Cooperative Behavior Acquisition by Learning and Evolution in a Multi-Agent Environment for Mobile Robots

by

Eiji Uchibe

Submitted to the Department of Mechanical Engineering for

Computer-Controlled Machinery

for the degree of

Doctor of Engineering

at the

Osaka University

on January 1999

Abstract

The objective of my research described in this dissertation is to realize learning and evolutionary methods for multiagent systems. This dissertation mainly consists of four parts.

We propose a method that acquires the purposive behaviors based on the estimation of the state vectors in Chapter 3. In order to acquire the cooperative behaviors in multiagent environments, each learning robot estimates the *Local Prediction Model* (hereafter *LPM*) between the learner and the other objects separately. The *LPM* estimate the local interaction while reinforcement learning copes with the global interaction between multiple *LPMs* and the given tasks. Based on the *LPMs* which satisfies the Markovian environment assumption as possible, robots learn the desired behaviors using reinforcement learning. We also propose a learning schedule in order to make learning stable especially in the early stage of multiagent systems.

Chapter 4 discusses how an agent can develop its behavior according to the complexity of the interactions with its environment. A method for controlling the complexity is proposed for a vision-based mobile robot. The agent estimates the full set of state vectors with the order of the major vector components based on the *LPM*. The environmental complexity is defined in terms of the speed of the agent while the complexity of the state vector is the number of the dimensions of the state vector. According to the increase of the speed of its own or others, the dimension of the state vector is increased by taking a trade-off between the size of the state space and the learning time.

The vector-valued reward function is discussed in order to cope with the multiple tasks in Chapter 5. Unlike the traditional weighted sum of several reward functions, we introduce a discounted matrix to integrate them in order to estimate the value function, which evaluates the current action strategy. Owing to the extension of the value function, the learning agent can estimate the future multiple reward from the environment appropriately.

Chapter 6 discusses how multiple robots can emerge cooperative behaviors through co-evolutionary processes. A genetic programming method is applied to individual population corresponding to each robot so as to obtain cooperative and competitive behaviors. The complexity of the problem can be explained twofold: co-evolution for cooperative behaviors needs exact synchronization of mutual evolutions, and three robot co-evolution requires well-complicated environment setups that may gradually change from simpler to more complicated situations.

As an example task, several simplified soccer games are selected to show the validity of the proposed methods. Finally, discussion and concluding remarks on our work are given.

Thesis Supervisor : Minoru Asada
Title : Professor of Graduate School of Engineering,
Department of Adaptive Machine Systems, Osaka University

Thesis Committee : Minoru Asada, Chair
Yoshiaki Shirai
Masao Ikeda

Acknowledgments

So many people have contributed to my growth during my work on this dissertation that I am sure I will not remember to name them all.

I would like to thank my advisor, Professor Minoru Asada for his support for his patient guidance and constant encouragement throughout this work. His valuable advice and detailed criticism have enabled me to complete this dissertation. I also wish to my gratitude to Professor Yoshiaki Shirai and Professor Masao Ikeda for their constructive readings of this dissertation and their valuable advice.

A number of people have helped me through my dissertation by giving me words of encouragement or enthusiasm. I would particularly like to thank Dr. Koh Hosoda for his valuable guidance and stimulating discussions at the Laboratory. I also wish to my gratitude to Dr. Sho'ji Suzuki for his useful advice and discussions. I owe thanks to Dr. Takayuki Nakamura for his useful comments. His dissertation is a good guide to the study of "reinforcement learning for a vision-based mobile robot."

I would like to thank all members of the laboratory. Members of RoboCup group, including Shoichi Noda, Yasutake Takahashi, Masateru Nakamura, Chizuko Mishima, Hiroshi Ishizuka, and Tatsunori Kato, helped me with the implementation of the real experiments.

My family has also been a great blessing and source of joy to me over the years. My deepest thanks to them for always being there for me.

This research was supported by the Japan Society for the Promotion of Science, in Research for the Future Program titled Cooperative Distributed Vision for Dynamic Three Dimensional Scene Understanding (JSPS-RFTF96P00501).

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Building agents that learn to accomplish tasks through interactions with their environments has been acknowledged as one of the major challenges facing artificial intelligence and robotics. Especially, multiagent systems have been receiving increased attention in the field of artificial intelligence, computer vision, artificial life and behavior-based robotics. Multiagent systems are important to study for a number of reasons, both in terms of the analysis of existing systems, and in terms of the more effective synthesis of new systems in such fields as distributed computation, decentralized control, distributed artificial intelligence. Multiagent systems are often required because of spatial or geographic distribution, or in situations where centralized information is not available or is not practical. Even when a distributed approach is not required, multiple agents may still provide an excellent way of scaling up to approximate solutions for very large problems by streamlining the search through the space of possible policies.

In a multiagent environment, interactions among agents is inevitable. Otherwise, we can not say "multiagent" environment. Interactions can be classified into several categories : competition, cooperation, interference, ignorance, and so on. In general, cooperative behaviors or social behaviors is difficult to design in a dynamic environment unless all behaviors of other agents can be completely predicted. However, perfect prediction is almost impossible due to the limit of sensing capability (partial information and dynamic changes in the environment caused by actions of multiagent).

Generally, we have following difficult problems in multiagent learning:

1. **State representation**
   Learning agents do not know other agents' policies in advance, therefore they need to estimate the policies through observations and actions. What's the worse is that they may change their policies through a learning process.

2. **Credit assignment**
   If the credit involves group evaluation only, one robot may accomplish a given task

by itself and others do just actions irrelevant to the task as they do not seem to interfere the one robot's actions. Else, if only individual evaluation is involved, robots may compete each other. This trade-off should be carefully dealt.

There have been a variety of approaches in distributed artificial intelligence, and behavior-based robotics.

Distributed Artificial Intelligence

Distributed artificial intelligence (hereafter DAI) research can be divided into two general categories: distributed problem solving and multiagent systems [26]. One of the key issues in DAI research has been finding efficient mechanisms for coordination among multiple intelligent agents. Coordination may also be achieved by means of negotiation. For example, the *contract net* protocol [59] assigns tasks to agents on the basis of a bidding mechanism. Agents respond to task announcements with bids indicating how well they believe they can perform the task [51].

Recently, there has been a great deal of interest in incorporating learning into DAI systems. Reinforcement learning [15, 27, 66] has been receiving increased attention as a method for robot learning with little or no *a priori* knowledge and higher capability of reactive and adaptive behaviors.

However, there is still a gap between them because the assumptions are different between the task for DAI and robotics. Although the information derived from DAI researches is seen to be of value to robotics, we need more study to realize multiagent learning.

Behavior-Based Robotics

Since Brooks [11] proposed the behavior-based approach called "subsumption architecture", their group invented several kinds of behavior-based robots. The behavior-based approaches might be promising as fundamental issues since they can react to a dynamic environment such as multiagent environment.

Because these robots can take reflexive actions against the environment, we still lack a capability of generating purposive behaviors. Therefore, some methods which combine the robustness of the subsumption architecture and machine learning in order to overcome this problem.

1.2 Our Approach

The objective of this dissertation is to develop techniques for behavior acquisition by learning and evolution in a multiagent environment. We assume that

- there are no communication among the learning agents explicitly,

- there are no global vision,

to acquire cooperative and/or competitive behaviors.

This dissertation consists of eight chapters and three appendices. Chapter 2 shows a survey of related works from several viewpoints.

Chapter 3 describes one of our basic ideas. We propose a concept of *Local Prediction Model* (hereafter *LPM*) to obtain an internal representation of several interactions from a viewpoint of integration of observation and action. In order to apply reinforcement learning to cooperative behavior acquisition in multiagent environments, each learning robot estimates the *LPMs* between the learner and the other objects separately. Based on the *LPMs*, robots learn the desired behaviors using reinforcement learning.

Chapter 4 gives an another application of *LPM*. In other words, we discusses how a robot can develop its state vector according to the complexity of the interactions with its environment. A method for controlling the complexity is proposed for a vision-based mobile robot. First, we provide the most difficult situation, and the robot estimates the full set of state vectors with the order of the major vector components based on the the *LPM*. According to the increase of the environmental complexities, the dimension of the state vector is increased by taking a trade-off between the size of the state space.

Chapter 5 shows the vector-valued reward function and behavior learning in the context of multiple behavior coordination to cope with the tradeoff between the individual and group utility. Unlike the traditional weighted sum of several reward functions, we define a vector-valued value function which evaluates the current action strategy by introducing a discounted matrix to integrate several reward functions. We implement the actor-critic architecture [66] as an integration of the *LPMs* and the vector-valued reward function.

Chapter 6 discusses how multiple robots can emerge cooperative behaviors through co-evolutionary processes. A genetic programming method is applied to individual population corresponding to each robot so as to obtain cooperative and competitive behaviors through evolutionary processes. The complexity of the problem can be explained twofold: co-evolution for cooperative behaviors needs exact synchronization of mutual skill developments, and three robot co-evolution requires well-complicated environment setups that may gradually change from simpler to more complicated situations so that they can obtain cooperative and competitive behaviors simultaneously in a wide range of search area in various kinds of aspects.

Chapter 7 discusses the open problems of our system and gives some comments. Finally, we summarize the key points of the dissertation and gives additional conclusion.

Chapter 2

Related Works

2.1 From a Viewpoint of Methodologies

2.1.1 Evolutionary Technique

Recently, co-evolution has been receiving increased attention as a method for multiagent simultaneous learning. *Genetic Algorithm* (hereafter *GA*) [20] and *Genetic Programming* (hereafter *GP*) [32, 33] are often used by the many researchers whose interests are co-evolution, Artificial Life, and so on.

Pursuit problem between two robots (prey and predator) is often argued as an example task. Cliff and Miller [12] have analyzed the relationship between a prey and a predator in computer simulation. Floreano and Nolfi [18] have implemented simulated and real robot experiments which co-evolved prey and predator robots of which skills gradually leveled up under certain conditions. In this problem, the relation between the prey and the predator is completely competitive.

Luke *et al.* [39] apply a *GP* method to the soccer game to evolve teams each of which can be regarded as an individual and attempts to beat other teams, that is, co-evolution for competition.

As described above, the previous researches related to co-evolution coped with competition between individuals. In the realm of nature, we can see various aspects of behaviors emerged in multiagent environments, not only competition but also cooperation, ignorance, and so on. That means there could be artificial co-evolution for other than competition.

2.1.2 Reinforcement Learning

Sen *et al.* [57] described two-agent block pushing experiments, where the agents try to make the block follow a line by independently applying forces to it. It is not clear why they used Q learning, since this is not a sequential reinforcement learning problem where the goal is to maximize the immediate reinforcement at each step.

Ono *et al.* [53] utilized Q learning and proposed modular Q learning in order to avoid the state explosion problem in reinforcement learning. This method has to decompose the state space and allocate suitable modules of modular Q learning in advance. Kohri *et al.* [31] improved their method which can allocate the Q modules appropriately. Arai *et al.* [3] applied modified profit sharing based on the Rationality Theorem [47] and Q learning into the pursuit problem in a toroidal grid world. They concluded that profit sharing is more suitable than Q learning experimentally in multiagent reinforcement learning. However, in these methods, the behaviors of other agents do not considered.

Littman [38] proposed the framework of Markov Games in which Q-learning agents try to learn a mixed strategy optimal against the worst possible opponent in a zero-sum 2-player game in a grid world. He assumed that the opponent's strategy is given to the learner (the opponent tries to minimize a single reward function, while it is to be maximized by the learning agent). Recently, Hu and Wellman [25] extended the Littman's work to the non-zero-sum 2-player game. However, they assume that

- the reward and action can be perceived each other, and
- the state space are shared among learning agents.

This assumption does not hold in robotic applications.

Sugita and Tani [64] implemented not only model learning but also behavior learning by one recurrent neural network and realize a simplified cooperative game between two robots in the real environment. However, they use the abstract features to train the network, and their application is essentially unstable.

Stone and Veloso [62, 63] proposed a hierarchical learning architecture called "layered learning" to develop a team of soccer agents. In the lowest level, the neural network coped with the interception of moving a ball. The second layer classify the state space whether the passing behavior can be accomplished or not by C4.5 [54] which is a kind of a decision tree algorithm.

2.2 From a Viewpoint of Agent Architectures

Hidden state is common in multiagent contexts, since each agent may only perceive its own local information using its own sensors. The perceptual aliasing problem has recently received a great deal of attention from AI researchers who are attempting to build *situated* or *embedded* agents, that is, agents that must perceive and act in the world.

2.2.1 Neural Network

Lin [37] compared window-Q based on both the current sensation and the N most recent sensations and actions with recurrent-Q based on a recurrent network, and he showed the latter is superior to the former because a recurrent network can cope with historical

features. However, it is still difficult to determine the number of neurons and the structures of network in advance. Furthermore, the number of context unit is originally fixed.

Rao and Fuentes [55] described a framework which would consist of two components for perception-based navigational behaviors in autonomous mobile robot. One is a stochastic hill-climbing method which copes with reactive behavior acquisition such as obstacle avoidance. The other is the *Predictive Sparse Distributed Memory* (hereafter *Predictive SDM*) based on Kanerva's model [28]. However, the *Predictive SDM* is a kind of supervised learning.

2.2.2 Tree Representation

McCallum [44] has developed the memory-based reinforcement learning based on the tree representation. His approach partitions the state space from sensory experiences, and uses memory of features to augment the agent's perceptual inputs.

Michaud and Mataric [45] proposed the method which introduce the use of the stored history within the subsumption framework. They checked a history of selected actions which is determined by a subsumption architecture, and modeled the interactions using tree representation. However, fundamental behaviors have been embedded as a form of subsumption architecture. Consequently, the agents can move around without their proposed method.

The interactions between multiagents in this way is said to be in *extensive* form.

2.3 From a Viewpoint of the Role of Other Agents

2.3.1 Learning by Teaching (Demonstration)

There might be several ways to categories a methodology of *learning from other agents* depending on who are other agents. Colleagues, opponents, teachers, critics, or gods. The most popular method of *learning from other agents* is learning by demonstration [35] in which other agents are kind and suggestive human operators.

Learning by teaching is often used in order to accelerate the learning speed. In classical teaching method, the designer has to prepare the optimal input-output data in advance. Recently, the method to evaluate the answer from the teacher was proposed by Dorigo and Colombetti [17]. In this case, the teacher can be regarded as other agent from a viewpoint of the learner. That is, *learning by teaching* is a kind of multiagent systems.

Learning from external critic proposed by Whitehead [74] is also categorized into this class although an critic (other agent) has not shown any demonstrations but gives an advice each time an agent takes an action. In these cases, other agents are cooperative but not involved in the environment in which an agent learns to accomplish the given tasks.

2.3.2 Imitation

Imitative learning is a term taken from ethology, and is also utilized to accelerate learning. There is a slight difference on the method to evaluate between *imitation* and *teaching*.

Yamaguchi *et al.* [76] proposed the idea of selective mimetism, which can let the agent to select whether learning or imitation by itself adaptively. This method assume that the learning agent can observe not only the internal information of other agents but also the action itself explicitly.

Demiris and Hayes [16] realized the imitation of translating movements between the teacher and the imitators in 2D space. In this method, the relationship between the teacher's and the imitator's posture was calculated.

Steels and Vogt [60, 61] proposed adaptive language games in order to communicate between agents through interactions. However, their method does not take the action of the learner into account. We suppose that the action as well as the sensory information are important to understand behaviors of other agents.

Kuniyoshi [34] proposed a framework called *Cooperation by Observation* and realize several patterns such as *Posing* (the observer is always orient toward the target robot), *Unblocking* (the observer get rid of obstacle that stand on the way), *Passing* (the observer take over the other robot's job of bucket brigading) in the real environment. These cooperative behaviors are accomplished by visual information. However, these architecture are designed by a programmer and the problem of the integrated architecture is the same as the case of the subsumption architecture.

Chapter 3

Local Prediction Model for Cooperative Behavior Acquisition

3.1 Introduction

In a multiagent environment, the standard reinforcement learning algorithm does not seem applicable because the environment including the other learning agents seems to change randomly from a viewpoint of the learning agent. There are two major reasons why the learning would be difficult in a multiagent environment:

A The other agents may use a stochastic action selector which might take a different action even if the same sensation occurs to it, and

B The other agents may have perception (sensation) different from the learning agent's. This means that the learning agent would not be able to discriminate different situations which the other agent can do, and vice versa.

As described above, the learning agents in multiagent environments need appropriate state representation in order for learning algorithms to converge safely. However, it is difficult to obtain a reasonable analytical model in advance. Although the uncertainties of sensor and actuator outputs are considered by a stochastic transition model in the state space, such a model cannot account for the accumulation of sensor errors in estimating the robot position. Further, from the viewpoint of real robot applications, the learning real robot should construct the state space so that it can reflect the outputs of the physical sensors which are currently available. Therefore, the modeling architecture based on its own experiences, that is, the sequences of observation and action are required to make the learning algorithms such as reinforcement learning and memory-based learning applicable.

In this chapter, we propose a concept of a *Local Prediction Model* (hereafter, *LPM*), which is a basic architecture for learning agents in multiagent environment. Each *LPM* estimates the relationships between a learner's behaviors and other robots through interactions (observation and action) based on the method of system identification. The basic,

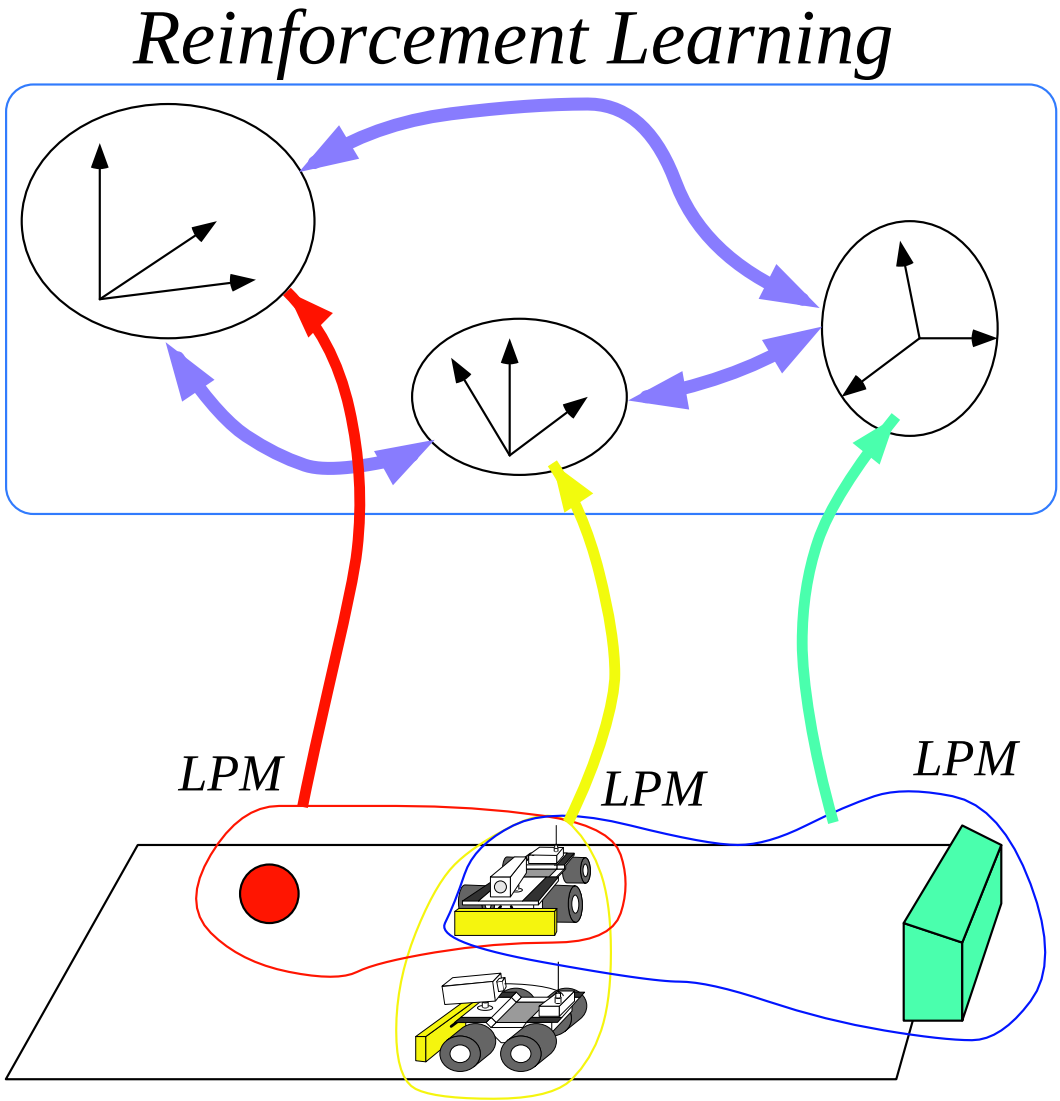


Fig.3.1 A basic idea of our approach: integration of multiple *LPMs* and the reinforcement learning

important idea is that the complexity of interaction can be regarded as the order of the state vector in the context of control theory. The problem that we face is how the agent can determine the order of the state vector and estimate the state vector from its experiences. In order for each *LPM* to construct the state vectors of other objects in an environment, we use Canonical Variate Analysis (CVA) [36] as a internal observer. Furthermore, we adopt Akaike's Information Criterion(AIC) [1] to determine the order of the estimated state vector. The model estimation process (construction of the state vectors) and the reinforcement learning can be regarded as identifying the lower environment dynamics of each object and obtaining the higher and non-linear interactions among the multiple *LPMs* and the given tasks. **Fig.3.1** shows a basic idea of our approach.

Next, we propose an efficient learning schedule which makes the learning processes stable and accelerates the learning in the early stage of learning. The only thing we do is to select the agent to be learned. Based on this scheduling method. The schedule specifies which agent should learn and and when to change its turn to another agent.

We apply the proposed method to a simplified soccer game. The task of the robot is to shoot a ball which is passed back from the other robot. Because the environment consists of the stationary agents (the goal), a passive agent (the ball) and an active agent (the other robot), the learner has to construct the appropriate state vectors for all of these agents separately. After the learning robot estimates the state vectors, the reinforcement learning is applied in order to acquire purposive behaviors. We show simulation results and real experiments and give a discussion.

3.2 Construction of Internal Model based on Observation and Action

3.2.1 Architecture for Each Learning Robot

As described above, the learning agent needs appropriate state vectors which can predict the future observation so as to make the learning processes stable. However, due to severe problems such as the limitation of sensing capabilities, the uncertainties of motion and unknown policies of other learning agent, the learning agent can not prepare the perfect state vectors in advance. Therefore, what the learning agent can do is an acquisition of state representation taking account of a trade-off between the number of parameters and the prediction error from the sequences of observation and learner's action.

Fig.3.2 shows a whole learning system for each agent to acquire purposive behaviors based on our basic idea. Suppose that there are N objects in the environment which can be discriminated by the learning agent. In the beginning, the learning agent estimates the interactions between the learning agent itself and one of N objects separately. Each LPM^i $(i = 1, \cdots, N)$ outputs the state vector $\boldsymbol{x}^i$ from the sequences of observation and action $\{\boldsymbol{y}^i, \boldsymbol{u}\}$. Based on the concepts of the *LPM*, a complexity of interaction is regarded as the dimension of the state vector. Furthermore, the learner has one reinforcement learning

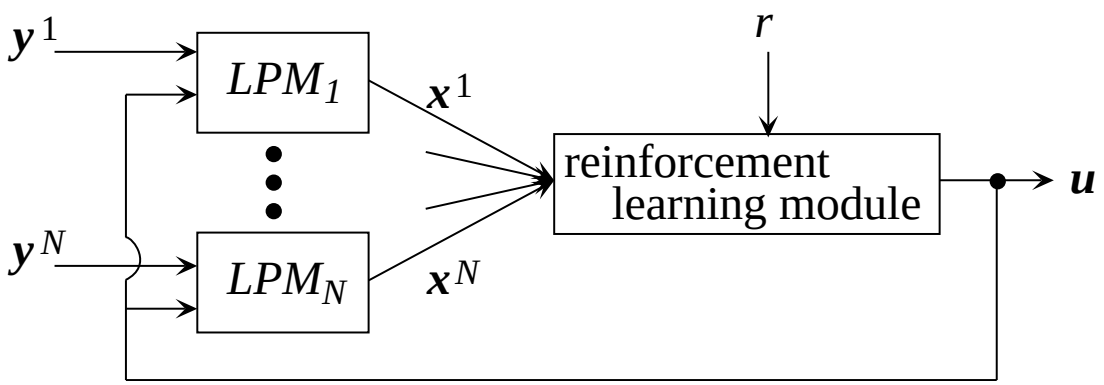


Fig.3.2 A whole system of our proposed method

module which estimate the action value function based on the set of N state ($i = 1, \cdots, N$) vectors $\boldsymbol{x}^i$ estimated by the LPM^i and a reward r from the environment.

Strictly speaking, all the robots do in fact interact with each other. Therefore, the learning robots should construct the state vectors taking these interactions into account. However, there are three problems:

1. It is intractable to collect the adequate input-output sequences and estimate the proper model because the dimension of state vector increases drastically,

2. The learner can not obtain the complete information to estimate them because of the partial observation due to the limitation of its sensing capability, and

3. There are several complexities of interactions between the learner and the other objects. The learning agent should have the different representations with respect to the complexity of interaction.

Therefore, the learning (observing) agent first estimates the state vector by the *LPM* to individual (observed) other agents or objects in an environment separately and then obtains the higher interactions among robots through the post reinforcement learning process.

3.2.2 State Representation in the LPM

The *LPM* has to cope with a MIMO (multiple input (action) and multiple output (observation)) system. A number of algorithms to identify MIMO combined deterministic-stochastic systems have been proposed in the field of system identification. In contrast to 'classical' algorithms such as PEM (Prediction Error Method), the subspace system identification algorithms [71] do not suffer from the problems caused by a priori parameterizations. Therefore, we utilize Larimore's Canonical Variate Analysis (CVA) [36] which

is one of the subspace state space identification methods. CVA is one of such algorithms, which uses canonical correlation analysis to construct a state vector. Here, we give a brief explanation of CVA method.

CVA uses a discrete time, linear, state space model as follows: Let be the input and output generated by the unknown system

$$\begin{aligned} \boldsymbol{x}_{t+1}^i &= \boldsymbol{A}^i \boldsymbol{x}_t^i + \boldsymbol{B}^i \boldsymbol{u}_t, \\ \boldsymbol{y}_t^i &= \boldsymbol{C}^i \boldsymbol{x}_t^i + \boldsymbol{D}^i \boldsymbol{u}_t, \end{aligned} \tag{3.1}$$

where $\boldsymbol{x}^i \in \Re^{n^i}$, $\boldsymbol{u} \in \Re^m$ and $\boldsymbol{y}^i \in \Re^{q^i}$ denote state vector, action code vector, and observation vector, respectively. Further, $\boldsymbol{A}^i \in \Re^{n^i \times n^i}$, $\boldsymbol{B}^i \in \Re^{n^i \times m}$, $\boldsymbol{C}^i \in \Re^{q^i \times n^i}$, and $\boldsymbol{D}^i \in \Re^{q^i \times m}$ represent parameter matrices.

