pp. 179–188, 1990.

[73] Steven D. Whitehead and Long-Ji Lin. Reinforcement learning of non-markov dicision processes. *Artificial Intelligence*, Vol. 73, pp. 271–306, 1995.

[74] Steven Whitehead, Jonas Karlsson, and Jsho Tenenberg. Learning multiple goal behavior via task decomposition and dynamic policy merging. In Jonathan H. Connell and Sridhar Mahadevan, editors, *ROBOT LEARNING*, chapter 3, pp. 45–78. Kluwer Academic Publishers, 1993.

[75] 榎田修一, 河野宗一, 大橋健, 江島俊朗. センサ空間の拡大を用いた eq-学習. 第19回日本ロボット学会学術講演会, pp. 85–86, 2001.

[76] 金谷健一. 画像理解 - 3次元認識の数理 -. 森北出版株式会社, 1990.

[77] 佐藤律子, 石黒浩, 石田亨. センサーデータの解析に基づくロボットに適した状態空間 の構成. 第8回自律分散システム・シンポジウム資料, pp. 111–116, 1996.

[78] 河口至商. 多変量解析入門. 森北出版株式会社, 1973.

Published Papers by the Author

Articles in Japanese Journal

- Yasutake Takahashi and Minoru Asada, "State-Action Space Construction for Multi-Layered Learning System", In *Journal of Robotics Society of Japan*, 2002 (submit)

- Yasutake Takahashi and Minoru Asada, "Behavior Acquisition by Multi-Layered Reinforcement Learning", In *Journal of Robotics Society of Japan*, Vol. 18, No. 7, pp. 1040-1046, 2000.

- Yasutake Takahashi and Minoru Asada, "Incremental State Space Segmentation for Behavior Learning by Real Robot", In *Journal of Robotics Society of Japan*, Vol. 17, No. 1, pp 118 - 124, 1999

Papers in Proceeding of International Conferences

- Y. Takahashi and M. Asada, "Multi-Controller Fusion in Multi-Layered Reinforcement Learning", International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI2001), pp. 7-12, 2001.

- Yasutake Takahashi, Takashi Tamura, and Minoru Asada. "Cooperation via Environmental Dynamics Caused by Multi Robots in a Hostile Environment". The Fourth IFAC Symposium on Intelligent Autonomous Vehicles, pp. 413-418, 2001.

- Y. Takahashi and M. Asada, "Vision-Guided Behavior Acquisition of a Mobile Robot by Multi-Layered Reinforcement Learning", IEEE/RSJ International Conference on Intelligent Robots and Systems, Vol. 1, pp. 395-402, 2000

- Yasutake Takahashi, Masanori Takeda, and Minoru Asada, "Improvement Continuous Valued Q-learning and its Application to Vision Guided Behavior Acquisition", The Fourth International Workshop on RoboCup, pp. 255-260, 2000
- Yasutake Takahashi and Minoru Asada, "Behavior Acquisition by Multi-Layered Reinforcement Learning", Proceeding of the 1999 IEEE International Conference on Systems, Man, and Cybernetics, pp 716 – 721, 1999
- Yasutake Takahashi, Masanori Takeda, and Minoru Asada, "Continuous Valued Q-learning for Vision-Guided Behavior Acquisition", Proceeding of the 1999 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems, pp 255 - 260
- Yasutake Takahashi, Minoru Asada, Shoichi Noda and Koh Hosoda, "Sensor Space Segmentation for Mobile Robot Learning" , Proceedings of ICMAS'96 Workshop on Learning, Interaction and Organizations in Multiagent Environment
- Yasutake Takahashi, Minoru Asada and Koh Hosoda, "Reasonable Performance in Less Learning Time by Real Robot Based on Incremental State Space Segmentation", Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems, pp 1518–1524, 1996

Papers in Proceedings of Japanese Conference and Miscellanies

- 高橋 泰岳, 浅田 稔, "複数の学習機構の階層的構築による行動獲得", ロボティクス・メカトロニクス講演会 2000 予稿集, CD-ROM, 2000.
- 高橋 泰岳, 浅田 稔, "複数の学習機構の階層的構築による行動獲得" , 第 17 回日本ロボット学会学術講演会予稿集, pp 981 - 982, 1999
- 高橋 泰岳, 細田 耕, 浅田 稔, "状態空間の自律的分割による実ロボットの実時間学習" , ロボティクス・メカトロニクス講演会'96, pp 121-124, 1996
- 中西, 高橋, 鈴木, 浅田, "全方位視覚を持った移動ロボットの自律的行動獲得", 第 41 回システム制御情報学会研究発表講演会予稿集, pp 41-42, 1997.
- 金石, 高橋, 鈴木, 浅田, "強化学習的手法を用いての視覚サーボに基づいた移動ロボットのホー ミング行動の実現", 第 6 回インテリジェント FA シンポジウム講演論文集, pp 29-32, 1997.