Now, we construct new vectors as follows:

$$\boldsymbol{p}_t^i = \begin{bmatrix} \boldsymbol{u}_{t-1} \\ \vdots \\ \boldsymbol{u}_{t-l} \\ \boldsymbol{y}_{t-1}^i \\ \vdots \\ \boldsymbol{y}_{t-l^i}^i \end{bmatrix}, \quad \boldsymbol{f}_t^i = \begin{bmatrix} \boldsymbol{y}_t^i \\ \boldsymbol{y}_{t+1}^i \\ \vdots \\ \boldsymbol{y}_{t+l^i-1}^i \end{bmatrix}. \tag{3.2}$$

where l^i is a historical length to be considered about the i-th object. Subspace state space identification can estimate the state vector directly from the relationship between two subspaces $\boldsymbol{P}^i$ and $\boldsymbol{F}^i$ constructed by two vectors $\boldsymbol{p}^i$ and $\boldsymbol{f}^i$, respectively [72]. In more detail, the state vector $\boldsymbol{x}^i$ is represented as a linear combination of the previous observation and action sequences

$$\boldsymbol{x}_t^i = [\boldsymbol{I}_{n^i} \ \ \mathbf{o}] \boldsymbol{M}^i \boldsymbol{p}_t^i, \tag{3.3}$$

where $\boldsymbol{M}^i \in \Re^{l^i(m+q^i) \times l^i(m^i+q)}$, n^i and $\boldsymbol{I}_{n^i}$ denote the memory function calculated by CVA, the dimension of the state vector and the identity matrix $(n^i \times n^i)$, respectively. Each element estimated by each *LPM* has a contribution-value μ with respect to future prediction,

$$\boldsymbol{x}_t^i = \begin{bmatrix} x_{t,1}^i \\ x_{t,2}^i \\ \vdots \\ x_{t,n^i}^i \end{bmatrix} \begin{matrix} \cdots & \mu_1^i \\ \cdots & \mu_2^i \\ & \vdots \\ \cdots & \mu_{n^i}^i \end{matrix} \tag{3.4}$$

where μ^i $(i = 1, \cdots, n^i)$ are between 0 and 1.

All together, the problem which the *LPM* has to solve is summarized as follows:

From the sequences of the observation and action $\{\boldsymbol{y}^i, \boldsymbol{u}\}$, the learner has to determine the historical length l^i and the dimension n^i and to estimate state vector $\boldsymbol{x}^i$. In addition, four parameter matrices $\boldsymbol{A}^i$, $\boldsymbol{B}^i$, $\boldsymbol{C}^i$, $\boldsymbol{D}^i$ should be calculated.

If n^i and l^i is given, this problem can be formulated as a generalized singular value decomposition. We show the detailed calculation of the matrix $\boldsymbol{M}^i$ in Appendix A, where we follow the explanation by Larimore [36] and van Overschee and De Moor [72].

3.2.3 Determination of the Parameters in the LPM

It is important to decide the dimension n^i of the state vector $\boldsymbol{x}^i$ and lag operator l^i that tells how long the historical information is related to size determination of the size of the state vector when we apply CVA to the classification of agents. Although the estimation is improved if l^i is larger and larger, much more historical information is necessary. However, it is desirable that l^i is as small as possible with respect to the memory size. In addition, the dimension n^i is also as small as possible because the expected learning time for the reinforcement learning algorithms such as Q learning is exponential in the size of the state space [74]. Therefore, we have to take a trade-off between the prediction error and the memory.

It is necessary for the learner to estimate the state vector which satisfies Markovian assumptions. Therefore the estimated state vector should guarantee minimum precision of prediction for behavior acquisition at the sacrifice of the memory consumption. Then, we determine the historical length l^i based on the norm of the covariance matrix of error $\boldsymbol{R}^i$. Next, we apply Akaike's Information Criterion (AIC) [1] which is widely used in the filed of time series analysis in order to determine n^i. AIC is a method for balancing precision and computation (the number of parameters). Let the error vector of observation vector $\boldsymbol{y}^i$ be $\boldsymbol{\varepsilon}^i$ and covariance matrix of error be

$$\hat{\boldsymbol{R}}^i = \frac{1}{N_{all} - 2l_i + 1} \sum_{t=l^i+1}^{N_{all}-l^i+1} \boldsymbol{\varepsilon}_t^i \boldsymbol{\varepsilon}_t^{iT}, \tag{3.5}$$

where N_{all} is the number of pairs (observation and action) to be considered to estimate the state vectors. Then $AIC(n^i)$ is calculated by

$$AIC(n^i) = (N_{all} - 2l^i + 1) \log |\hat{\boldsymbol{R}}^i| + 2\lambda(n^i), \tag{3.6}$$

where $\lambda(n^i)$ is the number of the parameters used in Eq.(3.1) when the dimension of the state vector is n^i. The number of parameters If we use n^i order state vector, the number of free parameters including in Eq.(3.1) is calculated by

$$\lambda(n^i) = n^i(n^i + m + q^i) + mq^i. \tag{3.7}$$

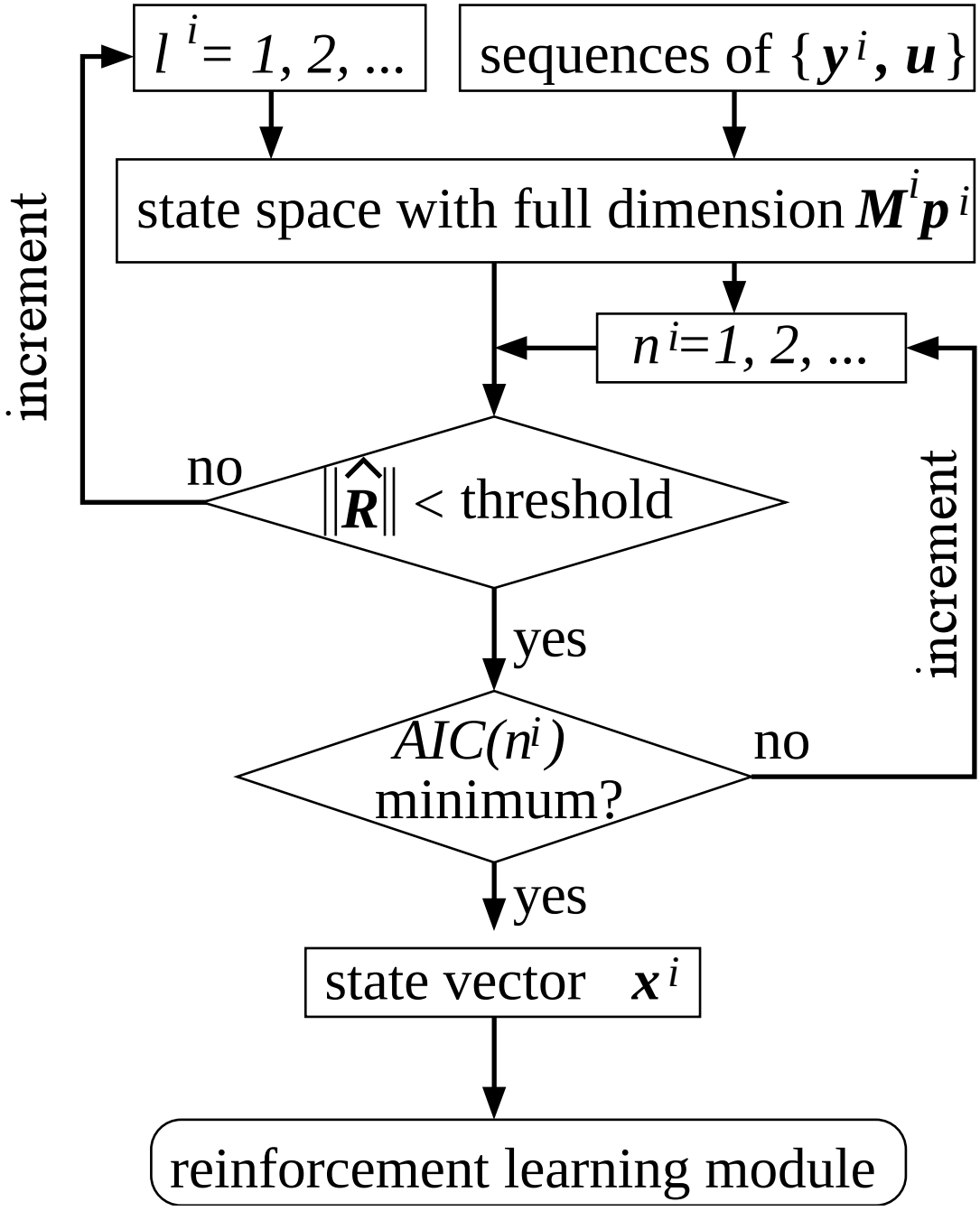


Fig.3.3 Flowchart of the *LPM* to determine the historical length and the dimension of the state vector

We regard the optimal dimension n^{i*} as the dimension which minimize $AIC(n^i)$. **Fig.3.3** illustrates a flowchart of processing data in the *LPM*. For the sequences of observation and action $\{\boldsymbol{y}^i,\ \boldsymbol{u}\}$, each *LPMs* determines the historical length l^i and the dimension of the state vector n^i as follows:

1. Initialize $l^i = 1$ and $n^i = 1$.
2. For l^i, repeat until l^i and n^i are determined.
   (a) Construct the subspaces, and calculate the memory matrix $\boldsymbol{M}^i$ by Eq.(A.9) from the sequences $\{\boldsymbol{y}^i,\ \boldsymbol{u}\}$.
   (b) For the current l^i and n^i ($n^i = 1, \cdots q^i l^i$), calculate the covariance matrix of error $\boldsymbol{R}^i$ by Eq.(3.5).
   (c) If $||\boldsymbol{R}^i|| > threshold$, increment l^i ($l^i \leftarrow l^i + 1$), and return to step (a). Otherwise, go to step (d).
   (d) Determine the optimal dimension n^{i*} by

$$n^{i*} = \arg\min_{n^i} AIC(n^i), \quad n^i = 1, \cdots q^i l^i$$

   then the procedure is finished.

We can regard the roles of two parameters l^i and n^i as follows. The parameter l^i copes with the lack of the current sensor outputs for prediction while the order estimation (n^i) eliminates the redundant representation.

3.3 Integration of the LPMs and Reinforcement Learning

3.3.1 State Space Construction for Reinforcement Learning

Reinforcement learning algorithms improve their performance on tasks from delayed rewards and punishments given by the environment. They are distinguished from supervised learning in that they have no teacher that tells the agent the correct answer to a situation. A reinforcement learning module receives the estimated state vectors from the *LPMs* as described in Section 3.2, and learns the relationships among them. Reinforcement learning algorithms such as Q learning find a policy that maximize discounted sum of the reward received over time,

$$v(\boldsymbol{x}_t) = E\left\{\sum_{n=0}^{\infty} \gamma^n r_{t+n} \middle| \boldsymbol{x}_t = \boldsymbol{x}\right\}, \tag{3.8}$$

where r_t is the reward received at step t given that the agent started in state x and executed policy f. γ ($0 \leq \gamma \leq 1$) is the discounting factor. An action value function $Q(\boldsymbol{x}, \boldsymbol{u})$ is

defined in terms of the state $\boldsymbol{x} \in \boldsymbol{X}$ and action $\boldsymbol{u} \in \boldsymbol{U}$, and updated incrementally based on the reward r_t from the environment.

In general, the reinforcement learning represents the action value function by

1. Look-up table :
   The action value function is represented by a look-up table, where continuous state and action spaces have to be quantized appropriately. In this representation, Q learning has been shown to converge with probability 1 to optimal solution under the Markovian decision processes. However, there is a severe problem. That is, this representation needs not only the large memory for tables but also the time and data to fill them precisely. This is so called the "curse of dimensionality."

2. Function approximator :
   A straightforward approach to the curse of dimensionality is to replace the lookup table with a generalizing function approximator. For example, the action value function is approximated by CMAC [65], neural network [37] and other approximation method [10]. In this representations, reinforcement learning can be applied to larger problems with higher dimensional and continuous state spaces, and we may expect some generalization for unknown states. However, this representation sometimes leads to local minimum.

In this chapter, we choose look-up table representation because we can check the results easily. Then we show how the continuous state vector is quantized. Because the covariance matrix of the state vector $\boldsymbol{x}^i \in \Re^{n^i}$ estimated by Eq.(3.3) is a unit matrix, that is, $\boldsymbol{\Sigma}_{xx} = \boldsymbol{I}_{n^i}$, we segment the each element $x_{i,j}$ $(j = 1, \cdots n)$ of the estimated state vector $\boldsymbol{x}^i$ to 3 sub-states as follows:

$$x_j^i < -1, \quad -1 \leq x_j^i < 1, \quad 1 \leq x_j^i. \tag{3.9}$$

Furthermore, the learner has to consider the case where the target object can not been observed. As a result, the total number of discrete states is $3n^i + 1$ taking an invisible situation into account when the state vector $\boldsymbol{x}^i \in \Re^{n^i}$ estimated by the LPM^i is quantized. Hereafter, we describe the combined and quantized state vector ${}^d\boldsymbol{x} \in \Re^N$ as

$${}^d\boldsymbol{x} = [{}^dx^1, \cdots, {}^dx^N]^T,$$

where ${}^dx^i$ denote the discrete state about the i-th object.

Since we segment the state space in a systematic manner mentioned above (Eq.3.9), one action does not always corresponds to one state transition. This problem is called "state action deviation problem" by Asada *et al.* [5]. Because of this problem, the robot frequently returns to the same state. which prevents the learning from converging correctly. Therefore, the robot continues to take one action primitive until the current state changes [5]. This sequence of the action primitive is called "an action." Once the state has changed, we update the action value function. Based on the ${}^d\boldsymbol{x}$, we utilize the modified reinforcement learning method [70] which can coordinate multiple behaviors taking account of a trade-off between the learning time and the performance (see Appendix B.2).

3.3.2 Modification of the Action Value Function according to the Change of the LPMs

After the agent moves around for a while, the LPM^i calculates the memory matrix $\boldsymbol{M}^i$ again and updates the dimension of the state vector. Therefore, the learning robot has to modify the action value function according to the change of the *LPMs*. In general, it seems difficult to acquire a function which maps the action value function to another one based on different state representation. However, it is not efficient to learn the behavior from the beginning while the previous learning results are cast off. Then, we reuse the learned policy as initial knowledge to reduce the learning time.

Let ${}^d\boldsymbol{x}_k$, f_k, Q_k be the estimated state vector, the optimal policy and the action value function at the k-th phase. We define the optimal mixture policy at the k-th phase taking account of a trade-off between the current action value function and the previous one by

$$f'_k({}^d\boldsymbol{x}_k) = \begin{cases} f_{k-1}({}^d\boldsymbol{x}_{k-1}) & \text{with probability } (1-\beta), \\ f_k({}^d\boldsymbol{x}_k) & \text{with probability } \beta, \end{cases} \tag{3.10}$$

where β $(0 \leq \beta \leq 1)$ is a balancing probability. and $f_k({}^d\boldsymbol{x}_k)$ is a pure optimal policy function calculated by

$$f_k({}^d\boldsymbol{x}_k) = \arg\max_{\boldsymbol{u}' \in \boldsymbol{U}} Q_k({}^d\boldsymbol{x}_k, \boldsymbol{u}'), \tag{3.11}$$

Eq.(3.11) is the definition of the optimal policy in the standard Q learning algorithm. Because we have to vary β from 0 to 1 gradually, β can be increased like

$$\beta = \frac{\text{trial} - 1}{N_{all} - 1}, \quad \text{trial} = 1, \cdots N_{all}.$$

In fact, the learning robot has to select several actions to explore the unknown situations, we use ϵ-greedy policy [66], meaning that most of the time the learning robot chooses an action that has maximal estimated action value by Eq.(3.10), but with probability ϵ it instead selects an action at random.

3.4 Learning Schedule for a Multiagent Environment

The actual learning process can be categorized into three ways.

1. Learning the policy in a real environment [40, 68]:
   Except an easy task in a simple environment, it seems difficult to implement. One alternative is teaching which can reduce the search space.

2. Learning the policy in computer simulation and transferring the policy to a real environment [5]:
   Since there is still a gap between the simulation environment and the real one, we need some modification in the real experiment.

3. Combination of computer simulation and real experiments [46]:
   Based on the simulation results, learning in a real environment is scheduled. This corresponds to the case that the teacher is not a human designer but a simulated learning agent. It become important to find differences between the computer simulation and real experiments to fill the gap.

We adopt the third one. However, in multiagent environments, there are uncertainties of state transition due to unknown policies of other learning agent. If the multiple robots learn the behaviors simultaneously, the learning process may be unstable, especially in the early stage of learning. Therefore, we need a method which can stabilize the learning processes especially in the early stage of learning. **Fig.3.4** shows a basic idea of the proposed learning schedule. At first, the designer selects one learning agent at random from multiple learning agents. Unselected agents do not update its own action value function and move around based on the policy which is previously acquired (the learner-i is selected in Fig.3.4). Therefore, the selected agent is the only agent that can choose an action freely in the environment at that time. After the selected agent has finished its learning, the designer changes the next agent to be learned. We repeat this for robots to acquire the purposive behaviors.

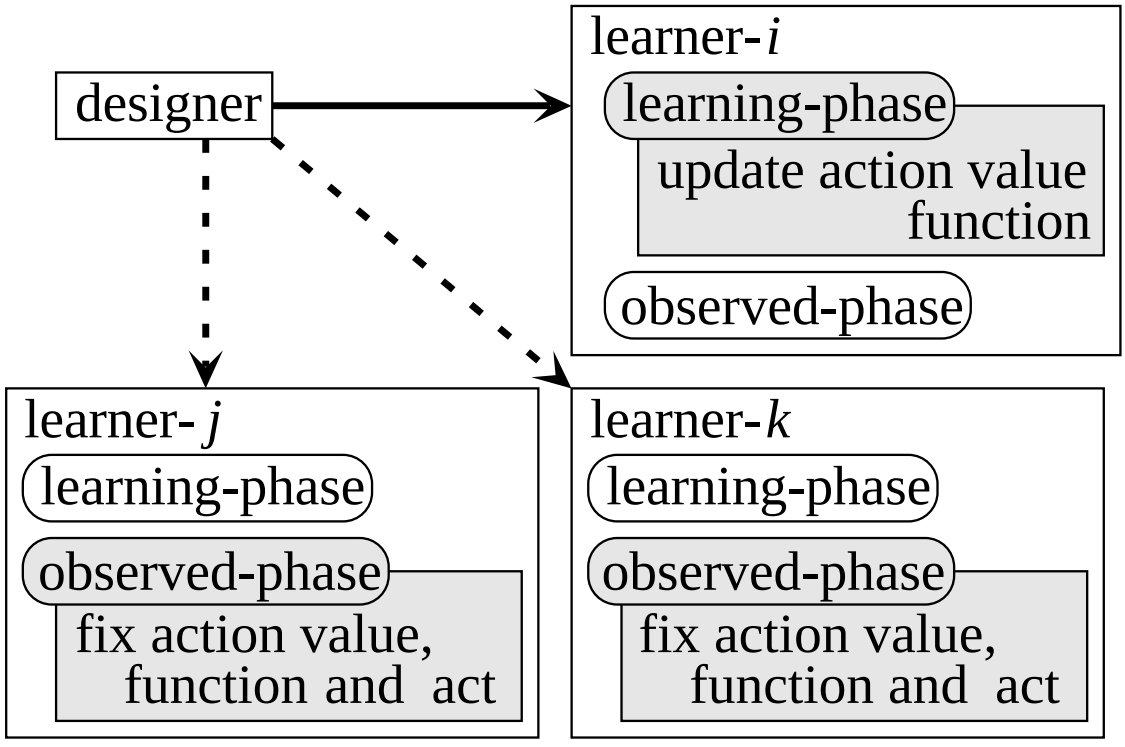


Fig.3.4 Schedule for efficient learning in multiagent environments

3.5 Task and Assumptions

3.5.1 Environment and Robots

We apply the proposed method to a simplified soccer game shown in **Fig.3.5** in the environment including two learning mobile robots (passer and shooter). The environment consists of a ball and a goal, and a wall is placed around the field except the goal. The sizes of the ball, the goals and the field are the same as those of the middle-size real robot league in the RoboCup Initiative [21].

Each robot has a single color TV camera and does not know the locations, the sizes and the weights of the ball and the other agent, any camera parameters such as focal length and tilt angle, or kinematics/dynamics of itself. The task for the shooter is to shoot a ball into the goal while the task for the passer is to pass a ball to the shooter. Furthermore, both robots have to avoid collisions with each other as possible.

3.5.2 Observation and Action Spaces

The effects of an action against the environment can be informed to the agent only through the visual information. The output (observation) vectors are shown in **Fig.3.6**. The appropriate state vectors can not be constructed if the set of selected image features is poor. Therefore, we utilize the fundamental image features as many as possible for observation vectors. In more detail, we select following image features:

- ball : the center position (x, y), the radius, and the area in the image plane,
- goal : the center position (x, y), the positions of four corners (x_i, y_i) $(i = 1, \cdots, 4)$ and the area, and
- other robot : the center position (x, y), the width w, the height h and the area.

As a result, the dimensions of the observed vector about the ball, the goal, and the other robot are 4, 11, and 5, respectively. Due to the limitation of our image processing unit, the size of an image is reduced to 64×60 pixels and the resolution in our simulator is adjusted.

Our mobile robots move around using a 4-wheel steering system shown in Fig.3.5. As motor commands, each mobile robot has a two degree of freedoms. The input $\boldsymbol{u}$ is defined as the 2 dimensional vector:

$$\boldsymbol{u} = \begin{bmatrix} v \\ \phi \end{bmatrix}, \quad v, \phi \in \{-1,\ 0,\ 1\}, \tag{3.12}$$

where v and ϕ are the velocity of motor and the angle of steering, respectively and both of which are quantized. Since two pairs of $(v, \phi) = (0, \pm 1)$ are no physical meaning, we eliminate them. All together, the robot can select seven actions.

Fig.3.5 Two robots and the environment

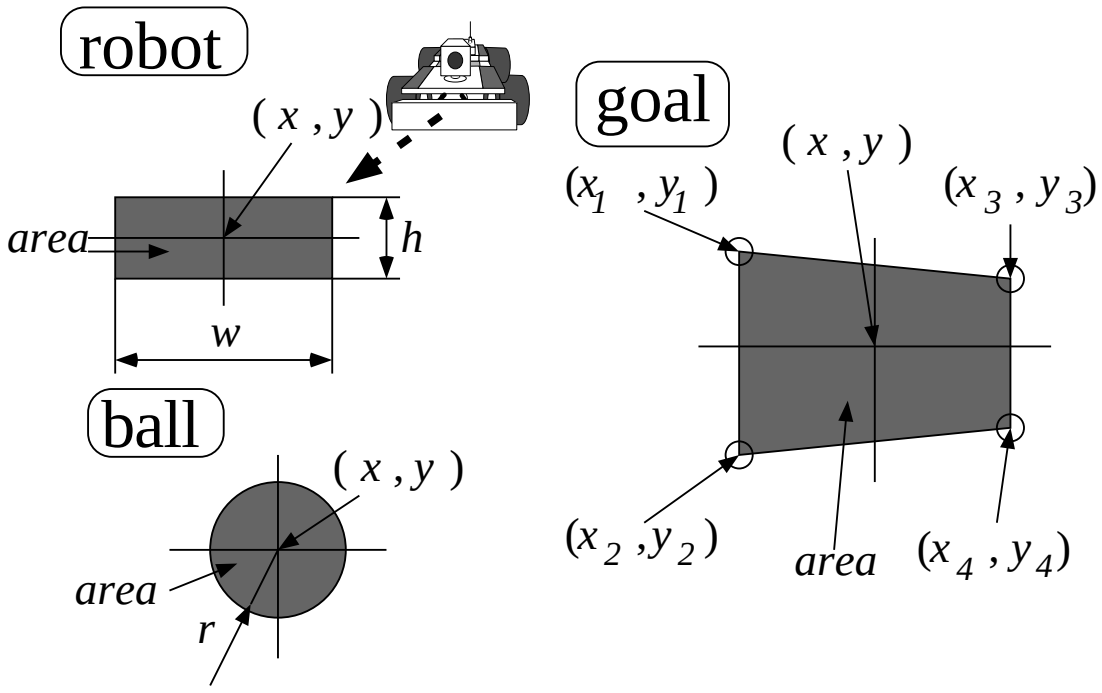


Fig.3.6 Image features of the ball, goal, and other robot

Due to the peculiarity of visual information, that is, a small change near the observer results in a large change in the image, one action does not always correspond to one state transition, that is, "state-action deviation problem" [5] occurs as described in Section 3.3.1. To avoid this problem, we reconstruct the action space as follows. Each action defined above is regarded as an action primitive. The robot continues to take one action primitive at a time until the current state changes. This sequence of the action primitive is called "an action."

3.5.3 Experimental Setup

Fig.3.7 shows a configuration of the real mobile robot system. The image taken by a TV camera mounted on the robot is transmitted to a UHF receiver and processed by Datacube MaxVideo 200, a real-time pipeline video image processor. In order to simplify and speed up the image processing time, we painted the ball, the goal, and the opponent red, blue, and yellow, respectively. The input NTSC color video signal is first converted into HSV color components in order to make the extraction of the objects easy. 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 tilt angle is about -26 [deg] so that robot can see the environment effectively. The horizontal and vertical visual angle are about 67 [deg] and 60 [deg], respectively.

At first, each learning robot constructs the state vectors by the *LPMs* and starts to learn purposive behaviors in computer simulation. One trial is terminated if (1) one of the robots shoots a ball into the goal, (2) two robots make a collision, (3) one of the robots or the ball goes out of the field, or (4) the pre-specified time interval expires. When a pass behavior is achieved[1], the trial still continues. We assign a reward value 1 when the shooter shoots a ball into the goal and the passer passes the ball toward the shooter. Further, a negative reward value -0.3 is given to the robots when a collision between two robots is happened. We set the probability ϵ as 20%, which means that the random action is selected with probability 20 %.

We set up the learning schedule as follows:
Computer simulation:

- **period A** (trial number $0 \sim 250 \times 10^2$) : The passer is selected. At first the passer constructs three state vectors (of the goal, the ball, and the shooter), and then improve its behavior. The shooter is stationary because it has no learned policies. As a result, the shooter does not store data for construction of the state vectors and improve the action value function.

- **period B** (trial number $250 \times 10^2 \sim 500 \times 10^2$) : The shooter starts to learn the shooting behaviors while the passer acts based on the acquired behaviors during the

[1]In this chapter, in case one of the robots pushes the ball which rolls back from the other robot in a short time, we judge that a pass behavior is achieved.

period A. In this period, the passer store the latest N_{all} sequences of observation and action.

- **period C** (trial number $500 \times 10^2 \sim 750 \times 10^2$) : The passer modifies the *LPMs* based on the collection data in the period B, and then update the action value function again. The shooter behaves based on the result of period B.
- **period D** (trial number $750 \times 10^2 \sim 1000 \times 10^2$) : From the end of the period C, the shooter learns in the same way.

Real experiment:

- **period E** (trial number $0 \sim 100$): We transfer the results of computer simulation to the real environment. During this period, both of the shooter and the passer
  1. update the *LPMs* based on the result of the period E,
  2. collect the latest N_{all} sequences of observation and action, and
  3. improve the action value functions[2] based on the obtained real data.
- **period F** (trial number $100 \sim 150$) : Both robots update the *LPMs* based on the data collected during period E. Both robots perform behavior learning.

After all, the learning schedule can be summarized in **Table 3.1**. In the computer simulation, first four period (from period A to D) are regarded as one set of experiments. We perform 10 sets of experiments. Then, we transfer the best results among 10 sets in the computer simulation to real experiments.

Table 3.1 Learning schedule in this experiment

phase	passer		shooter	
	the *LPM*	action value	the *LPM*	action value
A	store and update	update	do nothing	fixed
B	store	fix	store and update	update
C	update	update	store	fix
D	store	fix	update	update
E	update and store	update	update and store	update
F	update and store	update	update and store	update

[2]In the real experiments, the action value function can not be updated at on-line fashion due to the limitation of the experimental system. Therefore, we update the action value function after one trials ends.

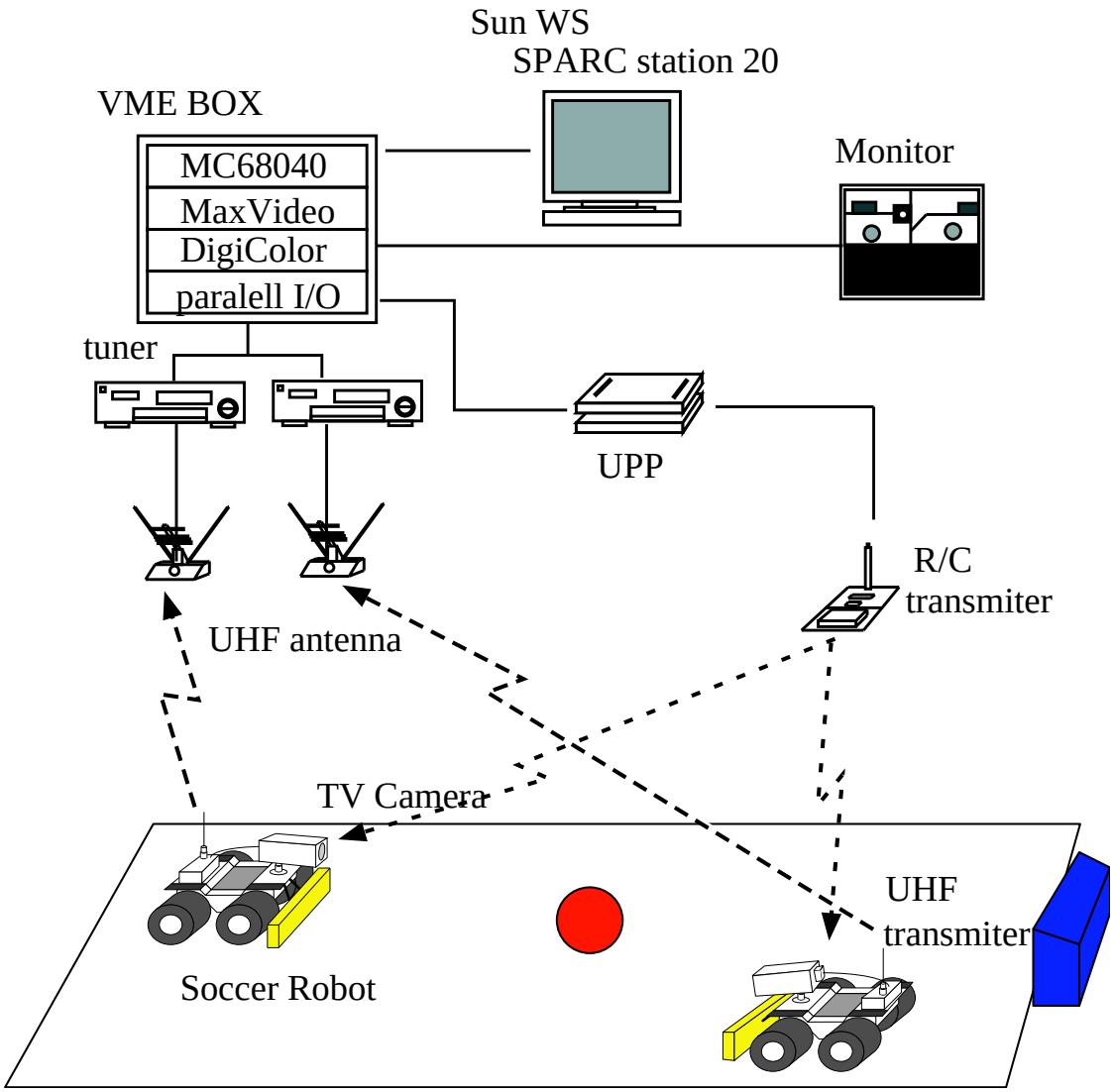


Fig.3.7 A configuration of the real system.

3.6 Experimental Results

3.6.1 Estimated Dimensions and Historical Length

We show the result of the estimated dimension n of the state vector and historical length l in computer simulation and real experiments in **Table 3.2** where we represent the results in the order of the experiments (simulation → simulation → real experiments). **Figs.3.8** (a), (b) and (c) show the sequences of prediction error (of center position y) of the ball, the goal and the passer from the viewpoint of the shooter during the period F. From Table 3.2 and Fig.3.8, we can conclude the following things.

- goal : Because the goal is stationary object, the learner (shooter) can estimate the interaction easily. As a result, the prediction error is the smallest among the objects. The dimension of the estimated state vector of the goal is smaller than that of the image feature vector. The prediction error become large at 15 sec because the goal is almost occluded by the ball (for example, see **Fig.3.12 (4)**).

- ball : Because the ball does not move by itself, the interaction with the learner is slightly simple. In addition, the ball is seldom occluded by other objects.

- passer : The interaction is the most complicated for the shooter in this experiment because the passer can move by itself. The reasons why the prediction error is large are that (1) the observation vector is not sufficient, (2) the *LPM* cannot represent the relationship between two or more objects in the environment, and (3) the passer is often occluded by the ball.

Next, we show the contributions of each element of the estimated state vectors for prediction in **Fig.3.9** by which we may conclude:

- goal : The image features of the goal directly correlate with learner's action because the goal is stationary. In this case, the order reduction procedure by AIC (Eq.(3.6)) is effectively performed.

- ball : In the selected four elements of the state vector, first two elements are dominant to predict the movement of the ball. The remainder has a influence at the ratio of about 60 or 70 percent as compared with the first two elements.

- other robot : As compared with the goal and the ball, the values are decreased gradually. The prepared image features of the other robot is not sufficient although the other robot can move actively.

We suppose the reasons why the estimated orders of state vectors and historical length are different between computer simulation and real experiments as follows:

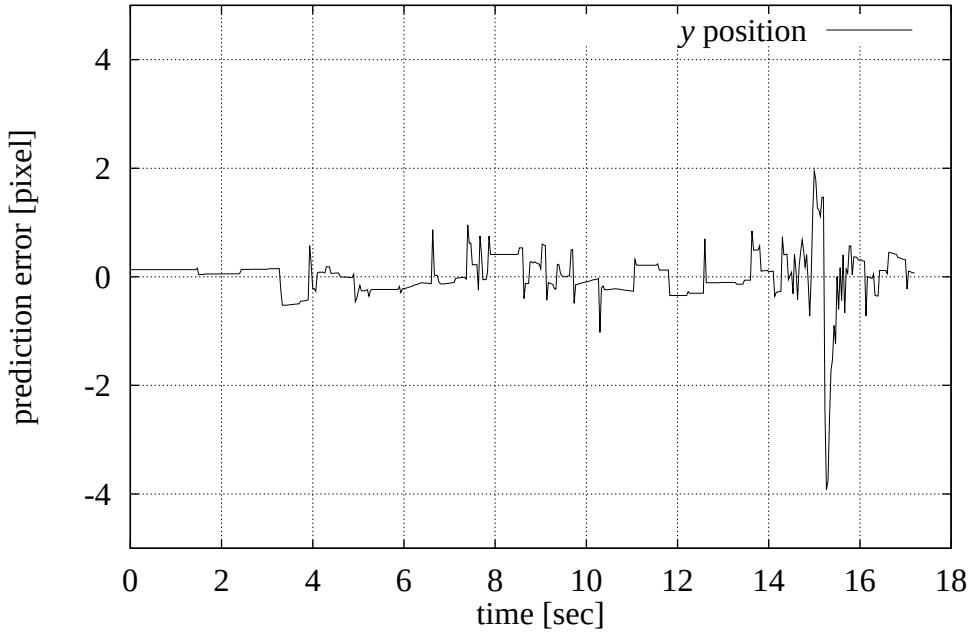
1. Because there are the noise of sensor and the uncertainties in actuator outputs, the prediction error of real experiments is much larger than that of computer simulation. As a result, precision does not improve in spite of the increase of the order of the estimated state vector.

2. In order to collect the sequences of observation and action, the robots do not select the random action but move based on the result of computer simulation. Therefore, the experiences of passer and shooter are biased.

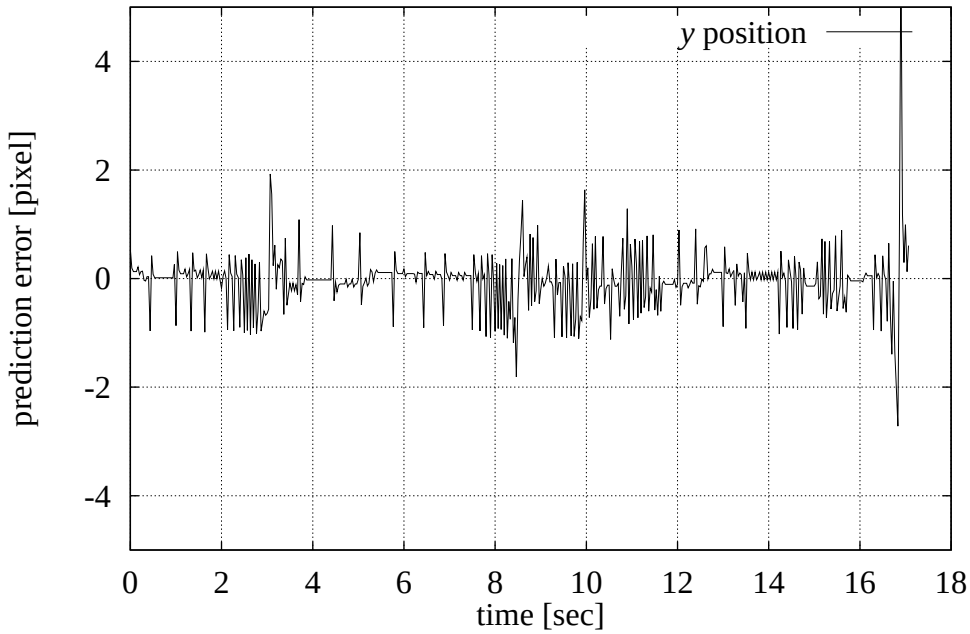
As a result, the historical length of the real experiments tends to become larger than that of the computer simulation. On the other hand, the estimated order of state vector for the other robot of real experiments is smaller than that of computer simulation since the components for higher and more complicated interactions can not be discriminated from noise in the real environments.

Table 3.2 Differences of the estimated dimension (simulation → simulation → real experiments)
† : The passer does not observe the goal after the pass behavior is acquired in this experiment. As a result, the *LPM* does not updated except for the first period A.

observer	target	estimated dimension (order)	historical length
shooter	goal	$2 \rightarrow 2 \rightarrow 3$	$1 \rightarrow 1 \rightarrow 1$
	ball	$4 \rightarrow 4 \rightarrow 4$	$2 \rightarrow 2 \rightarrow 4$
	passer	$6 \rightarrow 6 \rightarrow 4$	$3 \rightarrow 3 \rightarrow 5$
passer	goal†	3	2
	ball	$4 \rightarrow 4 \rightarrow 4$	$2 \rightarrow 2 \rightarrow 4$
	shooter	$5 \rightarrow 5 \rightarrow 4$	$3 \rightarrow 3 \rightarrow 5$

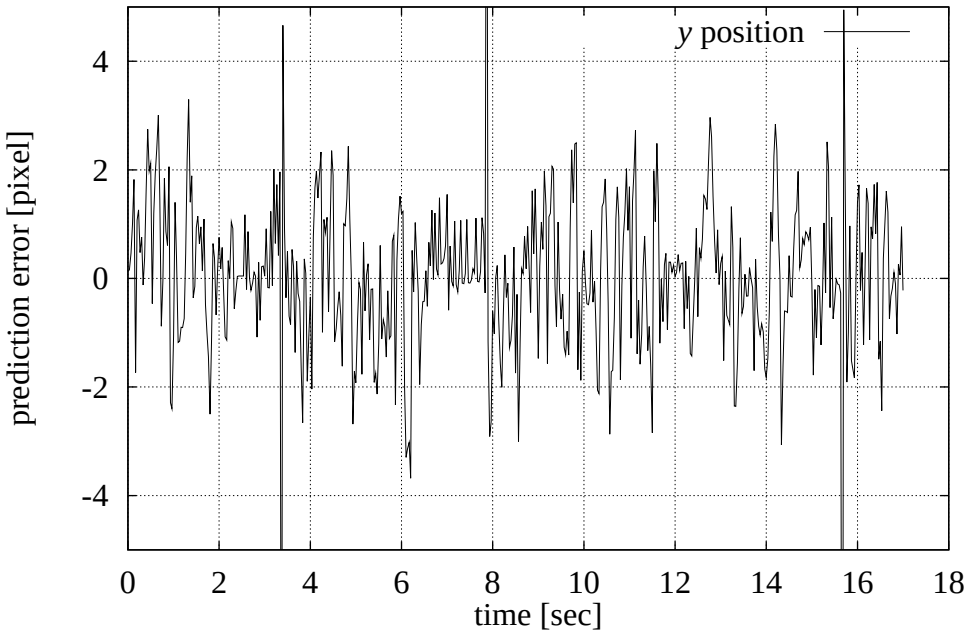


(a) y position of the goal image



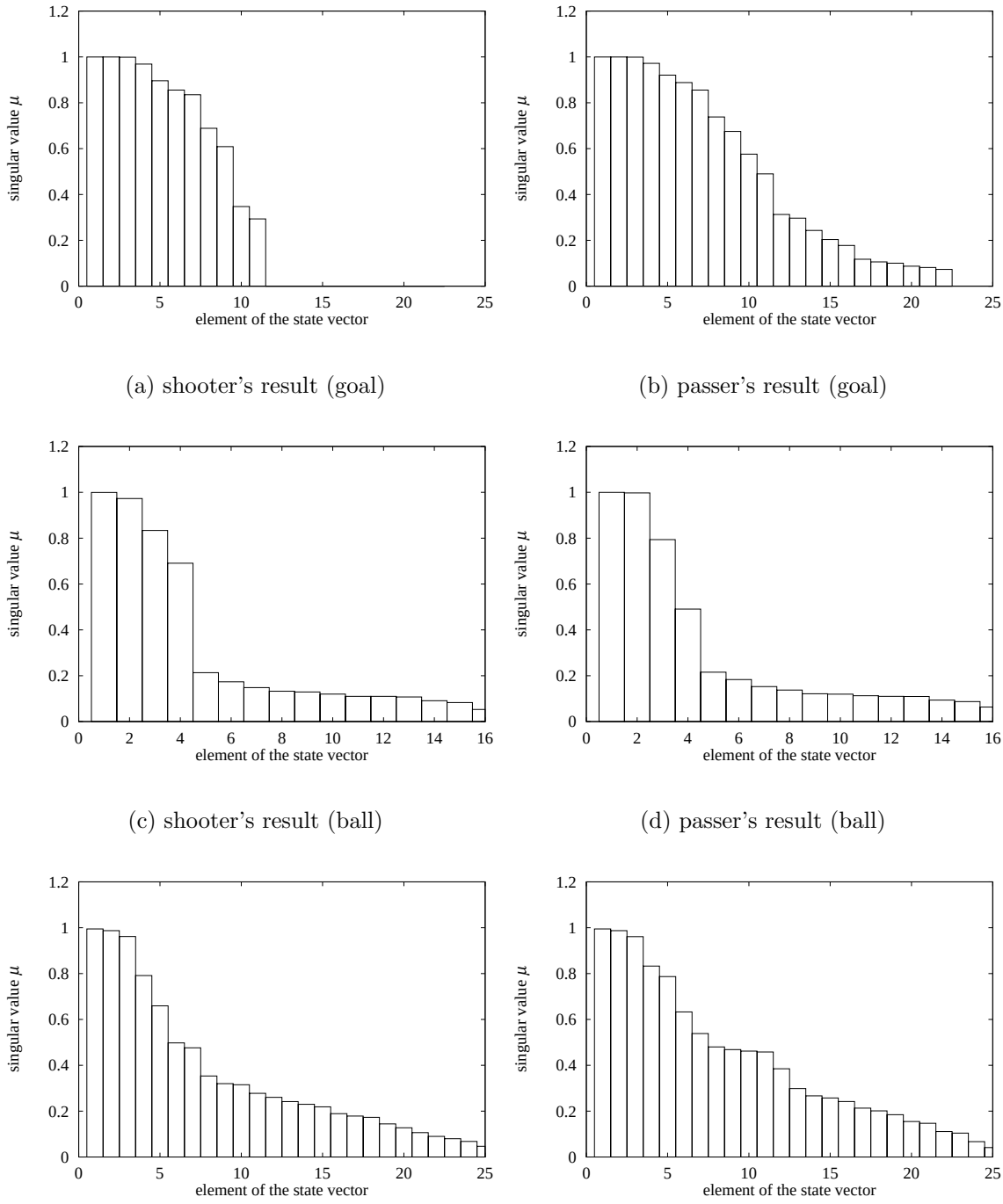
(b) y position of the ball image

Fig.3.8 Prediction error in the real environment



(c) y position of the other robot image

Fig.3.8 (continued) Prediction error in the real environment



(a) shooter's result (goal)

(b) passer's result (goal)

(c) shooter's result (ball)

(d) passer's result (ball)

(e) shooter's result (passer)

(f) passer's result (shooter)

Fig.3.9 Acquired singular values in the real environment. the number of elements is $l^i q^i$.

3.6.2 Obtained Performance

Next, we discuss the success rates of the acquired shooting and passing behaviors in computer simulation and real experiments. Success rate of each behavior is calculated by

$$\text{success rate} = \frac{\text{number of achievement}}{\text{number of trials}}. \tag{3.13}$$

We show the transition of the success rates of the proposed method and the previous method (state space is constructed by only sensor information) [70] in **Figs.3.10** (a) and (b).

- **period A** : During this period, the passer is the only agent that can act in an environment. In this situation, the task for the passer is to kick the ball to the stationary shooter. As a result, the main difference between the proposed method and the previous one is the representation about the ball. As you can see from Fig.3.10, the proposed method acquired better behaviors than the previous method. The reason is that the received reward is distributed over the large area of the state space.

- **period B** : The important differences between the period A and B is the existence of the other robot that can make a state transition. Both of the methods decrease their success rates. The shooter using the proposed method acquires the appropriate representation about the passer while the one using the previous method shoots the ball into the goal regardless the passer behaviors. As a result the performances of the passer are quite different.

- **period C** : The passer starts to learn behaviors again. Great differences between two methods are revealed after the passer moves in order to learn passing behaviors in the period C. Both of the success rates become stable. However, the passer improves its behaviors while the success rate of shooting behavior decreases again based on the previous method.

- **period D** : Again, the performances based on the previous method oscillate while the success rates of the proposed method are improved gradually.

Therefore, we can conclude that the *LPM* is an effective tools to acquire cooperative behaviors based on the reinforcement learning in a multiagent environment.

Table 3.3 shows the result in the real environment. The performances are improved as compared with the simple transfer of the results of computer simulation. We suppose that the reason why two robots successfully learn behaviors simultaneously in the real experiments as follows. Each robot has behaviors of computer simulation with high performance and selects optimal action with probability 80 %, it can concentrate on modification of the *LPMs* and the action value function.

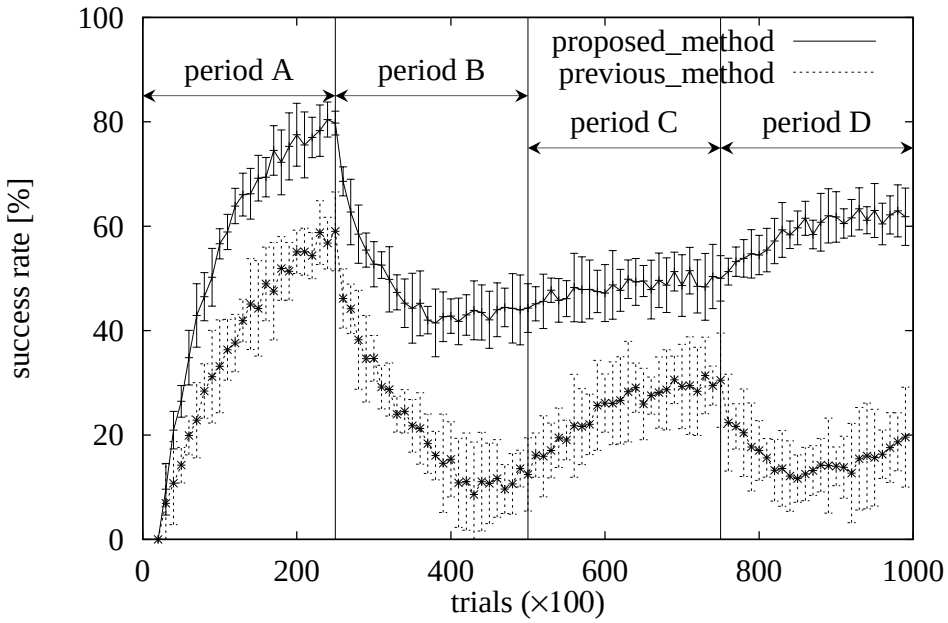
We checked what happened if we replace the *LPMs* between the passer and the shooter. Eventually, large prediction errors of both sides were observed. The reason why they

acquire the different representation of the same objects is that there are differences in several factors such as tilt angle, the angle of the steering and so on. Furthermore, their experience are quite different because they have different tasks, respectively. Therefore, the *LPMs* can not be replaced directly between physical agents even if they have the same body (homogeneous agents). This causes the following new research issues:

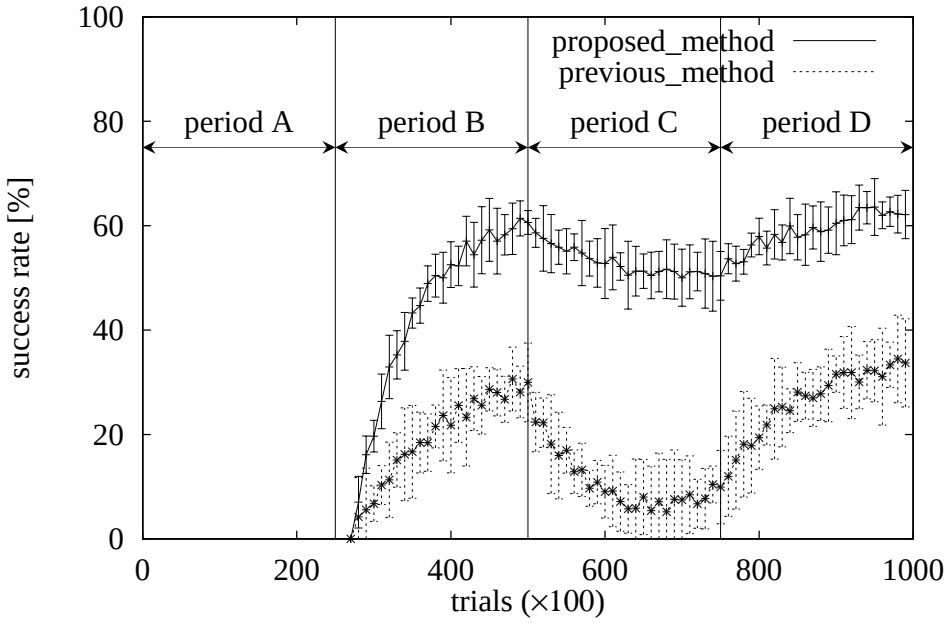
1. How the agent share their internal representation to cooperate?

2. How to cooperate between heterogeneous agents?

Finally, we show the acquired behaviors by the proposed method. **Fig.3.11** and **Fig.3.12** show a sequences of the cooperative behaviors in computer simulation and real experiments. At the beginning, the passer kicked the ball toward the shooter, which shot it into the goal. After the passer kicked the ball, it went backward so as to avoid collisions with the shooter in Fig3.12 (a).



(a) passing behavior



(b) shooting behavior

Fig.3.10 Success rates in computer simulation with the proposed scheduling method

Table 3.3 Performance results in real experiments

	period E	period F
success rate of shooting	57/100	32/50
success rate of passing	30/100	22/50
number of collisions	25/100	6/50

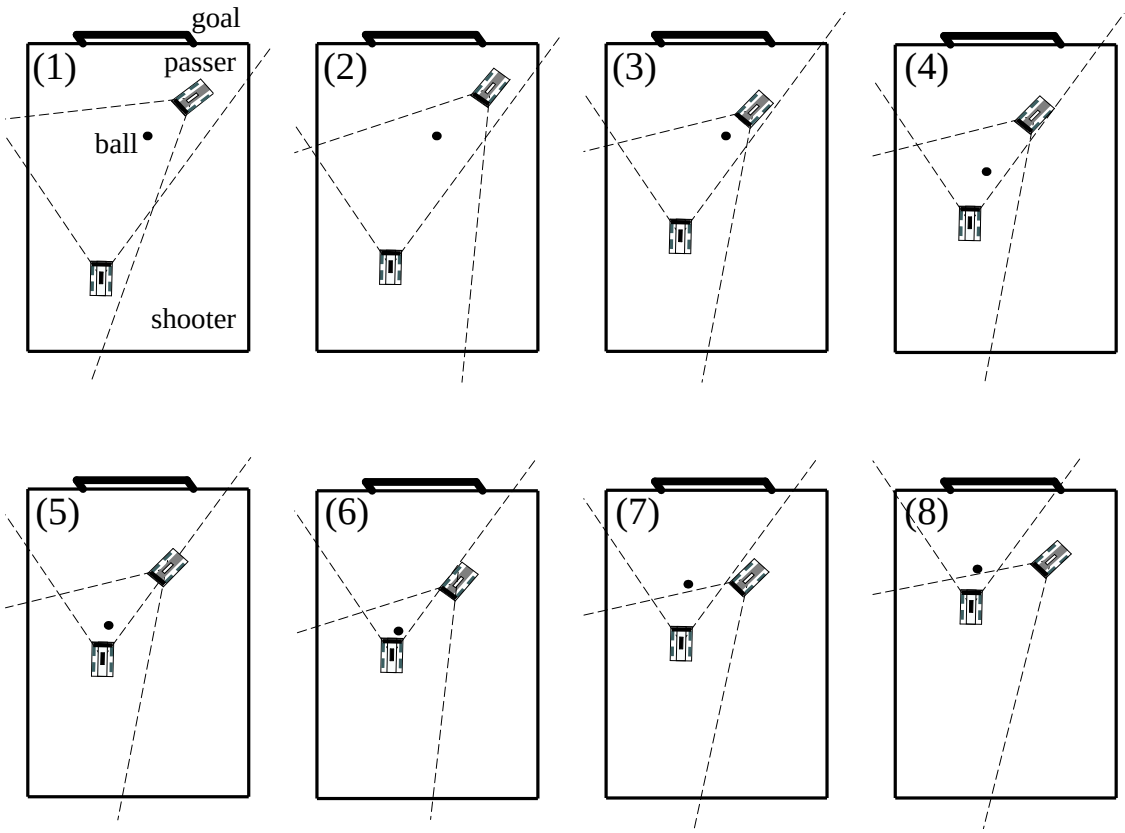
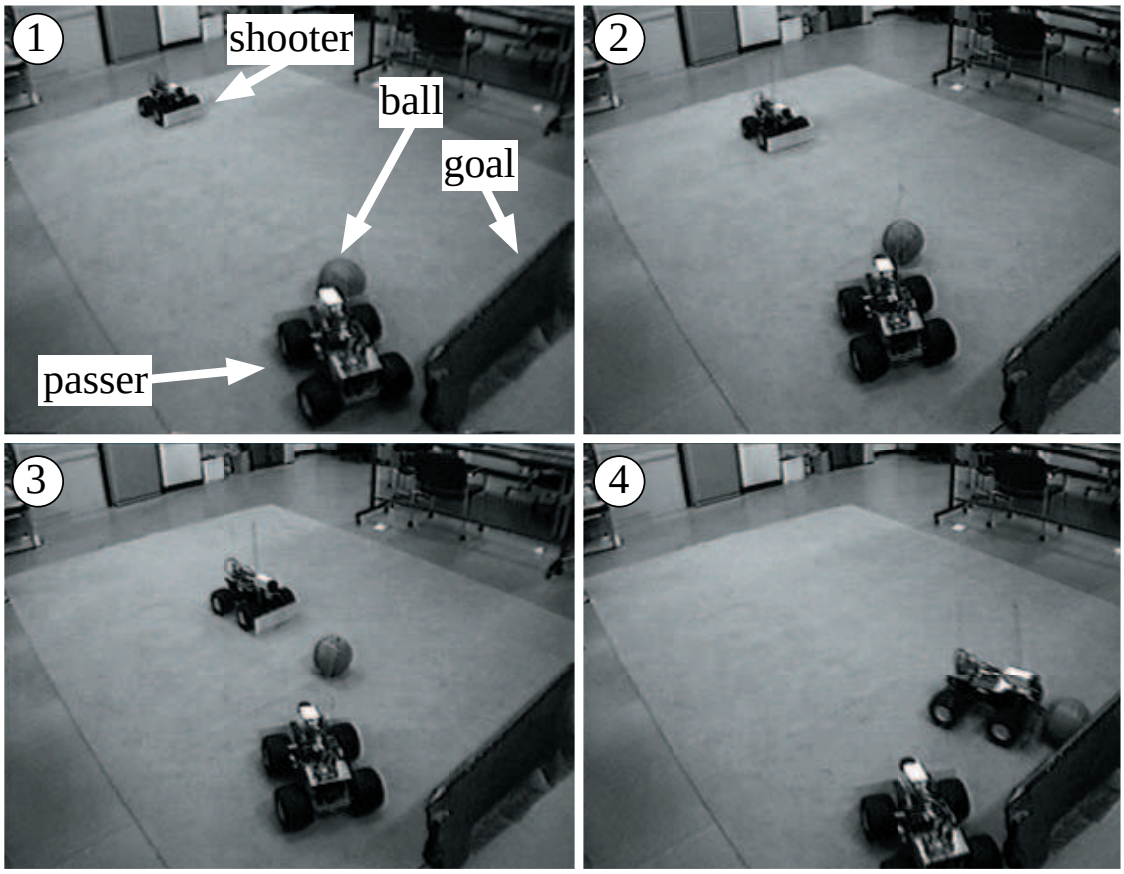
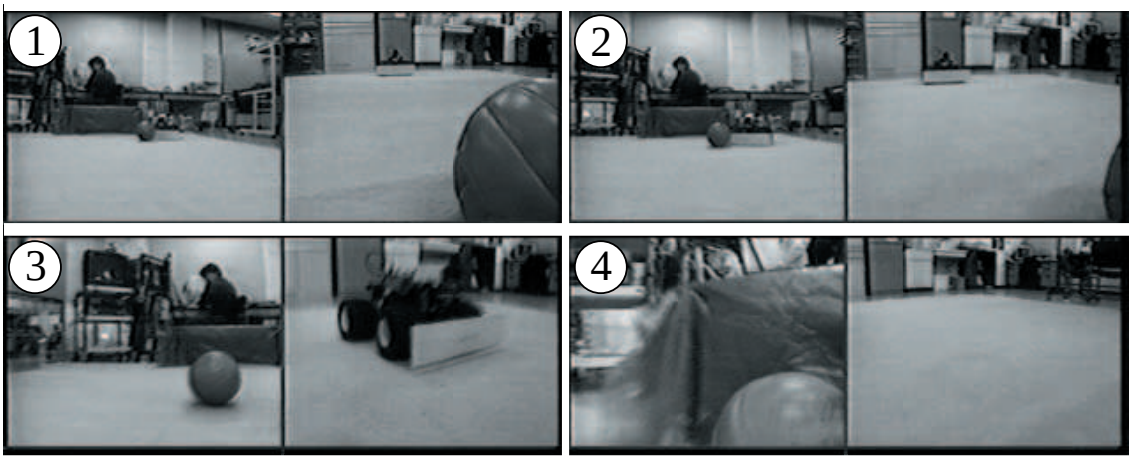


Fig.3.11 Acquired cooperative behavior in computer simulation



(a) top view



(b) obtained images (left: shooter, right: passer)

Fig.3.12 Acquired cooperative behavior in the real environment

3.7 Discussion and Future Works

This chapter shows how the *LPM* has an important role to acquire the cooperative behaviors based on the reinforcement learning in the multiagent environments. The *LPM* estimates the local interactions between the learner and the other objects through the sequences of observation and action while the reinforcement learning module copes with the global interactions among the *LPMs* and the given tasks. The *LPM* can reduce the ambiguity of the behaviors of other robots successfully. Our method takes account of the tradeoff among the precision of prediction, the dimension of state vector, and the length of steps to predict. We apply the proposed method to a simplified soccer game including two mobile robots, and demonstrate that two robots realize cooperative behaviors such as passing behavior, shooting behavior, and avoiding behavior even if the ball is rolling well.

Spatial quantization of the image into objects has been easily solved by painting objects in single color different from each other. Rather, the organization of the image features and their temporal segmentation for the purpose of task accomplishment have been done simultaneously by the proposed method. Each agent does not have explicit procedures how to cooperate, but only has a look up table to behave that has been obtained through the learning process. Therefore, the framework of behavior understanding of other robots through observation and action is needed. Observation and action correspond to message receiving and sending, that is, no explicit communication but the resultant behavior can be regarded as cooperation from a viewpoint of the spectators.

There are three main issues to be considered.

Segmentation Problem

In this chapter, we do not cope with segmentation problem. Constructed state space is quantized by Eq.(3.9) in order to apply the Q learning algorithm. However, there is no guarantee that such a state space is always appropriate for the learning agent. So long as the learning agent utilizes a tabular action value function, the learning agent are haunted by the segmentation problem of the state and action spaces. To overcome this problem, several quantization methods such as Parti game algorithm [48] and Asada's method [5] might be promising.

Exploration and Exploitation

In this experiments, we choose simple ϵ-greedy strategy as action selection. From a view point of the construction of the *LPMs* which represents the model of the dynamics of a certain phenomenon, the strategy of the robot needs to cover the whole space homogeneously. In other words, the inputs vector (motor command) applied to the system (environment) should be *persistently exciting*. However, it is almost impossible to sample the data because the time is limited and undesired inputs causes the destruction of the robots. This problem is the same one discussed as the trade-off between exploration and

exploitation in the field of the reinforcement learning. If the sampled data are biased, the matrices $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$ and $\boldsymbol{D}$ of the Eq.(3.1) changes. For example, the robot seldom kicks the ball, the model of the ball will be regarded as a stationary one.

Generally, for the less biased data, the more data and longer time are necessary. An effective method for data sampling should be developed, but there is a trade-off between the effectiveness and *a priori* knowledge on the environment and the robot.

Non-Linear Problem

The *LPM* utilizes a discrete time, linear, state space model as a representation of interactions. For example, suppose that the agent with omni-directional vision [67] uses the *LPM*.

In order to cope with non-linear problems, two measures are promising. The non-linearity is dealt with by identifying a time-varying system using recursive updating of the model. This corresponds to a local linearization of the nonlinear system. A second possibility is provided by the observation that (mild) nonlinearities do not matter as they can be incorporated in the control design.

In the current system, we consider just two robots, and regard that the current system can cope with global interactions. However, more robots in the field we have, more complicated and higher interactions occur. As future works, we challenge to extend our method when more than two robots learn cooperative and competitive behaviors.

Chapter 4

Environmental Complexity Control to Accelerate Reinforcement Learning

4.1 Introduction

Realization of autonomous agents that organize their own internal structure towards achieving their goals through interactions with dynamically changing environments is the ultimate goal of AI, Robotics, and A-Life. From a viewpoint of designing robots, there are two main issues to be considered:

- the design of the agent architecture by which a robot develop through interactions with its environment to obtain the desired behaviors.
- the policy how to provide the agent with tasks, situations, and environments so as to develop the robot. When the given tasks are too difficult for the learner to accomplish them, the reward is seldom given to the learner.

The former has revealed the importance of "having bodies" and eventually also a view of the internal observer [69]. These are summarized as follows:

1. Sensing and acting are tightly coupled and not separable [22, 23].
2. In order to achieve the goal, the sensor and actuator spaces should be abstracted under the resource bounded conditions (memory, processing power, controller etc.).
3. The abstraction depends on both the fundamental embodiments inside the agents and the experiences (interactions with their environments). The consequences of the abstraction are the agent-based subjective representation of the environment, and its evaluation can be done by the consequences of behaviors [6].

Asada [4] discussed how the physical agent can develop through interactions with its environment according to the increase of the complexity of its environment in the context of a vision-based mobile robot. In this chapter, we put more emphasis on the second issue, that is, how to control the environmental complexity so that the mobile robot can efficiently improve its behaviors.

"Shaping by successive approximation" is a well-known technique in psychology of animal behavior [56]. A simple and straightforward analogy to this situation is to design a reward function to accelerate the reinforcement learning. However, this often requires *a priori* precise knowledge about the details of the relationship between the given task and the environment. Instead of providing such knowledge, an alternative called "Leaning from Easy Missions" (LEM) paradigm was proposed [5] in which the learning time of the exponential order in the size of the state space can be reduced to the linear order according to the precision of the knowledge about the order of easy missions by setting up initial configurations scheduled by the changes of the action values. Based on the concepts of LEM, Asada *et al.* [7] reported that a learner can shoot a ball into the goal avoiding collisions with other robots if the strategy of the other competitive robot is changed from a simple behavior to complicated one gradually. However, the state space is fixed in the methods [5, 7]. Yang and Asada [77] proposed *Progressive Learning* which learns a motion to be learned from slow to fast and apply it to a peg insertion task. Omata [52] applied *Genetic Algorithm* [20] to acquire the neural network controller which can drive a bicycle. The designer give an initial velocity to the bicycle so as to control the bicycle easily. After the generation proceeds, the assist is slightly decreased. This technique makes it easier for the GA to locate the general area where the global optimum lies, at the early stages of the search.

The basic idea of LEM paradigm can be extended to more complicated tasks, but more fundamental issues to be considered are how to define complexity of the task and the environment, and how to increase the complexity to develop robots. Since these issues are too difficult to deal with as general ones, a case study on a vision-based mobile robot is given in this chapter where the environmental complexity is defined in the context of robot soccer playing and a method to control the environmental complexity is proposed in which *some periods* are found and used to decide when to increase the complexity and how much. In biology, for example, some song-birds such as chaff-inches and canaries can be said that baby birds can not sing unless they listen their parents singing during a certain period [50] called "critical period." Since the living animals are developing any time as both hardware and software systems from a viewpoint of physical robot design, this period would be more *critical* than physical robot of which hardware system cannot develop (be improved) so rapidly as the living animals. However, there seems be a kind of "critical period" for robots to develop. Here, we call this "weak critical period." In our example, "weak critical period" is found as a period when the robot cannot to cope with changing situations any more with the current state space and attempts at finding new axis in the state space to cope with the new situations.

The rest of this chapter is organized as follows: first we give an example of the complexity definition in this chapter. Next, a method for efficient learning and development coping with the increase of the task environment complexity is proposed. Then, an example task of shooting with avoiding a defender is introduced. The proposed method is applied to scheduling the speed of the defender for the efficient development of the learner that attempting at coping with new situations by adding a new axis in its state space. Finally, the preliminary experiments are shown, and other issues for the complexity control are discussed.

4.2 Definition of the Interaction Complexity

Since each animal species can be regarded to have its own kind of intelligence, difference of intelligence seems to depend on the kind of agent (capabilities in sensing, acting, and cognition), the kind of environment, and the relationship between them. If agents have the same bodies, differences or levels in intelligence can occur in the complexity of interactions with their environments. In case of multiagent environment, the complexity of interactions may change because of the presence of other agents in the environment. In the following, we present our view regarding the levels of complexity of interactions, especially from a viewpoint that takes account of the existence of other agents. To simplify the discussion, we assume that the learning agent has a kind of external sensor such as vision, sonar, and so on, which can observe the consequences of its own actions.

Self definition (boundary of the body) and static agents (environments):

The area in which an agent capable of action can directly correlate between motor commands and sensor information. By direct correlation between motor commands the agent sent and the sensor information observed during the motor command executions, the agent can discriminate the static environment from others. Theoretically, discrimination between "self body" and "static environment" is a difficult problem because the definition of "static" is relative and depends on the selection of the reference (the base coordinate system) which also depends on the context of the given task.

Nakamura and Asada [49] proposed *motion sketch* as an internal representation by optical flow. In their early stage, the action space is categorized by the direct correlation between motor commands and optical flows against the static environment. Hosoda and Asada [24] proposed an adaptive visual servoing method which performs an on-line estimation of image Jacobian by tracking a visual target and a feed forward control of the robot arm to accomplish a given task (trajectory tracking) without any a priori knowledge. This means that the parts which has a direct correlation with motor commands such as the self body or the static environment can be found in a sense of that the robot can estimate the image Jacobian on it.

Passive agents:

As a result of actions of the self or other agents, passive agents are moving or still. In [5], the ball is a passive agent. In their work of autonomous sensor space segmentation [6], the ball and the goal (a part of the static environment) are included, therefore the complexity of the environment can be regarded higher than a task of goal achieving in a static environment. Takahashi *et al.* [68] proposed a method of incremental sensor space separation by which a real robot could learn to shoot a ball into a goal. In their method, a ball is modeled as a static environment until the robot reaches it, then the ball is modeled as a part of self body because its shape and size are constant.

Other active agents:

Active agents do not have a simple and straightforward relationship with self motions. In the early stage, they are treated as noise or disturbance because of not having direct visual correlation with self motor commands. Later, they can be found from more complicated and higher correlations (coordination, competition, and others). The complexity is drastically increased.

As described in Chapter 3, we introduce the idea of the *LPM* to represent the interaction between the learner and the other objects in the environment. In the *LPM*, The complexity of the interaction with environment seems to develop the internal structure inside the physical agent, and as a result, robot may emerge a variety of behaviors. In order to realize such interactions, the robot needs the minimum mechanism to estimate the increase of the environmental complexity.

4.3 Learning from Easy Mission in Multiagent Environment

4.3.1 Basic Idea

In order for the agent to develop efficiently according to the increase of the environmental complexity, it needs a mechanism to measure the complexity based on its experience. As such a mechanism, a *Local Prediction Model* (hereafter, *LPM*) is used which estimates the relations between the learner's behaviors and the environment through interactions (observation and action) using the method of system identification. As describe in Section 3.2.2, each element estimated by each *LPM* has a contribution-value μ with respect to future prediction. That is, by approximating the relation between inputs (learner's action) and outputs (observation), each *LPM* gives the learning agent not only the successive state

of the agent but also the priority of state vector as follows:

$$\boldsymbol{x}_t^i = \begin{bmatrix} x_{t,1}^i \\ x_{t,2}^i \\ \vdots \\ x_{t,n^i}^i \end{bmatrix} \begin{matrix} \cdots & \mu_1^i \\ \cdots & \mu_2^i \\ & \vdots \\ \cdots & \mu_{n^i}^i, \end{matrix} \tag{4.1}$$

$$1 \geq \mu_1^i \geq \mu_2^i \geq \cdots \geq \mu_{n^i}^i \geq 0, \tag{4.2}$$

where μ^i $(i = 1, \cdots, n^i)$ are the singular values calculated by Eq.A.8 and these μ^i correspond to the cosines of the principal angles between the subspace $\boldsymbol{P}$ and $\boldsymbol{F}$. **Fig.4.1** shows an graphical interpretation of principal angles between two subspaces $\boldsymbol{P}$ and $\boldsymbol{F}$. Therefore, μ_i is between 0 and 1. Here, we focus on how to accelerate the reinforcement learning by appropriately increasing the environmental complexity based on the value μ_i.

One can use all the state vectors to make the agent learn, but it would take enormously long time due to the large size of the state space. Instead of using the all vectors, one can start with a small size of the state vector set first and then increase the dimension of the state space in the following stages. The action value function in the previous stage works as a priori knowledge so as to accelerate the learning. In order to transfer the knowledge smoothly, the state spaces in both the previous and current stages should be consistent with each other. Therefore, the learning agent should have a full list of the state vectors available in advance, and selects one among them at the periods when the robot no longer can cope with the changing environment with the current state vector set.

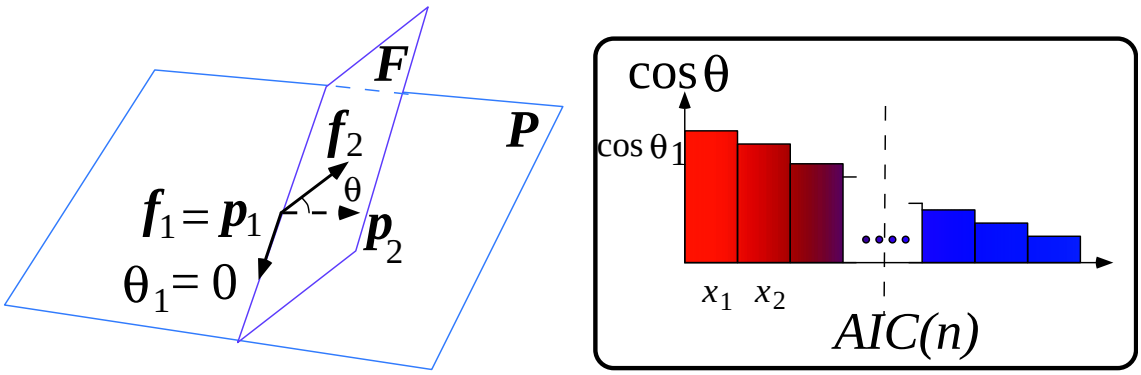


Fig.4.1 Interpretation of principal angles

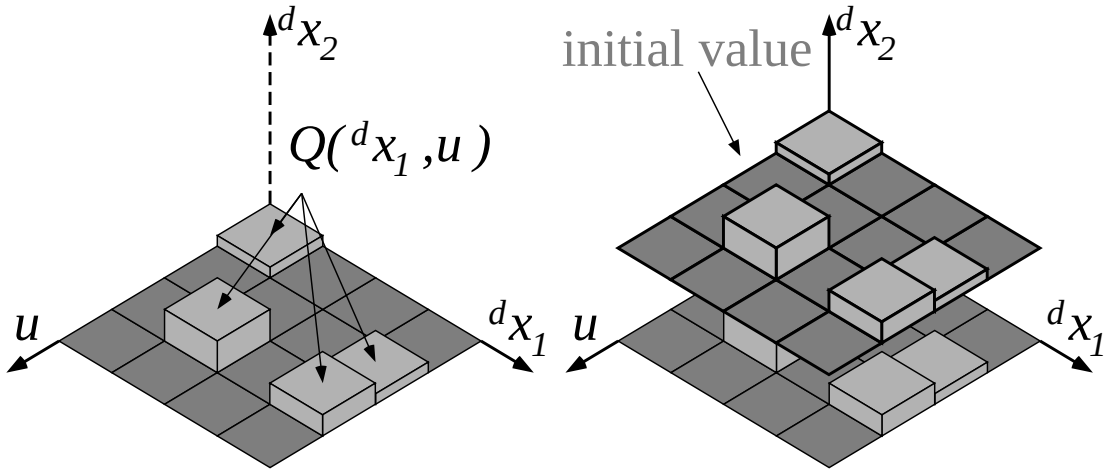


Fig.4.2 Transfer of the new action value function according to the environmental complexity

4.3.2 Algorithm for Efficient Reinforcement Learning

As described above, the learning agent obtain the rough ordering of easy situation. In general, it seems difficult to acquire a function which maps the action value function to another one based on different state representation. However, it is not efficient to learn the behavior from the beginning while the previous learning results are cast off. In order to reduce the expected learning time, there are two methods as follows.

- The learned policy is used as an initial controller in developing process. We adopt this method in Chapter 3 because the state representation is quite different.
- The learned action values are copied to the new action value function as initial knowledge in developing process.

In this chapter, we adopt the latter. Now, suppose that the learning agent adds the new axis ^dx_j into the current state representation $(^dx_1, ^dx_2, \cdots, ^dx_i)$. The size of the state space before extension is 3^i when we utilize the same segmentation by Eq.(3.9). The new action value function $Q(^dx_1, ^dx_2, \cdots, ^dx_i, ^dx_j)$ is generated by

$$Q(^dx_1, ^dx_2, \cdots, ^dx_i, ^dx_j) = \text{original value of } Q(^dx_1, ^dx_2, \cdots, ^dx_i). \quad \text{for all } ^dx_j, \tag{4.3}$$

Fig.4.2 illustrates the procedure when we transfer of the action value from $Q(^dx_1, u)$ to $Q(^dx_1, ^dx_2, u)$.

The learning agent has to observe the following two criteria: one is the convergence criterion, and the other is its own performance criterion. When the learning has converged

sufficiently, the following difference must be small,

$$\Delta Q_t(\boldsymbol{X}) = \sum_{{}^d\boldsymbol{x}\in\boldsymbol{X}} \left| \max_{\boldsymbol{u}'\in\boldsymbol{U}} Q_t({}^d\boldsymbol{x}, \boldsymbol{u}') - \max_{\boldsymbol{u}'\in\boldsymbol{U}} Q_{t-1}({}^d\boldsymbol{x}, \boldsymbol{u}') \right|. \tag{4.4}$$

Then, $\Delta Q_t(\boldsymbol{X}) <$ threshold, the agent regards the learning process as the end. Furthermore, we utilize the following equation

$$\text{success rate} = \frac{\text{number of achievement}}{\text{number of trials}}, \tag{4.5}$$

as a performance criterion which is the same criterion in Chapter 3 (Eq.(3.13)). The learning agent reports the end of the learning detected by Eq.(4.4) to the human designer in order to increase the interaction complexity while the learning agent adds the new axis from the state vector list according to the drop of the performance based on Eq.(4.5).

An algorithm to control the increase of the environmental complexity is given in **Fig.4.3**.

1. Construct the *LPMs* in the most complex task environment.
2. Set up the desired performance criterion to accomplish (for example, success rate of 80%).
3. Start with the minimum state vector set, say one or two dimensions for the lowest complexity of the task environment.
4. Keep the complexity until the agent learns the desired behavior (reach the performance criterion).
5. If the agent achieve the goal, increase the complexity slightly and return Step 4. Else, increase the dimension of the state space (find a new axis) and return Step 4. The time period of the latter case is called "weak critical period."

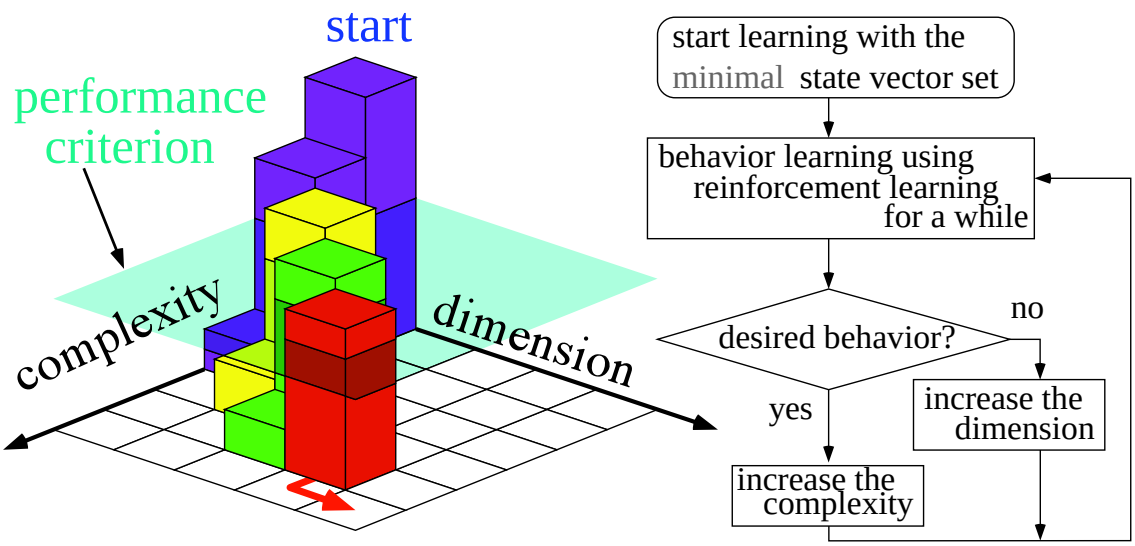


Fig.4.3 A flowchart of environmental complexity control method

4.4 Task and Assumptions

4.4.1 Environment and Robots

We apply the proposed scheduling method to a simplified soccer game shown in **Fig.4.4** (a) in which two mobile robots are included. The environment consists of a ball and a goal, and a wall is placed around the field except the goal. The sizes of the ball, the goals and the field are the same as those of the middle-size real robot league in the RoboCup Initiative [21].

Each agent has a single color TV camera and observes output vectors shown in Fig.4.4(b), which are the same observation vectors as described in Chapter 3. The dimensions of the observed vector about the ball, the goal, and the other robot are 4, 11, and 5, respectively. Our mobile robots move around using a 4-wheel steering system, which is also the same body in Chapter 3. The effects of an action against the environment can be informed to the agent only through the visual information. Furthermore, the agents receive two kinds of top down information. One is the reward that is given by the environment, and the other is the timing to alter the internal representation based on the proposed scheduling method.

As motor commands, each agent has seven actions such as go straight, turn right, turn left, stop, and go backward. Then, the input $\boldsymbol{u}$ is defined as the two dimensional vector as

$$\boldsymbol{u} = \begin{bmatrix} v \\ \phi \end{bmatrix}, \quad \begin{array}{rcl} v & \in & \{-V,\, 0,\, V\}, \\ \phi & \in & \{-\phi_{\max}, 0, \phi_{\max}\}, \end{array} \tag{4.6}$$

where v and ϕ are the velocity of motor and the angle of steering, respectively and both of which are quantized into three levels. Since two pairs of $(v, \phi) = (0, \pm\phi_{\max})$ are no physical meaning, we eliminate them. All together, the robot can choose one of seven actions. $\phi_{\max}$ is the maximum angle of steering, which is fixed through the learning processes. On the other hand, V is the actual velocity of the mobile robot, which can be controlled as the complexity parameter. That is, V is changed by

$$V = \frac{k}{K_{\max}} v_{\max}, \tag{4.7}$$

where k and $K_{\max}$ are the index and the maximum one, respectively. The agent increases k from zero or one to $K_{\max}$ in order to alter the interaction complexity gradually. We set $K_{\max} = 10$.

4.4.2 Experimental Setup

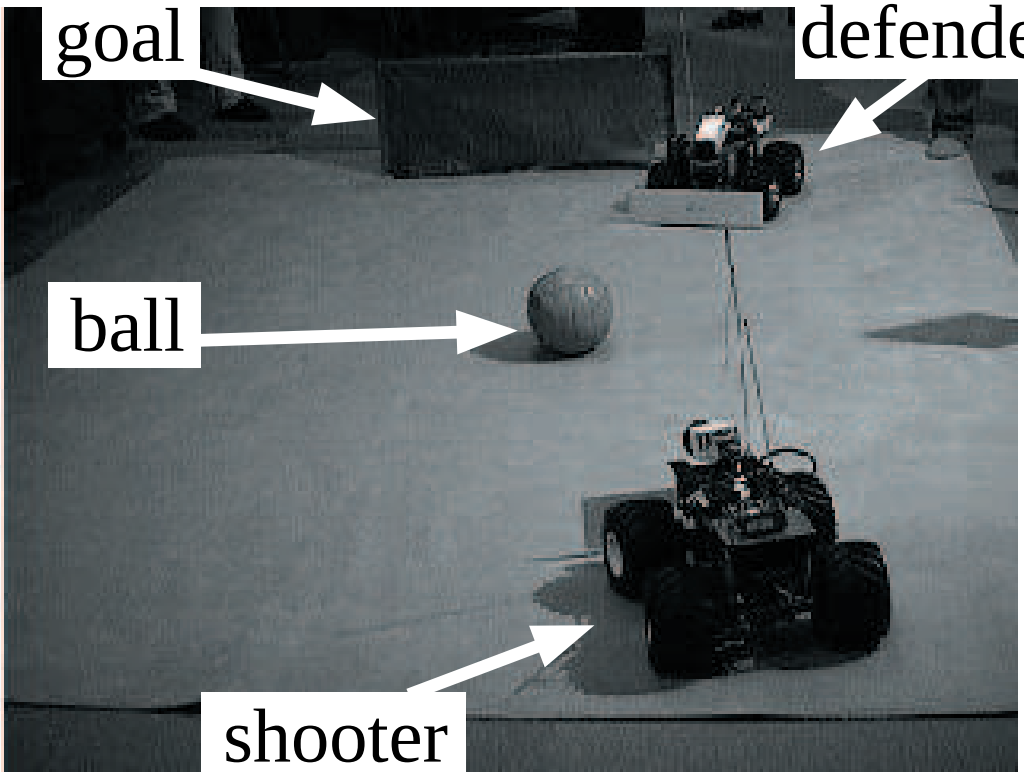
We perform two experiments to verify the proposed scheduling method to a simplified soccer task.

1. Speed control of the shooter
   There is one agent that can move actively in the environment. The task for the learner is to shoot the ball into the goal. As a control parameter of interaction, the speed of the learner is selected. In this experiment, the shooter vary the speed from slow to fast ($k = 1 \rightarrow 10$). We assign a reward value 1 when the ball is kicked into the goal or 0 otherwise.

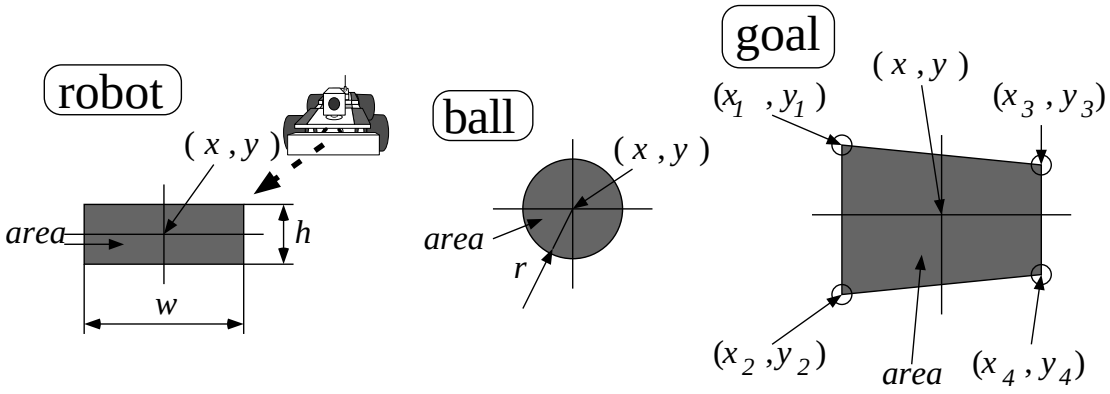
2. Speed control of the defender
   In this experiments, there are two agents in the environment. One is a shooter which learns to shoot a ball into a goal, and the other is a defender of which speed is a control parameter in the environment complexity. We assume that the defender has a basic behavior of moving to the ball, but its speed can be controlled as the complexity parameter. The defender vary the speed from slow to fast ($k = 0(\mathit{stationary}) \rightarrow 11$) while the shooter uses the maximum speed ($k = 10$). We assign a reward value 1 when the ball is kicked into the goal or 0 otherwise. On the other hand, a reward value -0.3 is given when the shooter hits the defender.

Table 4.1 shows the outline of the experiments.

To speed up the learning time, we select actions using the ϵ-greedy policy [66]. In this method, the learning agent executes random actions with a fixed probability ϵ. We set the probability of selecting a random action at $\epsilon = 10\%$.



(a) Two mobile robots (the shooter and the defender)



(b) Image features of the ball, goal, and other robot

Fig.4.4 Two real robots and the environment

Table 4.1 The outline of the experiments

item \ task	shooting	shooting and avoiding
k	$k = 1, \cdots, 10$ (10 levels)	$k = 0, \cdots, 10$ (11 levels)
selected LPM	ball	defender (other robot)
reward	1 or 0	1, -0.3 or 0
pre-specified success rate	70 %	50 % (shooting) 10 % (avoiding)

4.5 Experimental Results

4.5.1 Speed Control of the Shooter

At first, we show the experiments to control the complexity of the interactions in case of the fixed dimension of the estimated state vector about the ball. We utilize a standard tabular Q learning explained in Appendix B.1.2.

The shooter collects sequences of observation and action with the several speed ($k = 1, \cdots, 10$), and estimate the state vectors based on the *LPMs*. As a result, the shooter obtained the lists of the state vectors for the ball and the goal, respectively. The dimension of the estimated state vector of the ball and the goal is 4 and 2, respectively (See **Table 4.2**). We choose the *LPM* about the ball (other vectors are remained unchanged) to cope with the change of the complexity of the interaction.

Fixed Dimension regardless of the Change of the Shooter's Speed

Fig.4.5 shows the graphs of the success rate of shooting behavior with fixed dimension of the state vector for the ball (from 1 to 4). The speed is increased when the shooter achieves the pre-specified success rate (70%) or no improvement can be seen. The arrows indicate the time when the speed of the learner is accelerated ($k = 1, \cdots, 10$).

The best success rate of the shooting behavior is about 80 % except the case of $n = 1$.

- (a) the case of $n = 1$
  Because the size of the state space is the smallest, the expected learning time is the shortest in these experiments. At the beginning ($k \leq 2$), the shooter keeps the success rate of shooting behavior at about 50 %. However, the performance get worse drastically later.

- (b) the case of $n = 2$
  At $k = 1$ and 2, the shooter succeeds in shooting the ball into the goal with probability 80 %. We suppose that the ball can be regarded as the static environment

because the learner moves slowly. Therefore, the shooter fails according to the increase of the speed of its own. The performance get worse suddenly at $k = 3$ and 7.

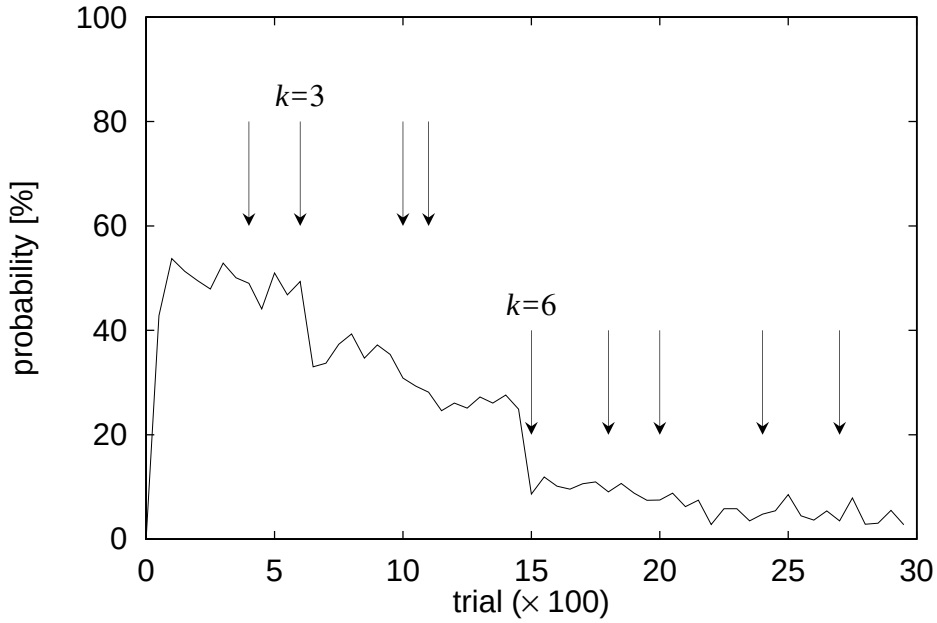
- (c) the case of $n = 3$
  From a viewpoint of the performance, we may classify the learning process into three phase as follows: (1) $k < 4$, (2) $4 \leq k \leq 6$, and (3) $k \geq 7$. This case can be regarded as the same phenomenon of the case of (b) $n = 2$.

- (d) the case of $n = 4$
  Because the size of the state space is the largest, the expected learning time is the longest. At the first step ($k = 1$), it takes about 52×10^2 trials to learn the behaviors. After that, the learner regularly increases the speed of its own every about 20×10^2 trials. In contrast to the other experiments, the success rate remains constant at $k = 3, 4$. Although the success rate decreases at $k = 7$, it is maintained in spite of the increase of the speed of the defender.

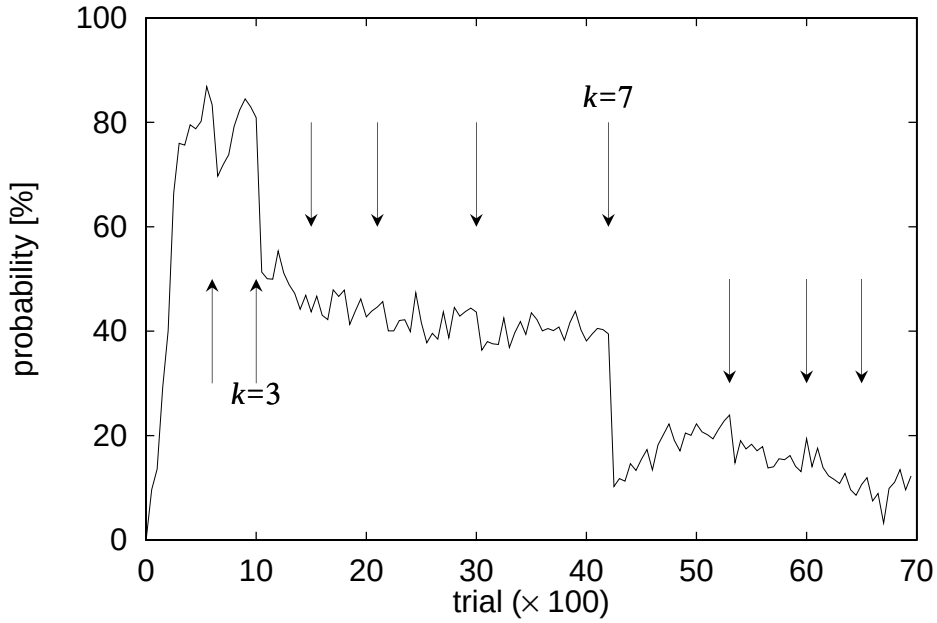
Consequently, it is sufficient to cope with this task by the case of $n = 2$, if the shooter moves slowly so that the movement of the ball may be ignored. However, it is necessary to represent the *LPM* of the ball by four dimensional state vector when the ball rolls well.

Table 4.2 The estimated dimension and the historical length in shooting task

target	estimated dimension (order)	historical length
ball	4	3
goal	2	2

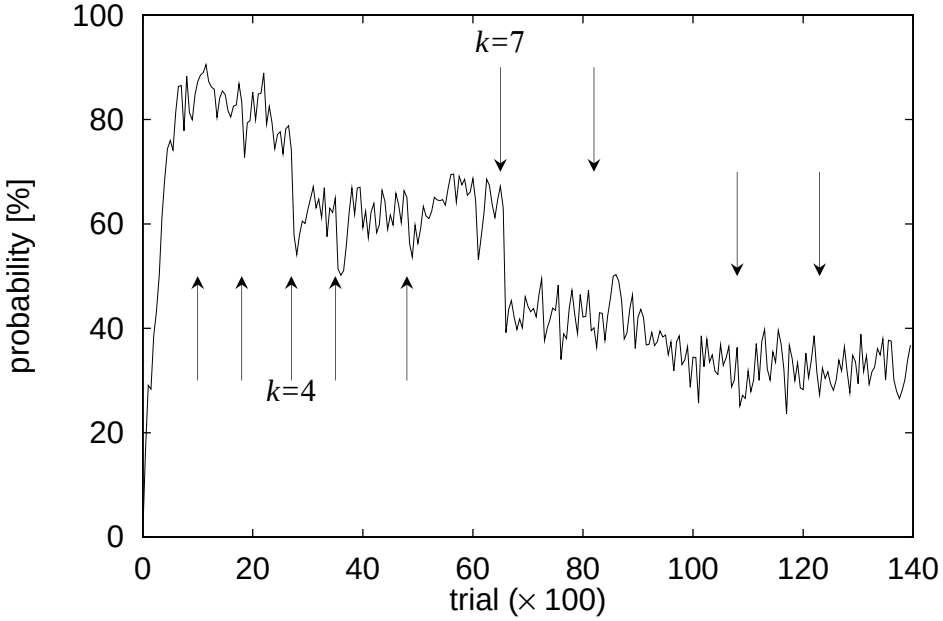


(a) $n = 1$ (minimum situation)

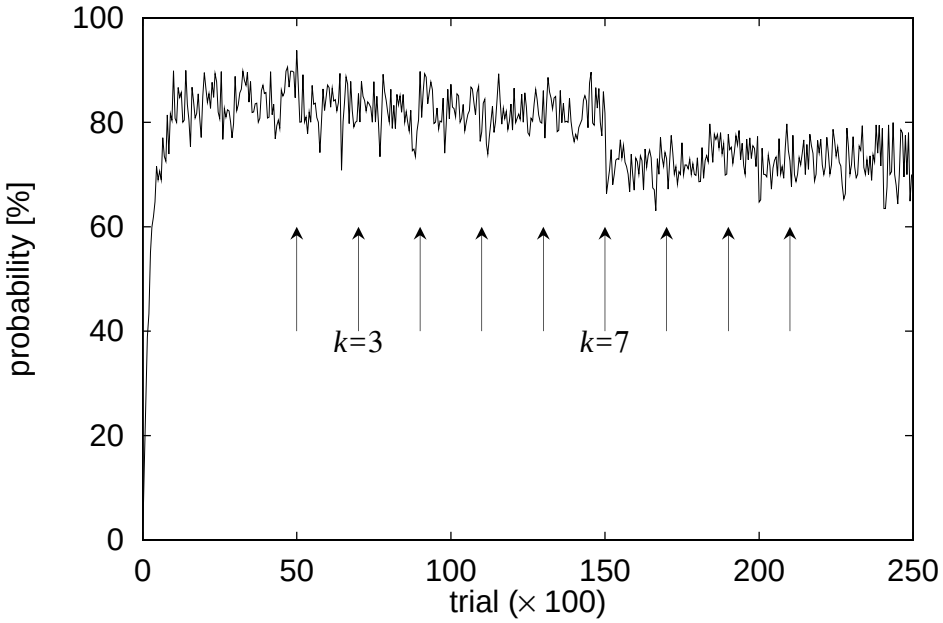


(b) $n = 2$

Fig.4.5 The success rate with the fixed dimension



(c) $n = 3$



(d) $n = 4$ (maximum dimension)

Fig.4.5 (continued) The success rate with the fixed dimension

Variable Dimension according to the Change of the Shooter's Speed

Fig.4.6 shows the result of the speed control of the shooter for the efficient learning. Short and long arrows indicate the times to increase the speed of the learner (shooter) and the dimension of the state vector, respectively. We set up 70 % performance criterion by which the timing of the speed increase of the shooter is decided.

Our proposed scheduling method can achieve the almost same performance faster than the case of learning by the maximum dimension of the state vector from the beginning. In the same way as Fig.4.5(a), the desired performance can not be accomplished by the case of $n = 1$. For this reason the shooter extends the state space from $n = 1$ to $n = 2$. After the speed of the learner is changed to $k = 3$, the success rate decreases (See Fig.4.5(b)). As a result, the shooter adds the new axis to the state space in succession ($n = 3, 4$). Compared with Fig.4.5(d), the proposed scheduling method reduce the learning time by about 60 %. It follows from this that our method is useful to acquire the purposive behaviors efficiently.

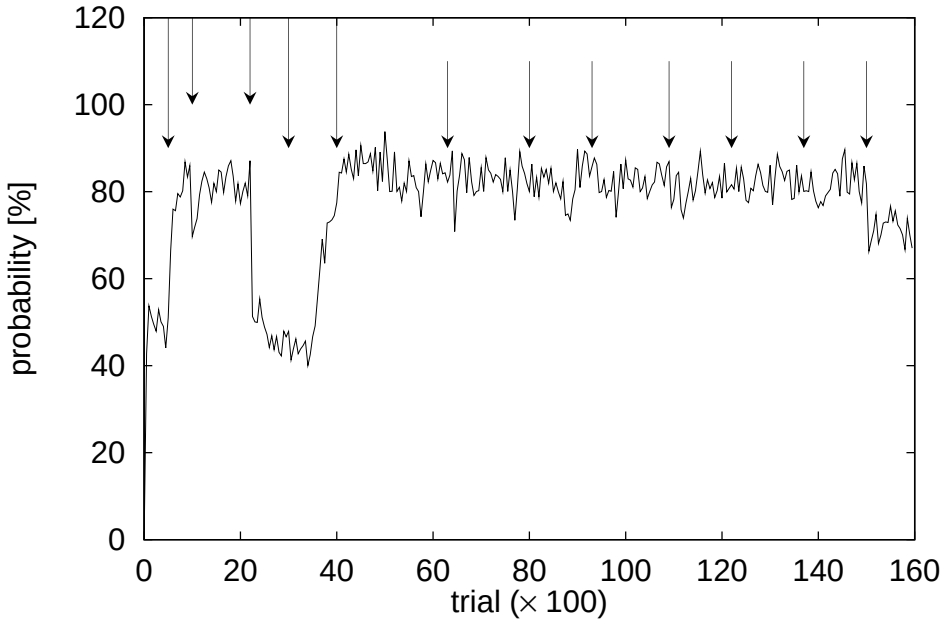


Fig.4.6 Success rate of shooting behavior with the variable dimension

4.5.2 Speed Control of the Defender

Next, we demonstrate the experiments to control the complexity of the interactions in case of the fixed dimension of the estimated state vector about the defender. As a learning method, we use modified reinforcement learning [70] described in Appendix B.2. This method can coordinate multiple behaviors (shooting behavior and avoiding one) taking account of a trade-off between the learning time and the performance.

At the beginning, the shooter collects sequences of observation and action with the maximum speed $v_{\max}$ of the defender (the speed of the shooter is fixed), and estimate the state vectors using the *LPMs*. Then, we obtain the list of the state vector for the defender and others. The dimension of the estimated state vector of the defender, the ball and the goal is 4, 4 and 2, respectively. We choose the dimension of the state vector only about the defender (other vectors are remained unchanged) which is estimated by the *LPM* to cope with the change of the complexity of the interaction.

Fixed Dimension regardless of the Change of the Defender's Speed

Fig.4.7 shows the graphs of the performance data (the probability of the success rate of shooting and making collisions) in terms of the speed of the defender with fixed dimension of the state vector for the defender (from 1 to 4). The speed is increased when the shooter achieves the pre-specified success rate (50%) or no improvement can be seen. The arrows show the time when the speed of the defender is accelerated (10 % speed increase of the maximum motion speed $v_{\max}$ from 0 (stationary)).

In spite of the number of dimensions, the best success rate of shooting is about 80 %. However it takes much time for learning agent to acquire the best performance when the dimension of the state space for the defender increases. In Fig.4.8 (d), the performance data until the speed of $0.4v_{\max}$ ($k = 4$) is shown because of the space limit. For all the cases, there is a sudden drop of the performance when the defender changes the speed to $0.2v_{\max}$ ($k = 2$).

- (a) the case of $n = 1$
  Although the expected learning time is the shortest in these experiments because of the smallest size of the state space, the performance of the learner go from bad to worse drastically according to the increase of k. At $k = 3$, the probability to make collisions is higher than the success rate of the shooting behaviors.

- (b) the case of $n = 2$
  The success rate of shooting falls gradually, and the shooter tends to make collisions with the defender. This trend is similar to the case of $n = 1$. However it takes much time to acquire the behaviors although it leads to the almost same performance.

- (c) the case of $n = 3$
  There is slight improvement of the success rate of shooting behaviors at $k = 3$, it

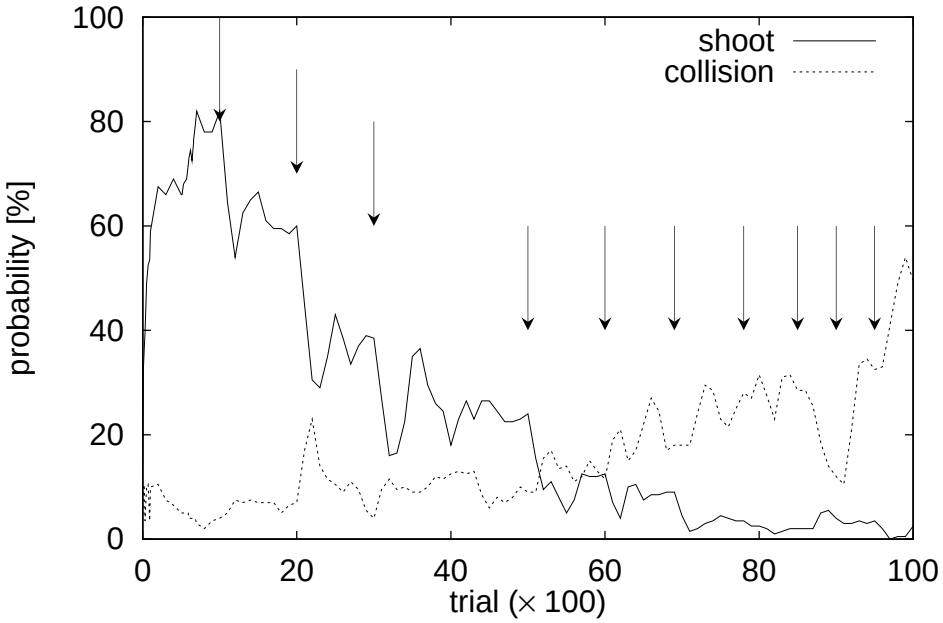
can not evade the falloff of the performance. The probability to make collisions tends to go up.

- (d) the case of $n = 4$
  Because the size of the state space is the largest, it takes the longest time to learn the behaviors. Note that the probability to make collisions remains low. As compared with the case of $n = 1, 2, 3$, the shooter keeps the probability in this experiment. The probabilities of the shooting and collisions are 38.5 % and 14.7 %, respectively.

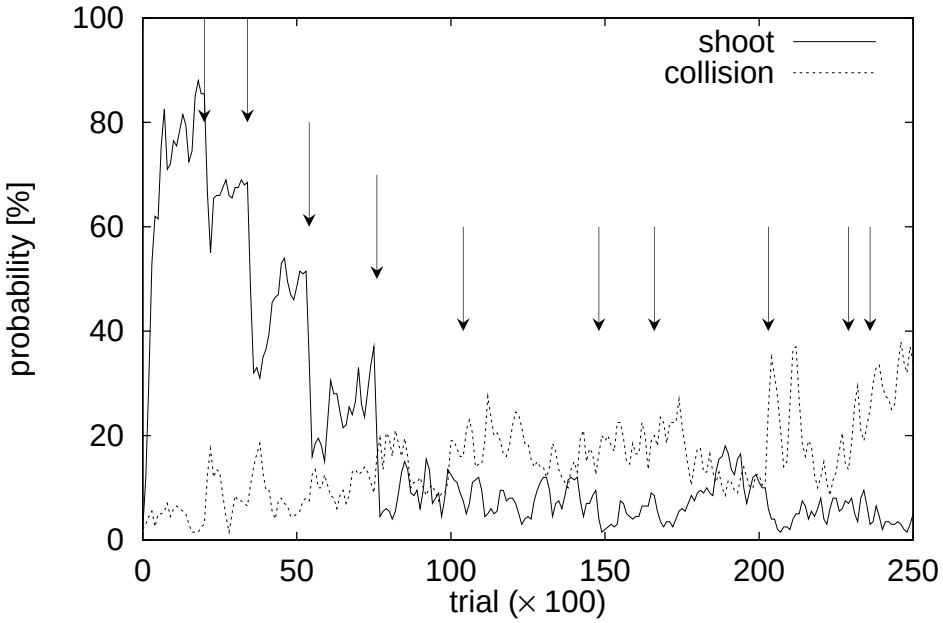
The reason why the learner fails to shoot the ball is that the action selection strategy that we utilize in this experiment. The ϵ-greedy strategy selects the random action with probability ϵ. Furthermore, the learning agent has to take account of the trade off between shooting behavior and avoiding behavior while the defender only pushes the ball. Therefore, the learning agent might not accomplish the shooting task if the defender moves quickly.

Table 4.3 The estimated dimension and the historical length in shooting and avoiding tasks

target	estimated dimension (order)	historical length
ball	4	3
goal	2	2
defender	4	4

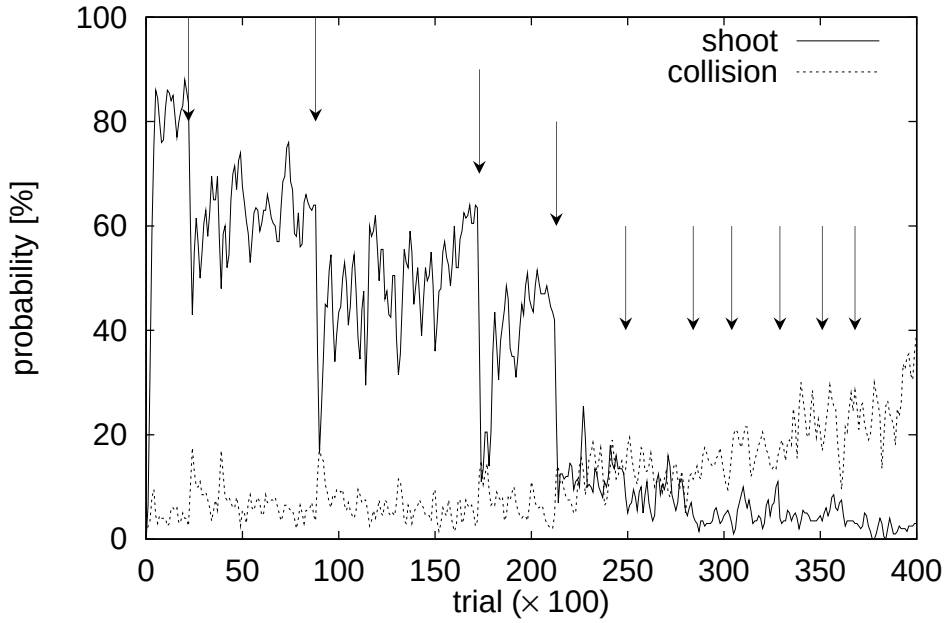


(a) $n = 1$ (minimum dimension)

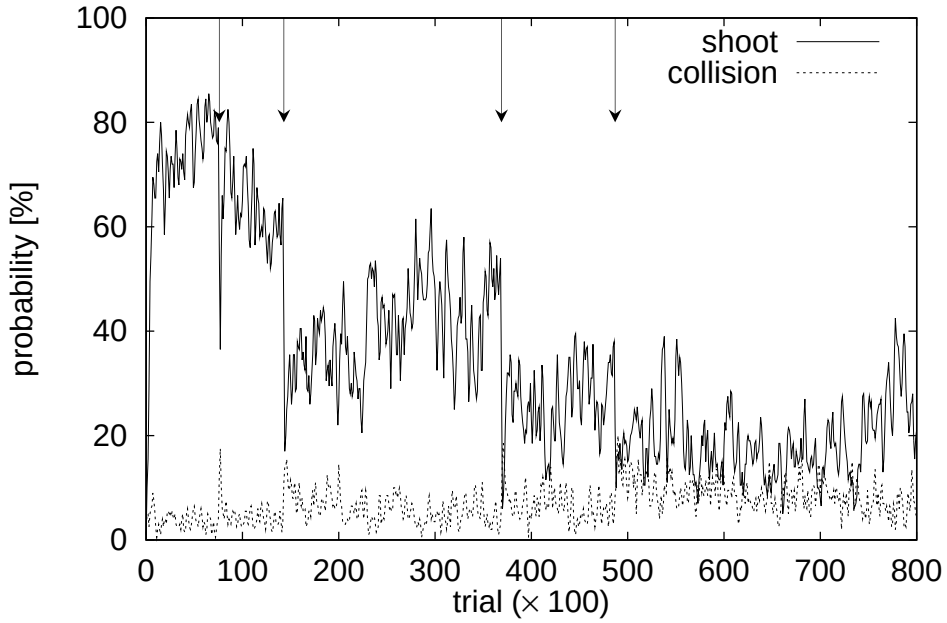


(b) $n = 2$

Fig.4.7 The success rate with the fixed dimension



(c) $n = 3$



(d) $n = 4$ (maximum dimension)

Fig.4.8 (continued) The success rate with the fixed dimension

Variable Dimension according to the Change of the Defender's Speed

Fig.4.9 shows the result of the speed control of the defender for the efficient learning. Short and long arrows indicate the times to increase the speed of the defender and the dimension of the state vector, respectively. The long arrows indicate "weak critical periods." We set up 50 % performance criterion by which the timing of the speed increase of the defender is decided.

Compared with Fig.4.7, we may conclude that the fewer dimensions of the state space contribute to the reduction of the learning time but less performance and vice versa. For example, one dimensional state vector cannot cope with $k = 2$ while two dimensional state vector can not represent the situation with $k = 3$ for the learner to learn shooting behaviors. If we start with one dimension case and step up the dimension, we also give up $k = 4$ but with four dimensions the collision rate is much less than the success rate within 150×10^2 trials.

Our proposed scheduling method can achieve the almost the same performance faster than the case of learning by the maximum dimension of the state vector from the beginning. We suppose that the reasons why our method can achieve the task faster are as follows. First, the time needed to acquire an optimal behaviors mainly depends on the size of the state space, which are determined by the dimension of the state vector estimated by the *LPM*. Our method assigns the appropriate dimension of the state vector according to the complexity while the full dimension of the state space (Fig.4.8(d)) is redundant in the early state of learning. Second, since our proposed method utilizes the action value function which is previously acquired as the initial value, it can reduce the learning time. In other words, our method consider not only the size of the state space according to the complexity but also the initial values of the action value function which is usually initialized zeros.

Finally, we show the example of an acquired behavior in **Fig.4.10**. The two lines emerged from each robot show its visual angle.

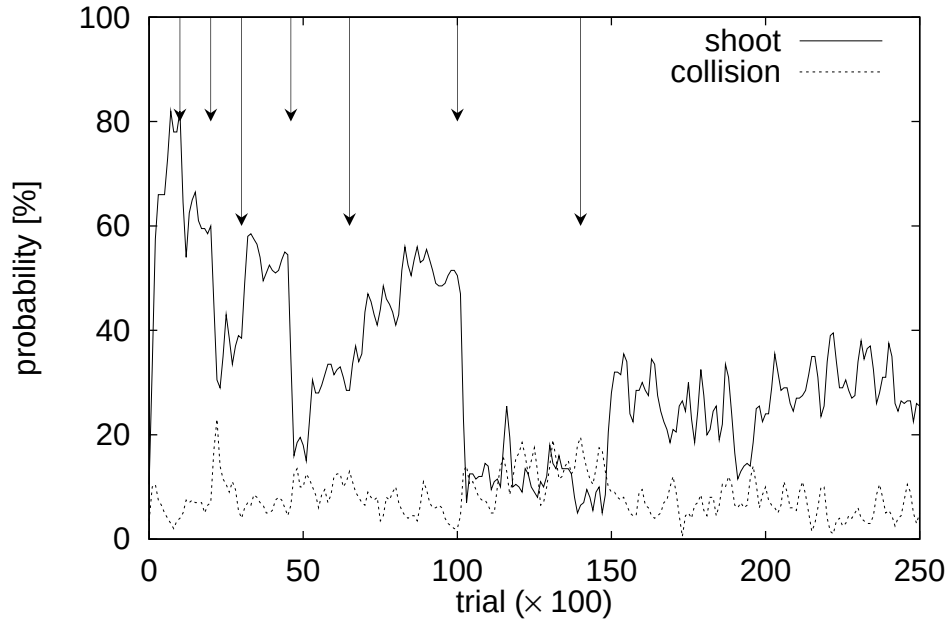


Fig.4.9 Success rate of shooting and avoiding behaviors with the variable dimension

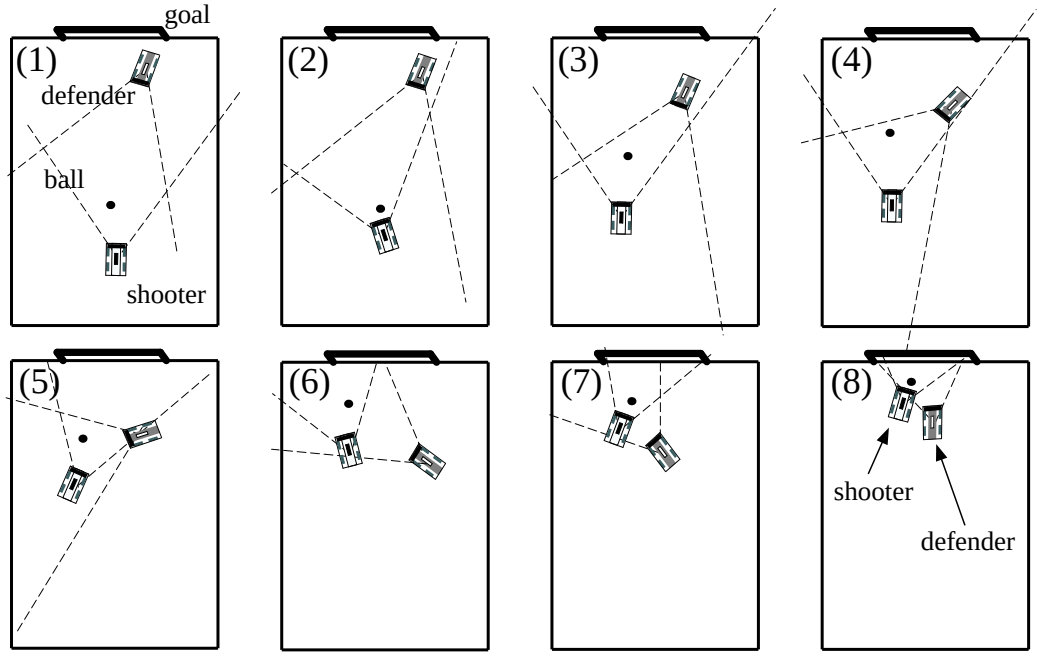


Fig.4.10 The shooter shoots the ball into the goal avoiding collisions with the defender

4.6 Discussion and Future Works

In this chapter, we show the method of controlling the environmental complexity based on the *LPM* proposed in Chapter 3. Generally, it seems difficult to know which situation is easier than others to accomplish the task in advance, unless *a priori* knowledge on ease of achieving the task is given. The proposed scheduling method substitute the state vector list (Eq.(4.1)) estimate by the *LPM* for the exact ordering of easier situation. We demonstrate two experiments along with a simplified soccer task including two mobile robots. In the experiment using two agents, although the relation between them is competitive, the defender should behave so that the learner can efficiently improve its behavior. In other word, the defender is a kind of teachers for the shooter. This can be considered as a problem of learning from other competitive agent. The large difference from the existing schemes such as learning by Watching [35] and social learning [42] is that the other agents is not always friendly nor suggestive to help the agent learn. Rather, the other agents are involved in the task which the agent has to accomplish. The issue of the change of difficulties according to the behavior of others is not irrelevant to the issue of mutual skill development which we will explain in Chapter 6.

Strictly speaking, the order acquired by the *LPM* is the contribution-value with respect to future prediction. Therefore, this ordering does not guarantee the optimal ordering with respect to the easier situation. Nevertheless, the prediction learning has an advantage of the learning speed as compared with behavior learning. There are two main issues to be considered.

Selection of the Control Parameters

First, the number of control parameters is one in each experiment, but generally multiple, each of which is related to each other. Even in the example task, the speed of the learner, the dimensions of the state space, the resolution of the each dimension (fixed (3 partitions) in the experiments) and the initial configurations of the ball, the goal, the learner, and the defender should be considered together with the speed of the defender. In such a case, since designer cannot completely understand the relationships among them, it seems difficult to decide how to control the complexity completely.

Online Estimation

Then, the second issue is revealed. To cope with unknown complexity, the learning agent should estimate the state vectors anytime when the task performance becomes worse. However, this causes inconsistency in state vector sets between the current and next learning stages. Therefore, the knowledge transfer is limited to the initial controller (action selection) and the agent needs much more memory and the learning time. Since this is against resource bounded condition, we should develop a new method which can take account of this trade-off.

Chapter 5

Vector-Valued Reward Function for Cooperative Behavior Acquisition

5.1 Introduction

Design of the reward function is one of the most difficult aspects in applying reinforcement learning systems, especially in a multiagent environment. Since our system does not facilitate the explicit communication among agents, the total performance may get worse if each agent attempt to maximizes its own utility. One of the reasons is that the relation among the reward functions is compromise or conflict. Zlotkin and Rosenschein [78] classified the interaction of agents from a viewpoint of the reward function into three categories: (1) cooperative, (2) compromise, and (3) conflict situation. In Chapter 3, each agent does not have explicit procedures how to cooperate, but only has a look up table to behave that has been obtained through the learning process. But the resultant behaviors seem cooperative ones. This is owing to the careful design of the reward function.

If we can prepare the reward function that satisfy the (1) cooperative one in advance, it is easy for the learning agents to cooperate because the purpose is consistent between the individual and social (global). However, it seems very difficult to generate the appropriate reward by itself since there are no criteria without explicit communication[1], rewards and/or penalties. Alternative is to consider not only the individual reward functions but also whole ones including cooperative factors. The problem is how the agents should cope with the multiple rewards because there is usually tradeoff between individual and team utilities.

Mataric [42, 43] designed the several reward function from a viewpoint of animal interactions and implemented the behavior learning based on the weighted sum of multiple rewards. In her case, fundamental behaviors have been embedded as a form of subsumption architecture, which makes learning itself simple. However, the methods of the weighted sum of reward functions are faced with the essential problem of weighting which

[1] If the agents can communicate each other, compromise situation may be realized by negotiation [78].

we will explain later.

In this chapter, we propose a vector-valued reward function to cope with multiple tasks. We implement an architecture of an actor-critic type as a learning mechanism. The critic is a state value function. After each action selection, the critic evaluates the new state to determine whether it has become better or worse than expected.

The rest of this chapter is organized as follows. First, we extend the scalar reward function to a vector-valued one. Then, we describe our algorithm for behavior learning based on the extended reward function by actor-critic method. We apply the proposed method to a series of simplified soccer tasks. Finally, we show the results of computer simulation, and a discussion is given.

5.2 Vector-valued Reward Function

5.2.1 Temporal Difference

Before explanation of the proposed method, we show a learning algorithm of Temporal Difference (hereafter TD) method and the state value function briefly for the reader's understanding. We consider the state-outcome sequences of the form $\boldsymbol{x}_t$, $\boldsymbol{x}_{t+1}$, $\cdots$, $\boldsymbol{x}_{t+n}$, r, where each $\boldsymbol{x}_t$ is a state vector at time t in the sequence, and r is the outcome of the sequence. Many such sequences will be normally obtained (See **Fig.5.1**).

The given task is to predict the future reward to receive at each state. TD(λ)[2] [66] maximizes (or minimizes) scalar cumulative discounted sum

$$v_t(\boldsymbol{x}_t) = \sum_{n=0}^{\infty} \gamma^n r_{t+n}, \tag{5.1}$$

where γ is a discount factor between 0 and 1.

Suppose that the environment makes a state transition from the current state $\boldsymbol{x}_t$ to the next state $\boldsymbol{x}_{t+1}$ and generates the reward r_t. If the predictions are accurate (v is optimal), we can obtain

$$v(\boldsymbol{x}_t) = r_t + \gamma v(\boldsymbol{x}_{t+1}),$$

from Eq.(5.1). The mismatch called "TD error" is the difference between two sides of this equation. Therefore, v_t is updated by

$$v_{t+1}(\boldsymbol{x}_t) = v_t(\boldsymbol{x}_t) + \alpha[r_t + \gamma v_t(\boldsymbol{x}_{t+1}) - v_t(\boldsymbol{x}_t)], \tag{5.2}$$

where α ($0 \leq \alpha \leq 1$) is a learning parameter. Q learning [73] incorporates the "action" into the idea of TD learning.

[2] λ is an exponential weighting parameter with recency [66]. For the sake of reader's understanding, we cope with the parameter $\lambda = 0$.

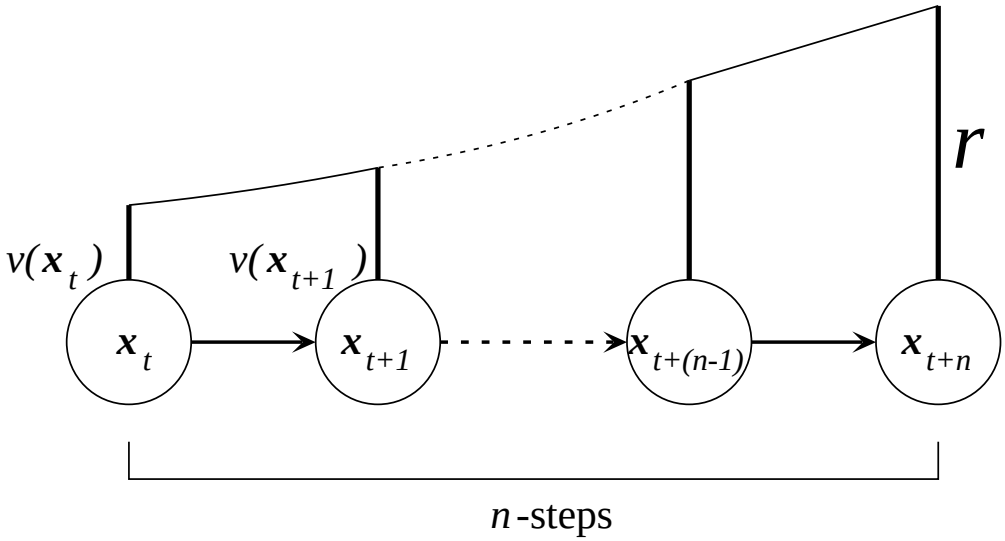


Fig.5.1 TD error ($\lambda = 0$)

5.2.2 Evaluation Function for Vector-Valued Rewards

As described above, in case of realizing cooperative behaviors in a multiagent environment, the learning agent has to consider the tradeoff between the individual and the team purposes as possible. Suppose that the learner has N tasks to accomplish. The multiple rewards from the environment are given to the agent as follows:

$$\boldsymbol{r} = \begin{bmatrix} r^1 \\ r^2 \\ \vdots \\ r^N \end{bmatrix} \begin{matrix} \cdots \\ \cdots \\ \\ \cdots \end{matrix} \begin{matrix} \text{the reward for} \\ \\ \\ \end{matrix} \begin{matrix} \text{the first task,} \\ \text{the second task,} \\ \vdots \\ \text{the } N\text{-th task.} \end{matrix} \tag{5.3}$$

In order to cope with multiple rewards, one of the simplest implementation of them is a weighted combination of the rewards like

$$r = \sum_{i=1}^{N} w^i r^i, \tag{5.4}$$

where w^i is a weight for the reward r^i, e.g, the objective is to maximize (or minimize) the weighted sum

$$\sum_{n=0}^{\infty} \gamma^n (w^1 r^1 + \cdots + w^N r^N).$$

This reduces the problem to the case of scalar-valued reinforcement values. However, this combination has the following deficits.

- The discounted factor γ is common to the all tasks.

- The estimated value function is unstable when we give both the positive and negative rewards.

Since γ controls to what degree rewards in the distant future affect the total value of a policy, we want to set the appropriate value for the corresponding tasks. A typical example is "collision avoidance" which has different property (negative) from that of goal directed behaviors (positive). That is, any action can be allowed to be taken unless it causes collisions with other objects. In order to learn such a behavior, γ should be much smaller so that the utility for the distant future cannot be affected.

Considering the above mentioned issue, we extend Eq.(5.1) to the vector-valued evaluation function. The discounted sum of the vector-valued reward can be expressed by

$$\boldsymbol{v}_t(\boldsymbol{x}_t) = \sum_{n=0}^{\infty} \boldsymbol{\Gamma}^n \boldsymbol{r}_{t+n}, \tag{5.5}$$

where $\boldsymbol{\Gamma}$ is $N \times N$ matrix. We call $\boldsymbol{\Gamma}$ discounted matrix. If the eigen value of the matrix $\boldsymbol{\Gamma}$ exists within the unit circle, the value expressed by Eq.(5.5) converges. Taking into account the original meaning of the discounted factor, $\boldsymbol{\Gamma}$ should be a diagonal matrix

$$\boldsymbol{\Gamma} = \begin{bmatrix} \gamma_1 & & & 0 \\ & \gamma_2 & & \\ & & \ddots & \\ 0 & & & \gamma_N \end{bmatrix}. \tag{5.6}$$

A vector-valued TD error can be calculated by

$$\boldsymbol{\delta}_t = \boldsymbol{r}_t + \boldsymbol{\Gamma}\boldsymbol{v}_t(\boldsymbol{x}_{t+1}) - \boldsymbol{v}_t(\boldsymbol{x}_t), \tag{5.7}$$

As a result, the state value function in tabular representation is updated by

$$\boldsymbol{v}_{t+1}(\boldsymbol{x}) = \boldsymbol{v}_t(\boldsymbol{x}) + \alpha\boldsymbol{\delta}_t, \tag{5.8}$$

where α is a learning parameter between 0 and 1.

1. Initialize the value function $\boldsymbol{v}(\boldsymbol{x})$ to 0s for all states.
2. Perceive the current state $\boldsymbol{x}$.
3. Execute an action. As a result, the environment makes a state transition to the next state $\boldsymbol{x}'$ and generates the reward $\boldsymbol{r}$.
4. Update the state value function from $\boldsymbol{x}$, $\boldsymbol{x}'$, and $\boldsymbol{r}$ by Eq.(5.8).
5. Return to 2.

5.3 Behavior Learning based on the Vector-Valued Reward Function

5.3.1 Estimation of the Policy

Actor-critic methods are TD methods that have a separate memory structure explicitly, and represent the policy independent of the value function. The policy structure is known as the actor, because it is used to select actions, and the estimated value function is known as the critic, because it criticizes the actions made by the actor.

Although convergence proofs for the actor-critic algorithms are less than value iteration based algorithms such as Q learning, the actor-critic algorithms have the following practical advantages [30].

- It is possible to implement multidimensional continuous action, that is often mixed with discrete action.
- It is easy to incorporate an expert's knowledge into the learning system by applying conventional supervised learning techniques to the actor [13].

The TD error can be used to evaluate the action $\boldsymbol{u}$ taken in the state $\boldsymbol{x}$. If the TD error is positive, it suggests that the tendency to select the action $\boldsymbol{u}$ should be strengthened for the future. On the other hand, if the TD error is negative, it suggests the tendency should be weakened. Let $\boldsymbol{p}_t(\boldsymbol{x}, \boldsymbol{u})$ be the vector at time t for the modifiable policy. Then the strengthening or weakening described above can be implemented by increasing or decreasing the value $\boldsymbol{p}_t(\boldsymbol{x}, \boldsymbol{u})$ by

$$\boldsymbol{p}_{t+1}(\boldsymbol{x}, \boldsymbol{u}) = \boldsymbol{p}_t(\boldsymbol{x}, \boldsymbol{u}) + \boldsymbol{B}\boldsymbol{\delta}_t, \tag{5.9}$$

where $\boldsymbol{B}$ is a positive step-size parameter. To simplify the learning rule, we set $\boldsymbol{B} = \beta \boldsymbol{I}$, where $\boldsymbol{I} \in \Re^{N \times N}$ is the identity matrix.

5.3.2 Pareto based Action Selection

For a pair of ($\boldsymbol{x}$, $\boldsymbol{u}$), the learning agent has a following objective function [3],

$$\max_{\boldsymbol{u} \in \boldsymbol{U}} [p_1(\boldsymbol{x}, \boldsymbol{u}), \cdots, p_N(\boldsymbol{x}, \boldsymbol{u})]. \tag{5.10}$$

The natural choice is *Pareto optimality*, where an action $\boldsymbol{u}$ is a *Pareto optimal* action in the state $\boldsymbol{x}$ if there are no no action $\boldsymbol{u}'$ which satisfies $p_i(\boldsymbol{x}, \boldsymbol{u}') \geq p_i(\boldsymbol{x}, \boldsymbol{u})$ $(i = 1, \cdots, N)$.

The learning agent has to select several actions to explore the unknown situations. Then, we use ϵ-greedy strategy [66], meaning that most of time the agent chooses an optimal action, but with probability ϵ it instead selects an action at random. We summarize the action strategy as follows.

[3] For the reader's understanding, we discuss the case that all the value function has to be maximized. Actually, the some value function has to be minimized.

1. Execute the random action with probability ϵ, or goto step 2.
2. Initialize $rank(\boldsymbol{u}) = 0$ for all $\boldsymbol{u} \in \boldsymbol{U}$. For $i = 1, \cdots N$,
   (a) Calculate the optimal action corresponding to each p_i ($\boldsymbol{p}^T = [p_1, \cdots, p_N]$),
   $$\boldsymbol{u}_i^* = \arg\max_{\boldsymbol{u} \in \boldsymbol{U}} p_i(\boldsymbol{x}, \boldsymbol{u}). \tag{5.11}$$
   (b) Increment $rank(\boldsymbol{u}_i^*) = rank(\boldsymbol{u}_i^*) + 1$.
3. Select action,
   $$\boldsymbol{u}^* = \arg\max_{\boldsymbol{u} \in \boldsymbol{U}} rank(\boldsymbol{u}). \tag{5.12}$$

After all, the all actions are ordered with respect to the *Pareto optimality*, the learning agent select the action which satisfy the *optimal* action as possible.

Fig.5.2 shows the integration of the *LPMs* proposed in Chapter 3 and the actor-critic learning architecture. At first, the agent perceive the situation of the environment by its own sensors. Based on the sequences of observation and action, the agent constructs the *LPMs* for all objects, independently, and estimates the order and the state vector. After the sets of the state vectors are obtained, they are sent to the module of the *value function* and the *policy*. The value function evaluates the current policy from the estimated state vectors and the rewards from the environment. On the other hand, the policy selects the action based on the TD error from the value function. After the agent executes the selected action, the environment makes a state transition to the next situation. The agent acquires the purposive behaviors through this interaction with the environment.

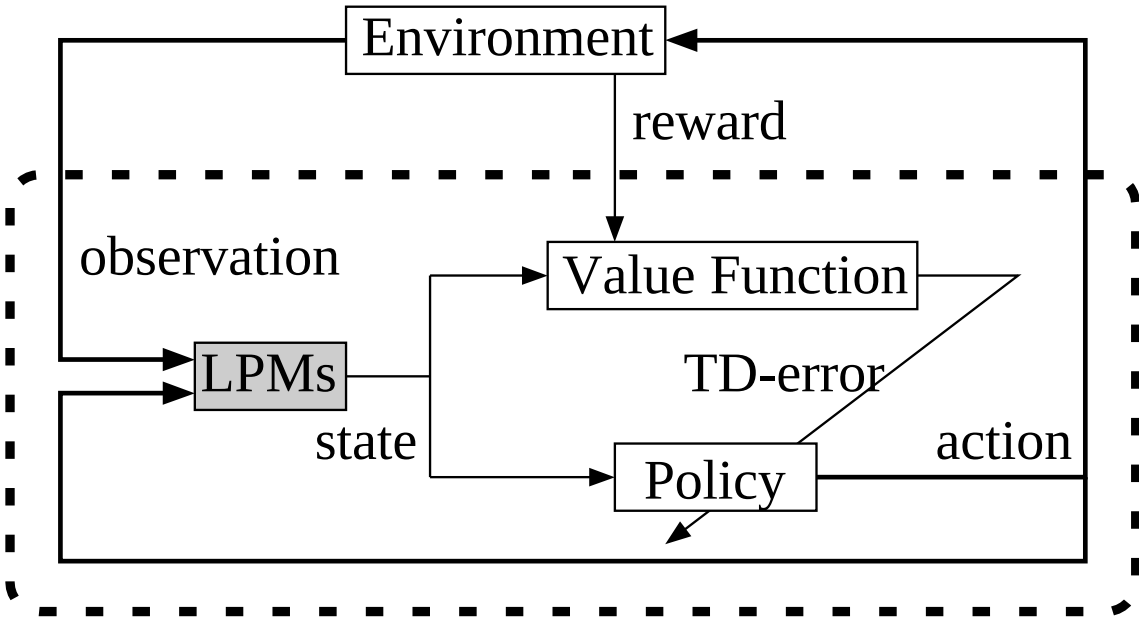


Fig.5.2 The integration of the LPMs and the actor-critic architecture

5.4 Task and Assumptions

5.4.1 Environment and Robots

We apply the proposed method to a simplified soccer game in the environment including three learning mobile robots. Because robotic soccer has been increasingly attracting as a benchmark for distributed artificial intelligence and robotics, and so on, many researchers around the world challenge this task [21]. The environment consists of a ball and a goal, and a wall is placed around the field except the goal. The sizes of the ball, the goals and the field are the same as those of the middle-size real robot league in the RoboCup Initiative [21].

Each robot does not know the locations, the sizes and the weights of the ball and the other agent, any camera parameters such as focal length and tilt angle, or kinematics/dynamics of itself. Each agent has a single color TV camera and observes output vectors which are the same observation vectors as described in Fig.3.6. That is, we prepare the same image feature vectors used in Chapter 3. The dimensions of the observed vectors about the ball, the goal, and the other robot are 4, 11, and 5, respectively.

The robot moves based on a Power Wheeled Steering (PWS) system which is different from 4WS used in Chapter 3 and 4. As motor commands, each mobile robot has a two

degree of freedoms. The input $\boldsymbol{u}$ is defined as a 2 dimensional vector:

$$\boldsymbol{u} = \begin{bmatrix} v \\ \omega \end{bmatrix} \quad v, \omega \in \{-1, 0, 1\}, \tag{5.13}$$

where v and ω are the velocities of translation and rotation of the robot, respectively. In this experiment, v and ω are quantized into three levels, which are uniformly distributed. Totally, the agent can select one from nine actions at each state.

5.4.2 Experimental Setup

We perform four experiments to verify the proposed learning algorithm using a simplified soccer game shown in **Fig.5.3**.

(i) Shooting a ball into the goal (Fig.5.3 (a))
In the environment, there is one learning agent **r0** which learns to shoot the ball into the goal. The task is not only to shoot the ball into the goal but also to kick the ball. The second task, that is, to kick the ball is not originally the primal task for the robot. The purpose of this experiment is to compare the differences between the proposed method and the method of weighted sum when we introduce the sub-reward in order to accelerate the learning processes.

(ii) Shooting a ball into the goal without collisions with other robot (Fig.5.3(b))
In this experiment, there are two agents in the environment. One is a shooter **r0** which learns to shoot a ball into a goal avoiding collisions with the other robot as possible, and the other **r2** is a defender to disturb the learning agent. That is, this is the same situation described in Chapter 4. The defender moves based on the fixed policy.

In these two experiments, only one agent learns the behavior. We discuss the differences between the proposed method and the previous one (weighted sum by Eq.(5.4)) because it is easy to focus our attention on the differences between the proposed method and the previous one. Then, we perform the following third and fourth tasks. In order to make the learning processes stable, we utilize the learning schedule which we have proposed in Section 3.4.

(iii) Shooting and passing a ball without collisions with each other (Fig.5.3(c))
There are two learning agents (**r0** and **r1**) in the environment. The setting of this experiment is as the same as that of one described in Chapter 3 except that the role of each agent has not been pre-specified. That is to say, the reward function is common between two learning agents.

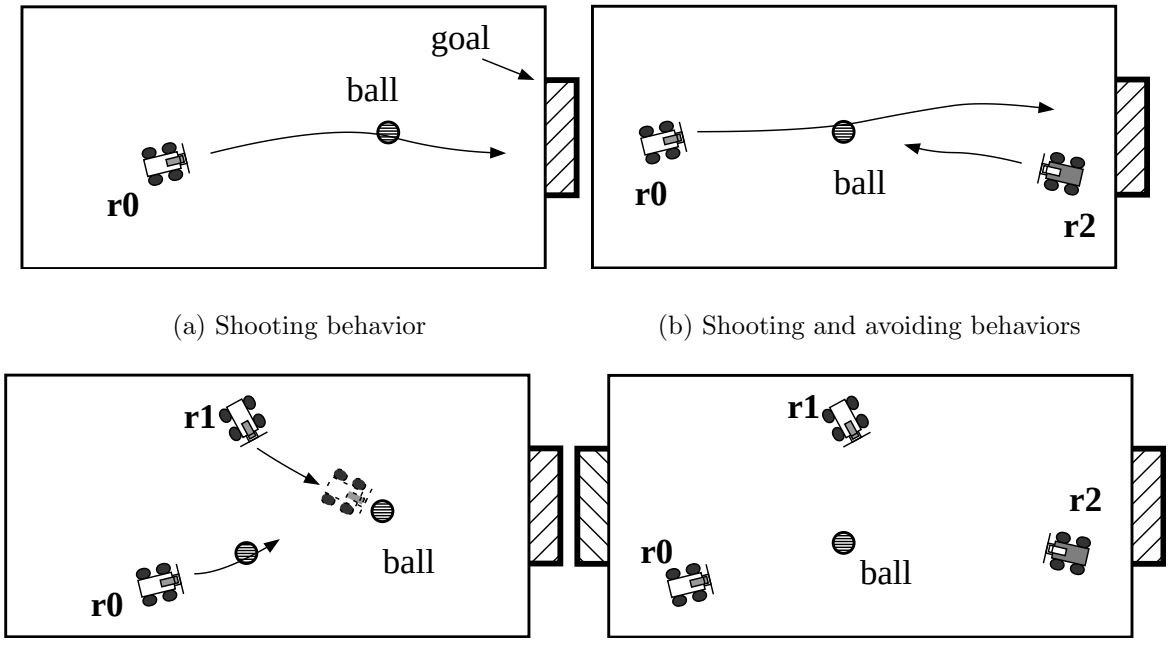
(iv) Simplified game among three robots (Fig.5.3(d))
At last, we perform a simplified soccer game by three learning agents. **r0** and **r1** are teammates while **r2** is a competitor against them. The difference from the

experiments described above is involvement of competition. This experiment will be examined in the next Chapter 6.

One trial is terminated if one of the robots shoots a ball into the goal or the pre-specified time interval expires. The trial still continues even if a pass behavior is achieved, the robots make collisions, or the robot pushes the ball. **Table 5.1** shows the summary of these experiments.

Table 5.1 The summary of a series of the experiments

task no.	(i)	(ii)	(iii)	(iv)
the number of agents	1	2	2	3
the number of tasks	2	3	4	6



(a) Shooting behavior

(b) Shooting and avoiding behaviors

(c) Shooting, passing and avoiding behaviors

(d) A simplified game

Fig.5.3 Three robots and the environment

5.5 Experimental Results

5.5.1 Shooting a Ball into the Goal

We give the learning agent two tasks in this experiment. One is to shoot a ball into the goal, and the other is to kick the ball. Then, the rewards are two dimensional vector as

$$\boldsymbol{r} = \begin{bmatrix} r^s \\ \\ r^k \end{bmatrix} \begin{array}{ll} \cdots & \text{the positive reward for success of shooting} \\ & \text{the ball into the goal,} \\ \cdots & \text{the positive reward for kicking the ball.} \end{array} \tag{5.14}$$

We set a reward $r^s = 1.0$ for the success of shooting the ball into the goal while a reward $r^k = 1.0$ is given when the robot kick the ball. The discounted matrix $\boldsymbol{\Gamma}$ is selected,

$$\boldsymbol{\Gamma} = \mathrm{diag} \begin{bmatrix} 0.9 & 0.6 \end{bmatrix}.$$

In addition, we compare the proposed method with the following linear weighting reward function,

$$r = r^s + w^k r^k,$$

where w^k is a weight. We prepare three weights as (a) $w^k = 0.0$, (b) $w^k = 1.0$, and (c) $w^k = 0.2$. The discounted factor γ in order to estimate the scalar-valued reward function is set to 0.9.

Fig.5.4 indicates the differences among the proposed method and the other method. It follows from this that

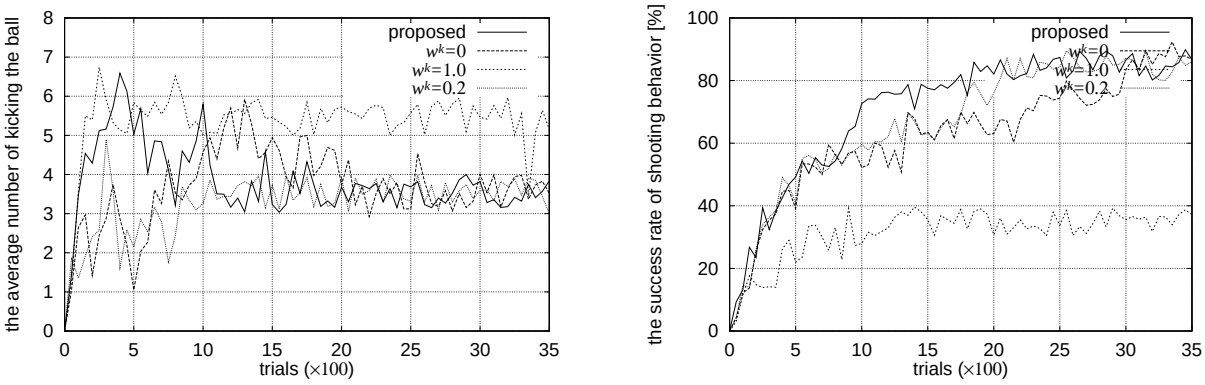
- (a) $w^k = 0.0$: This is the same reward function that we utilized for acquisition of shooting behaviors in Chapters 3 and 4. This can be the ordinary reward function which is widely used to achieve the monolithic task [5].

- (b) $w^k = 1.0$: The learning agent based on this method achieved the lowest success rate of shooting the ball in to the goal. The reason is that the weight is not appropriate to coordinate these behaviors (shooting and kicking). The same reinforcement signal is given to the agent, which cannot distinguish the event with respect to the value function. As a result, this agent tends only to kick the ball from a viewpoint of us.

- (c) $w^k = 0.2$: Because the value of w^k is so appropriate that this agent realize the higher performance than the case of (b). In this experiment, the value 0.2 is chosen by trial and error, however, it seems difficult to give the good reward value in more complicated tasks.

- The agent based on the proposed method acquire the almost same behavior a little bit faster than the case of original learning method.

Finally, we show a sequence of shooting behavior by the proposed method in **Fig.5.5**.



(a) the number of kicking the ball

(b) the success rate of shooting the ball into the goal

Fig.5.4 Differences of the acquired performance in a case of shooting behavior

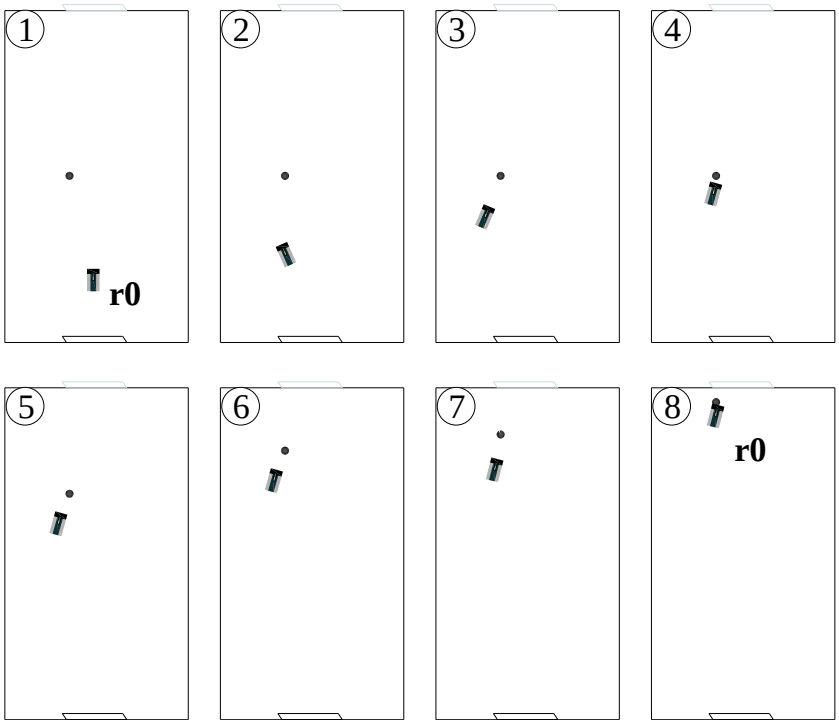


Fig.5.5 Acquired shooting behavior in computer simulation

5.5.2 Shooting a Ball into the Goal without Collisions

Next, we consider the coordination of shooting and avoiding behavior. The defender **r2** has a fixed policy of chasing the ball, and its motion speed is a 30 % of the maximum speed taking account of the result[4] described in Section 4.5.2. In this task, one agent learns the behavior while the other moves around based on the fixed policy. We add a new reward to Eq.(5.14),

$$\boldsymbol{r} = \begin{bmatrix} r^c \\ r^s \\ r^k \end{bmatrix} \begin{array}{ll} \cdots & \text{the negative reward for collisions,} \\ \cdots & \text{the positive reward for success of shooting,} \\ \cdots & \text{the positive reward for kicking the ball,} \end{array} \tag{5.15}$$

for the learner. We set the same rewards $r^s = 1.0$ and $r^k = 1.0$ for the success of shooting the ball into the goal and kicking the ball. On the other hand, we assign a negative reward $r^c = -1.0$ when the learner make collisions with the other robot. The discounted matrix $\boldsymbol{\Gamma}$ is selected,

$$\boldsymbol{\Gamma} = \mathrm{diag}\begin{bmatrix} 0.3 & 0.9 & 0.6 \end{bmatrix}.$$

In the same way of Section 5.5.1, we prepare the linear weighting reward function

$$r = r^c + r^s + w^k r^k, \tag{5.16}$$

to compare the proposed method. We select two values: (a) $w^k = 0.0$, (b) $w^k = 1.0$. The discounted factor γ in order to estimate the scalar-valued reward function is set to 0.9.

We show the experimental results about the frequencies of reception of the external rewards in **Fig.5.7**. It follows from Fig.5.7(a), the agent with $w^k = 1.0$ tends to push the ball more frequently than other agents. Then, the success rate of shooting behavior is worse than the case of the proposed method and no weight.

The reason is that the reward function expressed by Eq.(5.15) is a mixture of the positive and negative value. Then, the value function based on Eq.(5.16) may not estimate the value (utility) appropriately[5]. Also, the problem of the estimator based on Eq.(5.16) is to use the same discounted factor among the given tasks. Suppose that the learning agent make a collision with other agent. The learning agent receive r^c more frequently than r^s, which lets the agent to acquire the avoiding behavior at the beginning. As a result, the agent can not seek the feasible solutions for acquisition of shooting behavior.

Although the proposed method consume the much memory, the value function can estimate the utility much easier because it can assign different value function corresponding to the given tasks. We have already proposed the method which can coordinate multiple behaviors [70] described in Appendix B.2.2. We utilized this learning algorithm to acquire the purposive behaviors in Chapters 3 and 4. The demerits of this method are

- to modify the discounted factor γ appropriately, and

[4]It seems too difficult for the agent to shoot a ball without collisions with the defender which moves with the maximum speed.

[5]The value function sometimes fluctuates when the negative reward is given to the agent.

- to store the expected physical number of steps to reach the goal.

Therefore, this method is said to be in *extensive* form. As compared with the such methods, the proposed method might be promising for integration of multiple behaviors.

Fig.5.6 shows an example behavior, where **r0** shoots the ball into the goal avoiding collisions with **r2** (defender). After **r0** push the ball near **r2** at (see 5), **r0** does not approach the ball in order to avoid collisions with **r2** (see 6 and 7). Consequently, the ball rolls towards the goal.

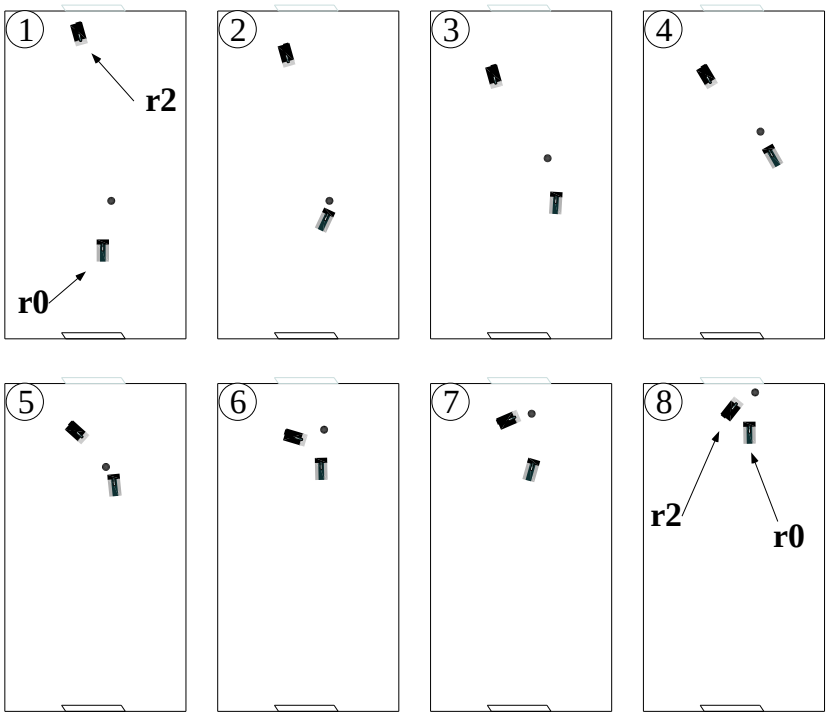
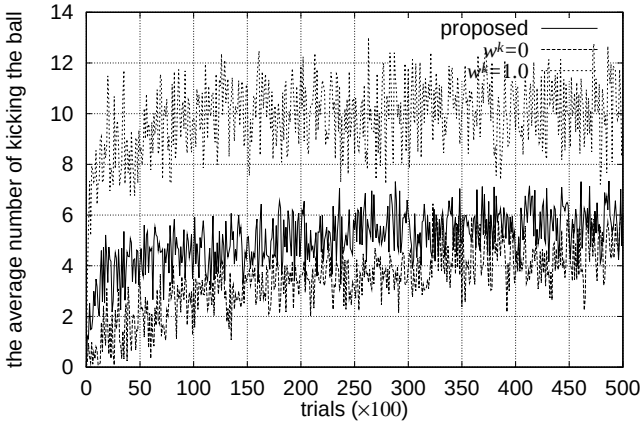
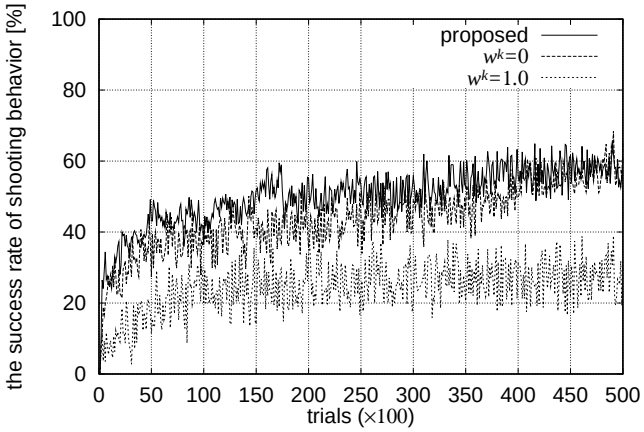


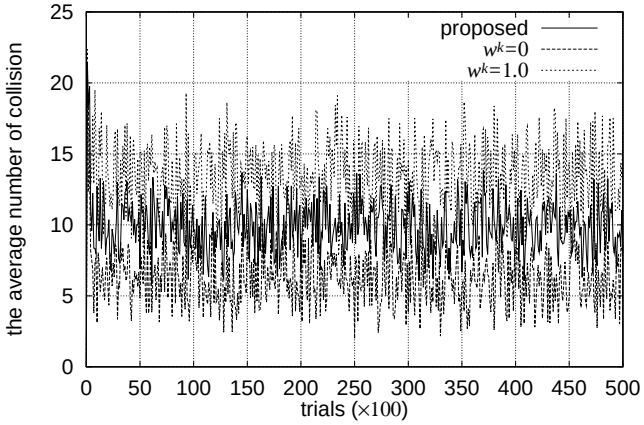
Fig.5.6 Acquired shooting and avoiding behavior in computer simulation



(a) the number of kicking the ball



(b) the success rate of shooting the ball into the goal



(c) the number of collisions

Fig.5.7 Experimental results of the acquired performance in a case of shooting and avoiding behaviors

5.5.3 Shooting and Passing a Ball without Collisions

In this experiment, there are two learners. The tasks for both learners are (i) to shoot a ball into the goal, (ii) to avoid collisions, (iii) to kick the ball, and (iv) to pass the ball to the teammate. The fourth reward function is a kind of cooperative one. Then, we test the coordination of shooting, passing and avoiding behaviors. We use the reward function common to them. That is, it is slight different from the case of Chapter 3. The reward function is

$$\boldsymbol{r} = \begin{bmatrix} r^c \\ r^s \\ r^p \\ r^k \end{bmatrix} \begin{array}{ll} \cdots & \text{the negative reward for collisions,} \\ \cdots & \text{the positive reward for success of shooting,} \\ \cdots & \text{the positive reward to pass the ball,} \\ \cdots & \text{the positive reward for kicking the ball.} \end{array} \tag{5.17}$$

We assign the positive reward $r^p(= 1.0)$ when the pass behavior is accomplished. The definition of the pass behavior in this experiment is the same as that of Chapter 3, that is, the agent A receives r^p after the other agent touches the ball which is pushed by the agent A in a short time. Other rewards r^c, r^k and r^s are the same values as described above, respectively. Further, we select the discounted matrix,

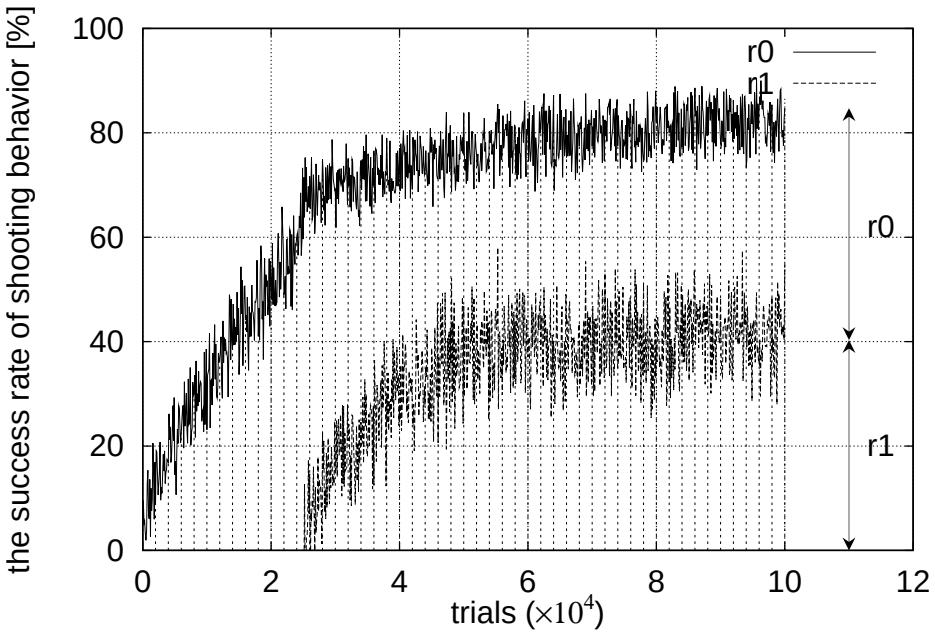
$$\boldsymbol{\Gamma} = \mathrm{diag}\begin{bmatrix} 0.3 & 0.9 & 0.9 & 0.6 \end{bmatrix}.$$

In this experiment, since there are multiple agents that learn behaviors, simultaneous learning may cause poor performance, especially in the early stage of learning. Then, we apply the learning schedule proposed in Chapter 3. We set up the same learning schedule described in Section 3.5.3, that is,

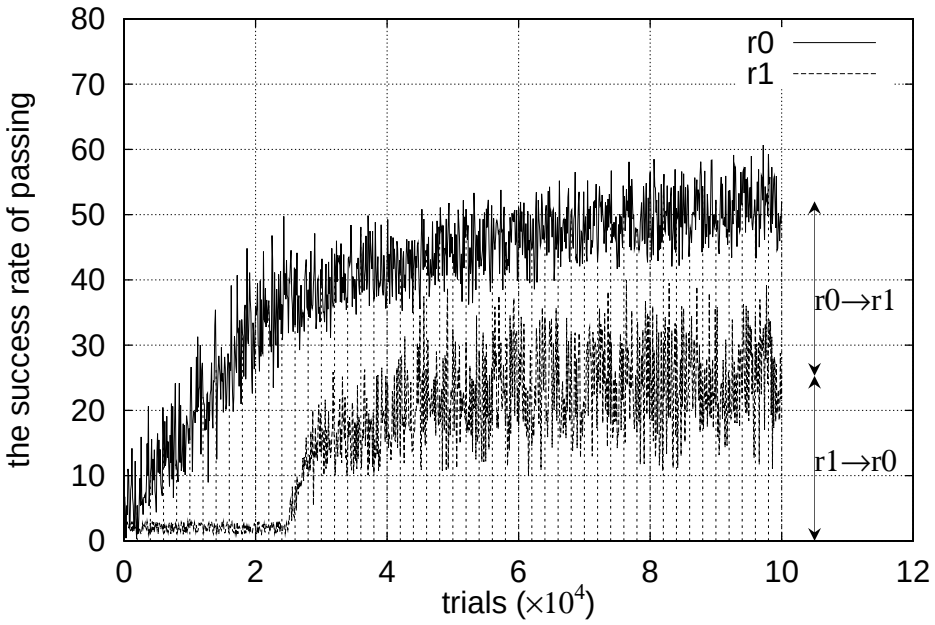
- **period A** (trial number $0 \sim 250 \times 10^2$) : **r0** is a learner while **r1** is stationary,
- **period B** (trial number $250 \times 10^2 \sim 500 \times 10^2$) : **r1** is a learner while **r0** moves around based on the result and policy obtained during the period **A**,
- **period C** (trial number $500 \times 10^2 \sim 750 \times 10^2$) : **r0** is selected as a learner again while **r1** moves around, and
- **period D** (trial number $750 \times 10^2 \sim 1000 \times 10^2$) : **r1** construct the *LPMs*, and learn the behaviors again.

Fig.5.8 shows the learning curves with respect to the frequencies of the reception of the rewards r^s and r^p. We omit the learning curves of r^c and r^k since the curves seem to be the almost same trend as the curves described in Section 5.5.2. As we can see from Fig.5.8(a), the success rate of shooting behavior gradually increased through the interactions. Until the end of the period **A**, only **r0** shot the ball into the goal because **r1** did not move in this period.

It follows from Fig.5.8(b) that the frequencies of passing behaviors also increased. The passing behaviors not only from **r0** to **r1** but also from **r1** to **r0** are reinforced in the



(a) shooting behavior



(b) passing behavior

Fig.5.8 Experimental results of the acquired performance in a case of shooting and avoiding behaviors

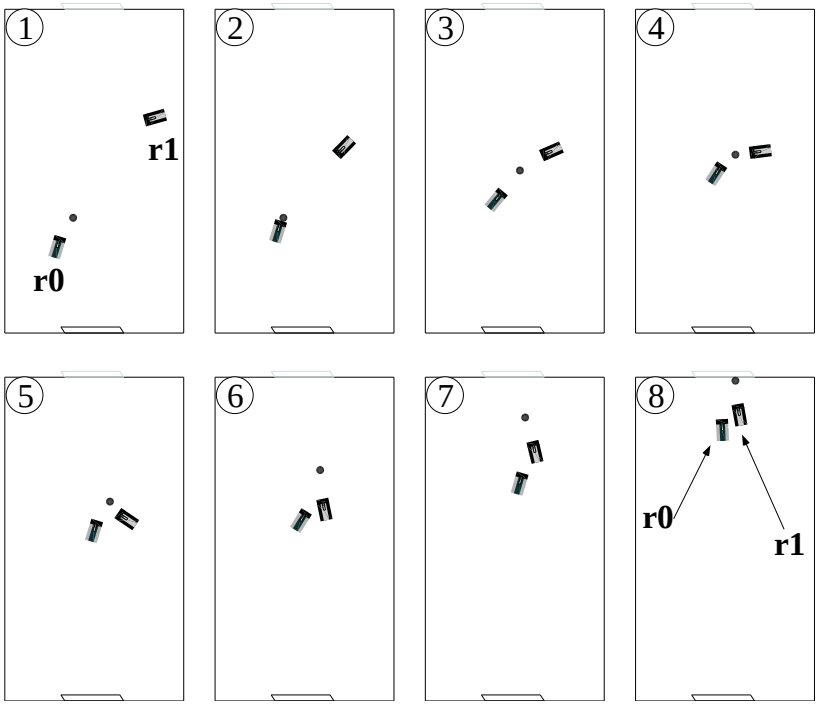


Fig.5.9 Acquired cooperative behavior (shooting and passing) in computer simulation

period **A**. Even though **r1** was stationary, the ball was sometimes run into the **r1** by accident. Then, the r^p was given to **r1** after **r0** pushed the ball.

We show an acquired behavior in **Fig.5.9**. As we can see from this figure, both robots tend to push the ball. For instance, after **r0** received the reward r^p at Fig.5.9 (5), **r0** follows **r1** in order to shoot the ball into the goal. The reason is is that **r0** does not know the policy of **r1** explicitly in the current algorithms, If **r0** can predict the behavior of **r1**, **r0** has a chance to select another action so as not to disturb the behavior of **r1**. Reactive planning based on the *LPMs* might be promising in order to realize such a purposive action selection mechanism.

5.5.4 A Simplified Soccer Game among Three Robots

Finally, we perform three-robots' experiments. This experiment involves the cooperative and competitive tasks. **r0** and **r1** are teammates. We add a new reward function r^l to Eq.(5.17). The six dimensional reward vector is

$$\boldsymbol{r} = \begin{bmatrix} r^c \\ r^l \\ r^m \\ r^s \\ r^p \\ r^k \end{bmatrix} \begin{array}{ll} \cdots & \text{the negative reward for collisions,} \\ \cdots & \text{the negative reward for losing scores,} \\ \cdots & \text{the negative reward to pass the ball to the opponent,} \\ \cdots & \text{the positive reward for success of shooting,} \\ \cdots & \text{the positive reward to pass the ball the teammate,} \\ \cdots & \text{the positive reward for kicking the ball.} \end{array} \tag{5.18}$$

We assign the negative reward $r^l(= -1.0)$ when the team to which the agent belongs lose the goal. Other rewards r^c, r^s, r^p and r^k are the same value used before, respectively. Furthermore, we choose the discounted matrix,

$$\boldsymbol{\Gamma} = \mathrm{diag}\left[\begin{array}{cccccc} 0.3 & 0.9 & 0.9 & 0.9 & 0.9 & 0.6 \end{array}\right].$$

We apply the learning schedule in the same way to make learning stable in the early stage. We set up the following three learning schedules,

- case (a) : r0 → r1 → r2
- case (b) : r2 → r0 → r1
- case (c) : r0 → r2 → r1 → r2

Each interval has the same length in Section 5.5.3. After each agent learned the behaviors (all the agent was selected at once), we recorded the total scores in each game. **Fig.5.10** shows the histories of the game.

As we can see from this figure, the obtained result depends on the order to learn. Although this game is two-to-one competition, **r2** won the game if we selected **r2** as the first agent to learn. Otherwise, a team of **r0** and **r1** defeated **r2**. Even though **r2** was selected twice in one cycle (case (c)), **r2** could not win the game.

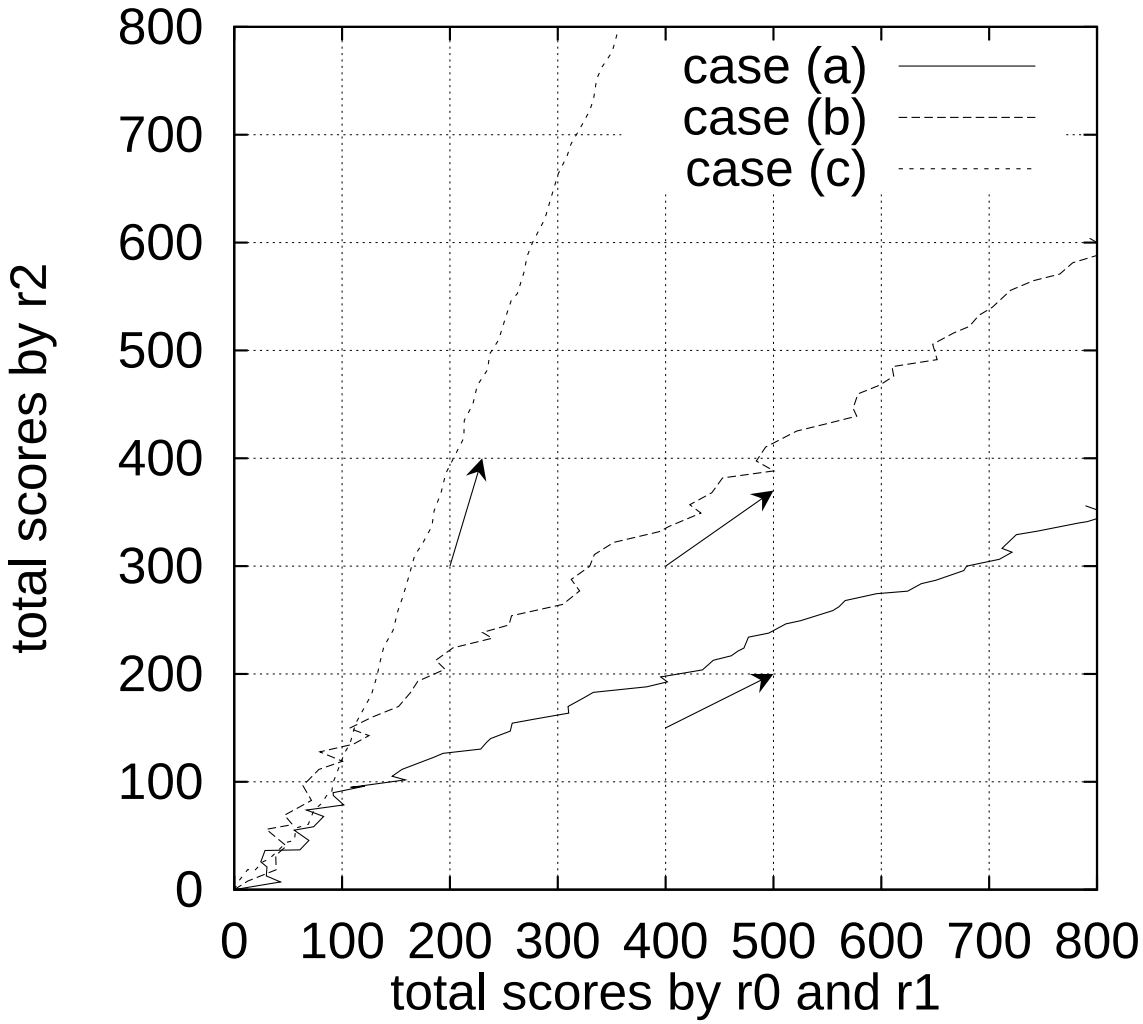


Fig.5.10 Curves of scores

5.6 Discussions and Future Works

This chapter has shown how the learning agent cope with the multiple tasks in multiagent environment. In order to realize cooperation, global evaluation factors are added to the reward function. In other words, the task for the agent is to consider the tradeoff between the individual evaluation and the global one. One of the simplest implementation of the multiple rewards is to sum up all the rewards from the environment. However, this method sometimes leads to poor estimation.

Instead, we proposed the vector-valued reward estimator to evaluate the multiple rewards. Although this method needs much more memory than the weighting method, the vector-valued reward estimator can distinguish the reward. We apply the proposed method to several simplified soccer games, and demonstrate that the learning agents can acquire the purposive behaviors taking account of the tradeoff between individual and group goals. Now, we are planning to implement real experiments to check the validity of the proposed method and the obtained behaviors. There are two main issues to be considered.

Design of the Discounted Matrix

In this chapter, we use the diagonal matrix as $\boldsymbol{\Gamma}$ because it is easy to understand the meaning of $\boldsymbol{\Gamma}$. $\boldsymbol{\Gamma}$ can be regarded as a kind of interaction among given tasks. It is not evident how the general $\boldsymbol{\Gamma}$ will affect learning of actor-critic expressed by Eqs.(5.8) and (5.9). We have to make clear that how we select $\boldsymbol{\Gamma}$.

The Merits and Demerits of the Learning Schedule

If the multiple robots learn the behaviors simultaneously, the learning process may be unstable, especially in the early stage of learning. Then, we have already proposed the learning schedule to make the learning processes stable in Chapter 3. We have applied this method to some experiments in Sections 5.5.3 and 5.5.4 as the same way. This scheduling is a kind of teaching, help the agents to search the feasible solutions from a viewpoint of the designer. The results in Chapter 3 showed the validity of this scheduling when the task is cooperative one.

However, the demerits of this method is also revealed when we apply it to the competitive tasks described in Section 5.5.4. That is, the learning schedule often leads the competitive game to the local maxima. Therefore, we need to extend the learning schedule to the one including competitive situations. We will discuss the alternative in next Chapter 6 from a viewpoint of evolutionary approaches.

As future works, we are planning to extend our method in order to predict the behavior of other agents from a viewpoint of the reward functions.

Chapter 6

Co-Evolution for Cooperative and Competitive Behavior Acquisition

6.1 Introduction

In this chapter, we discuss how multiple agents can acquire cooperative and competitive behaviors through interactions. As stated in the previous chapter, it seems difficult to apply conventional learning algorithms such as reinforcement learning to co-evolution of cooperative agents since the environment including other agents may cause unpredictable changes in state transitions for learning agents. We have shown reinforcement learning supported by *Local Prediction Models* (hereafter *LPMs*) and learning schedule in Chapter 3. This method estimated the relationships between learner's behaviors and other robot ones through interactions. In this method, only one robot may learn and other robots had to fix their policies for successful learning to converge.

Recently, emergence of cooperative behaviors between multiple robots has been receiving increased attention as a problem of multiagent simultaneous learning. In the realm of nature, we can see various aspects of behaviors emerged in multiagent environments, not only competition but also cooperation, ignorance, and so on. That means there could be artificial co-evolution for other than competition. This chapter discusses how multiple robots can obtain cooperative behaviors through co-evolutionary processes. As a task example, a simplified soccer game with three learning robots is chosen and a *Genetic Programming* (hereafter *GP*) method [32, 33] is applied so as to experimentally evaluate obtained behaviors in the context of cooperative and competitive tasks. Each robot has its own individual population, and attempts to acquire desired behaviors through interactions with its environment that is ever changing in the co-evolutionary process. The complexity of the problem can be explained twofold:

1. Co-evolution for cooperative behaviors needs exact synchronization of mutual evolutions.

2. Three robot co-evolution requires well-complicated environment setups that may

contribute to providing a wide variety of searching area from simpler to more complicated situations in which they seek for better strategies so that they can emerge cooperative and competitive behaviors simultaneously.

The rest of this chapter is organized as follows. First, we describe our views on co-evolution in the context of cooperative and competitive tasks. Next, we explain our task example, a simplified soccer game in which cooperative and competitive tasks are involved. Then, we give a brief implementation of *GP* and two fitness functions: one is fixed and the other varying. Finally, the results of computer simulation are shown, and a discussion is given.

6.2 Co-Evolution in Cooperative Tasks

Generally, we have the following three difficult problems in multiagent simultaneous learning:

1. **Unknown Policy**
   Learning agents do not know other agents' policies in advance, therefore they need to estimate them through observations and actions. What's the worse is that the agent policies may change through a learning process.

2. **Synchronized Learning**
   Mutual learning robots have to improve their learned policies simultaneously. If the opponent learning converged much earlier than itself, one robot could not improve its strategy against the difficult environment that its opponent has already fixed.

3. **Credit Assignment**
   Credit assignment to learning robots for cooperation seems difficult. If the credit involves group evaluation only, one robot may accomplish a given task by itself and others do just actions irrelevant to the task as they do not seem to interfere the one robot's actions. Else, if only individual evaluation is involved, robots may compete each other. This trade-off should be carefully dealt.

Co-evolution is one of potential solutions for the first problem by seeking for better strategies in a wide range of searching area in parallel. The second and third ones might be solved by careful designs of environmental setups and fitness functions. Emerging patterns by co-evolution can be categorized into three.

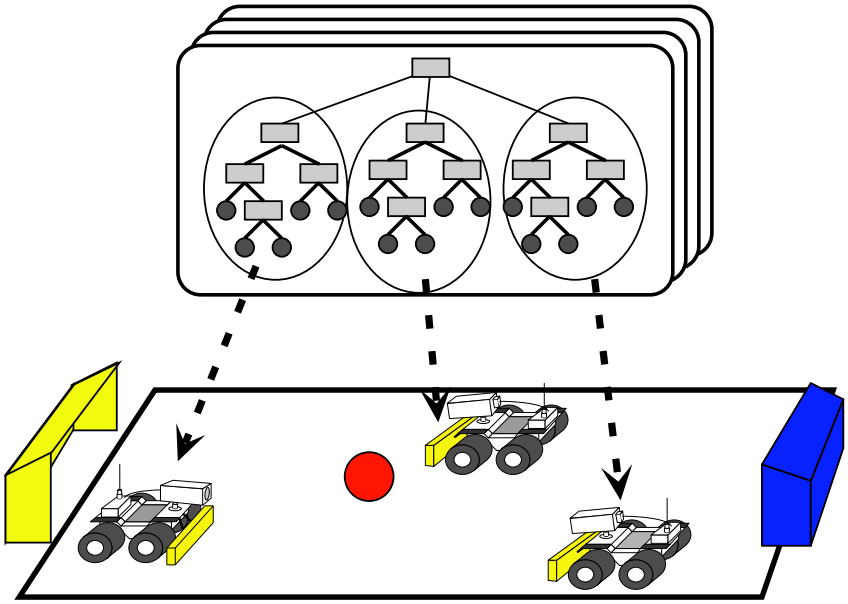
1. **Cycles of switching fixed strategies**
   This pattern can be often observed in a case of a prey and predator which often shift their strategies drastically to escape from or to catch the opponent. The same strategies iterate many times and no improvements on both sides seem to happen.

2. **Trap to local maxima**
   This corresponds to the second problem stated above. Since one side overwhelmed its opponents, both sides reached to one of stable but low skill levels, and therefore no change happens after this settlement.

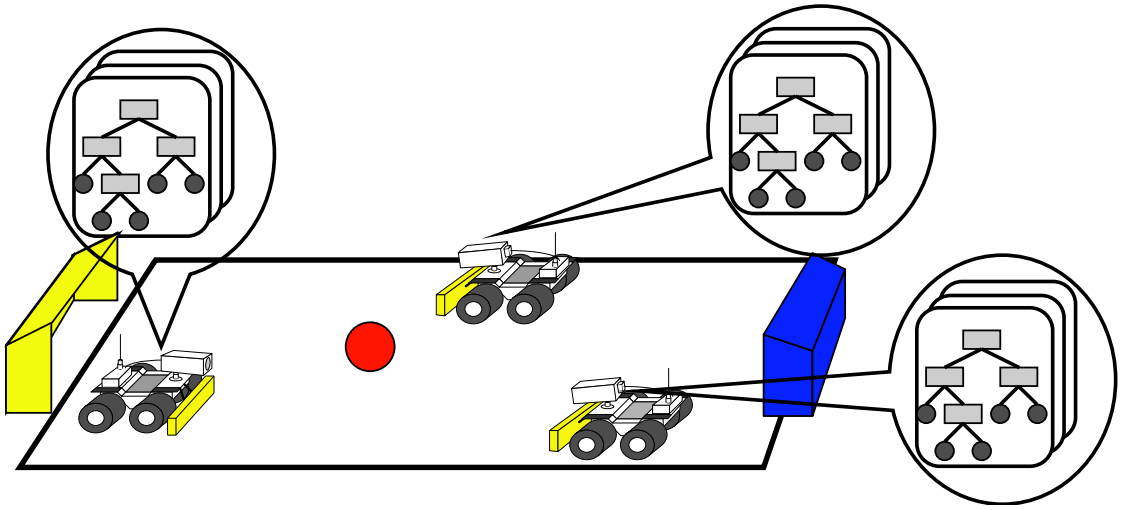
3. **Mutual skill development**
   In certain conditions, every one can improve its strategy against ever-changing environments owing to improved strategies by other agents. This is real co-evolution by which all agents evolve effectively.

As a typical co-evolution example, a competitive task such as prey and predator has been often argued [12, 18] where heterogeneous agents often change their strategies to cope with the current opponent. That is, the first pattern was observed. In a case of homogeneous agents, Luke *et al.* [39] co-evolved teams consisting of eleven soccer players among which cooperative behavior could be observed. However, co-evolving cooperative agents has not been addressed as a design issue on fitness function for individual players since they applied co-evolving technique to teams (See **Fig.6.1** (a)).

We believe that between one-to-one individual competition and team competition, there could be other kinds of multiagent behaviors by co-evolutions than competition. Here, we challenge to evaluate how the task complexity and fitness function affect co-evolution processes in a case of multiagent simultaneous learning for not only competitive but also cooperative tasks through a series of systematic experiments. Fig.6.1 (b) shows an our basic approach. First, we show the experiments for a cooperative task, that is, shooting supported by passing between two robots in Section 6.4.1 where unexpected cooperative behavior regarded as the second pattern was emerged. Next, we introduce a stationary obstacle in front of the goal area into the first experimental set up in Section 6.4.2 where the complexity is higher and an expected behavior was observed after longer generation changes than the previous one. Finally, we exchange an active learning opponent with the stationary obstacle to evaluate how both cooperative and competitive behaviors are emerged in Section 6.4.3. We have tried several fitness functions, and we may conclude that the same level fitness functions among them seems better to co-evolve cooperative and competitive agents, and different ones tend to evolve only one side, that is the second pattern.



(a) Luke's approach [39]: a team as an individual



(b) Our approach: each agent has its own population

Fig.6.1 Difference between the previous method and ours

6.3 Task and Assumptions

6.3.1 Environment and Robots

Before explanation of the proposed method, we show a concrete task for reader's understanding of the method. We have chosen a simplified soccer game consisting of two or three robots as a testbed for the problem because both competitive and cooperative tasks are involved as stated in RoboCup Initiative [21]. The environment consists of a ball and two goals, and a wall is placed around the field except the two goals. The sizes of the ball, the goals and the field are the same as those of the middle-size real robot league of RoboCup Initiative. **Fig.6.2** shows the size of the environment and the robot used for modeling.

The robots have the same body (power wheeled steering system) and the same sensor (on-board TV camera), that is, homogeneous agents. In this simulator, the robot can not obtain the complete information because of limitation of its sensing capability and occlusion of the objects.

6.3.2 Function and Terminal Sets

It is essential to design the well-defined function and terminal sets for appropriate evolution processes. This can be regarded as the same problem to construct the well-defined state space, which has been argued in Chapter 3.

As sets of functions, we prepare a simple conditional branching function "IF_a_is_b" that executes its first branch if the condition "a is b" is true, otherwise executes its second branch, where a is a kind of image features, and b is its category. **Table 6.1** shows these features and their categories.

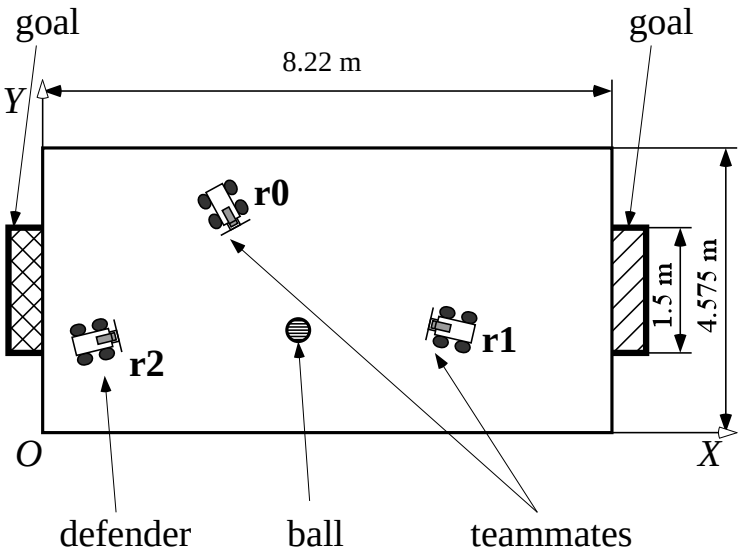
Terminals in our task are actions that have effects on the environment. A terminal set consists of the following four behaviors based on the visual information:

1. `shoot` : the robot shoots a ball into the opponent goal.
2. `pass` : the robot kicks a ball to one teammate.
3. `avoid` : the robot avoids collisions with other robots.
4. `search` : the robot searches the ball by turning to left or right.

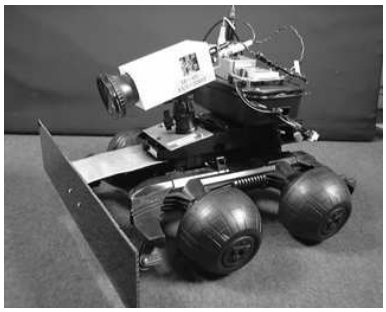
These primitive behaviors have been obtained by the proposed learning algorithms described in Chapter 3.

6.3.3 Fitness Measure

Another issue to apply an evolutionary algorithm is the design of fitness function which leads robots to appropriate behaviors. It is so-called "credit assignment problem" [15] in



(a) The size of an environment



(b) A real robot used for modeling

Fig.6.2 Three robots and the environment

Table 6.1 Function sets

a	ball, goal, other robot 0, other robot 1, $\cdots$
b	left, middle, right, small, medium, large, lost

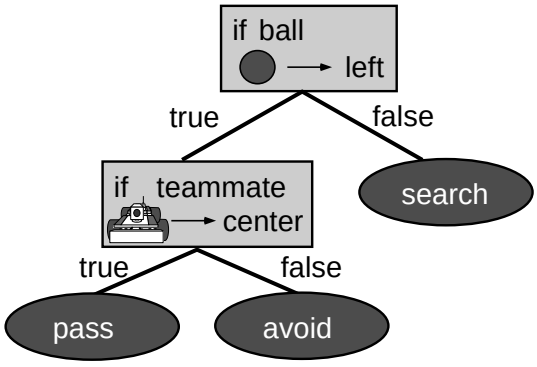


Fig.6.3 An example of a tree controller

the field of the reinforcement learning: when a trial is complete, which agents get more credit (reward) for its success or failure?

We utilize the standardized fitness representation, that has a positive value. The smaller is the better (0.0 is the best). We first consider the following parameters to evaluate team behaviors such as cooperation between teammates and competition with opponents:

- $G(i)$: the total number of achieved goals for the team to which robot i belongs,
- $L(i)$: the total number of lost goals for the team to which robot i belongs.

With these parameters only, most robots tend to be idle (passive cooperation) except one that attempts at achieving the goal for itself, and therefore no active cooperation has been seen. Then, we introduce the following more individual evaluation to encourage robots to interact with each other while minimizing the number of collisions:

- $K(i)$: the total number of ball-kicking by robot i,
- $C(i)$: the total number of collisions between robot i and others.

In addition to the above, the following is involved to make robots achieve the goal earlier.

- *steps* : the total number of steps until all trials[1] end, where a step is defined as a time period for one action execution against the sensory input of a robot (1/30 [msec]).

Next, we combine these fitness measures to evaluate individuals. Angeline and Pollack [2] point out that a *competitive fitness function* is useful instead of the explicit fitness function. Because the competitive fitness function is based on the result of "competition" between individuals, it can be applied to acquire the *GP* tree when the all agents are considered as one *GP* individual. However, in our case, it is not useful because this method can not cope with the "credit assignment problem" between agents. For example, suppose the case of three robots game like Fig.6.2 (a). How should we evaluate the contribution of the robots when the robots 0 and 1 win the game? Thus, we examine the following two fitness function in the following experiments.

Fixed Fitness Function

The simplest fitness function is calculated by a linear combination of these parameters. In this method, the weights of the fitness function are fixed over the all generations. The fitness value which the robot i receives is given by:

$$\begin{aligned} f^f(i) &= \alpha_g^f \cdot h(G(i), T_{\max}) + \alpha_l^f \cdot L(i) \\ &\quad + \alpha_k^f \cdot h(K(i), \beta^f) + \alpha_c^f \cdot C(i) + \alpha_s^f \cdot steps, \end{aligned} \tag{6.1}$$

[1] One trial is terminated if one of the robots shoots a ball into the goal or the pre-specified time interval expires.

and $h(x, y)$ is a threshold function,

$$h(x,y) = \begin{cases} y - x & \text{if } x < y \\ 0 & \text{otherwise,} \end{cases}, \tag{6.2}$$

where $T_{\max}$, $\alpha_k^s \sim \alpha_s^s$, and β^s denote the maximum number of trials, and constants.

Varying Fitness Function

In a case of a fixed fitness function, we do not consider the change of each measure. Ideally, the individual factor (K) and the penalty (C and *steps*) are kept high and low, respectively, at the beginning of the evolution by *GP* in order to reduce the search area.

As the evolution proceeds, the individual factor decreases gradually with the generation index while the penalty factors increase. Near the end of the run, the individual factors reach appropriately small values. As one of such fitness function, a linearly varying fitness function can be expressed as

$$\begin{aligned} f^v(i) &= \alpha_g^v \cdot h(G(i), T_{max}) + \alpha_l^v \cdot L(i) \\ &\quad + \frac{G-g}{G} \cdot \alpha_k^v h(K(i), \beta^v) + \frac{g}{G} \cdot \alpha_c^v \cdot C(i) + \frac{g}{G} \cdot \alpha_s^v \cdot steps, \end{aligned} \tag{6.3}$$

where g and G are the generation index and the maximum number of generations, respectively.

Table 6.2 shows the parameters which have been determined empirically. If two or more individuals have the same fitness value, we prefer to one with more compact tree depth.

Table 6.2 The parameters used in the fitness functions

fitness function	α_g	α_l	α_k	α_c	α_s	β
fixed	1.0	0.5	1.0×10^{-3}	5.0×10^{-3}	1.0×10^{-4}	4000
varying	1.0	1.0	1.0×10^{-2}	1.0×10^{-2}	1.0×10^{-2}	4000

6.3.4 The GP Implementation

Other parameters in *GP* used are shown in **Table 6.3**. The best performing tree in the current generation will survive in the next generation. In order to select parents for crossover, we use tournament selection with size 10. After each population selects one individual separately, the selected individuals participate in the game. We perform 20 games to evaluate them. As a result, it needs 1600 trials to alter a new generation. The hardware used for the simulation is DEC VT-Alpha 600, which takes about 16 hours to evaluate one experiment whole generations.

Table 6.3 Other parameters used in *GP*

the size of each population	80
the number of generations for which the evolutionary process should run	60
the maximum depths during the creation	10
the maximum depths by crossing two trees	25
the crossover probability	95 %
the reproduction probability	5 %
the mutation probability	10 %

6.4 Experimental Results

6.4.1 Two Learners

At first, we demonstrate the experiments to acquire cooperative behaviors between two robots. Both robots belong to the same team, and they obtain the score if they succeed in shooting a ball into the goal. The number of function sets is 28 (= 7 (ball) $+2 \times 7$ (two goals) +7 (teammate)).

The tree depths and the numbers of nodes in cases of the fixed and varying fitness functions are shown in **Table 6.4**. The tree of the best **r0** (expected to be a passer) is deeper than that of the best **r1** (expected to be a shooter) in the fixed fitness functions, but the average depth and the number of nodes of **r0** are smaller than those of **r1** in both fitness functions. Actually, the acquired behavior of **r1** is purposive while **r0** does not move appropriately from a viewpoint of the designer.

One of the successful behaviors based on the fixed fitness function are shown in **Fig.6.4**. In this case, **r0** does not kick the ball by itself but shakes its body by repeating the behaviors `search` and `avoid`. On the other hand, **r1** approaches the ball and passes the ball to **r0**. After **r0** receives the ball, it executes a `shoot` behavior. However, **r1** approaches the ball faster than **r0**. As a result, **r1** shoots the ball into the goal while **r0** avoids collisions with **r0**. We checked the case of the varying fitness functions, and found that the resultant behaviors were similar to the behavior by the fixed case. In this task, the best **r0** does not kick the ball toward **r1** at the end of the generations.

Figs.6.5 and **6.6** show the results of evolution process based on the fixed and varying fitness functions, respectively. Although both robots kick the ball frequently until 25th generation, $K(\mathbf{r1})$ is gradually decreased according to the evolutionary process (See (d)). Because the fixed fitness function ignore $C(i)$ and *steps* implicitly[2], the values of them are larger than those of the varying fitness function (See (e) and (f)).

We suppose that the reasons why they acquire such behaviors are as follows:

- In order for **r0** to survive by passing the ball to **r1**, **r1** has to shoot the ball which is passed back from **r0**. This means that the development of both robots needs to be exactly synchronized. It seems very difficult for such a synchronization to be found.

- **r1** may shoot the ball by itself whichever **r0** kicks the ball or not. In other words, **r1** does not need the help by **r0**.

In this task, **r0** and **r1** do not have even complexities of the tasks. As a result, the behavior of **r1** dominates this task while **r0** cannot not improve its own behavior. This is the second pattern explained in Section 6.2.

[2]The values of α_c^s and α_s^s are much smaller than α_g^s, etc.

Table 6.4 The tree depths and the number of nodes in a case of two learners experiments each content is listed with respect to the fitness of (best, average, worst).

fitness function	the tree depth		the number of nodes	
	r0	**r1**	**r0**	**r1**
fixed	(30, 24.3, 2)	(15, 15.03, 17)	(729, 335.5, 3)	(981, 909.4, 897)
varying	(12, 17.5, 5)	(13, 14.7, 16)	(632, 594.5, 21)	(784, 821.4, 108)

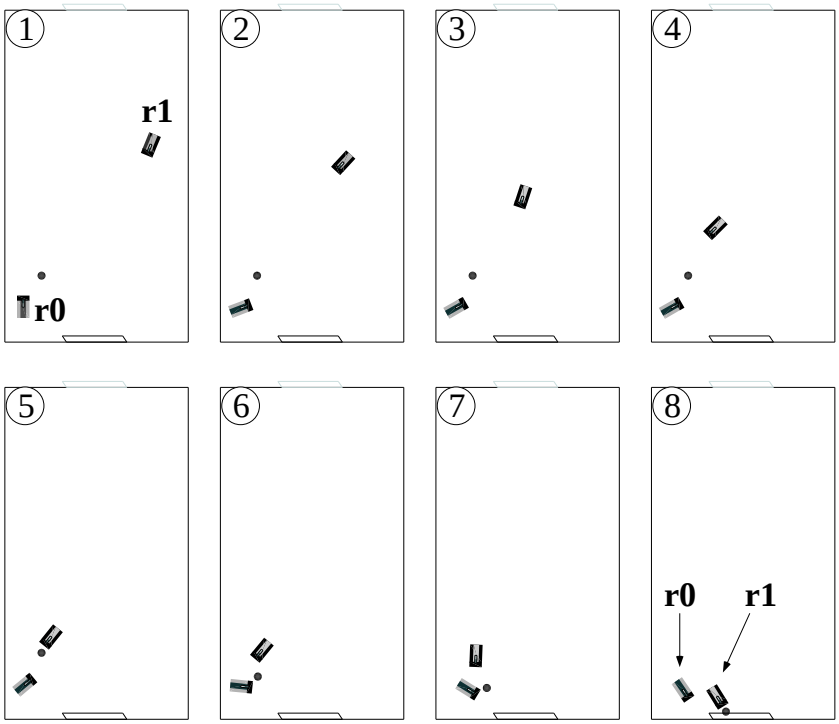
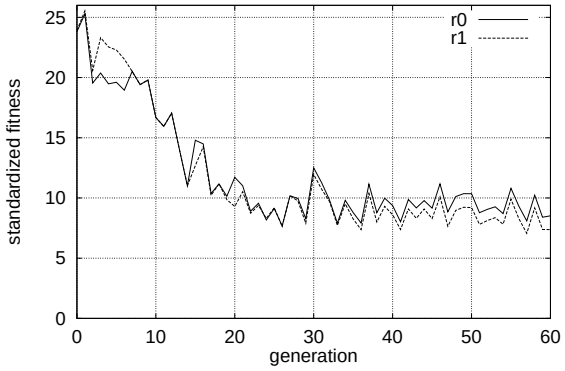
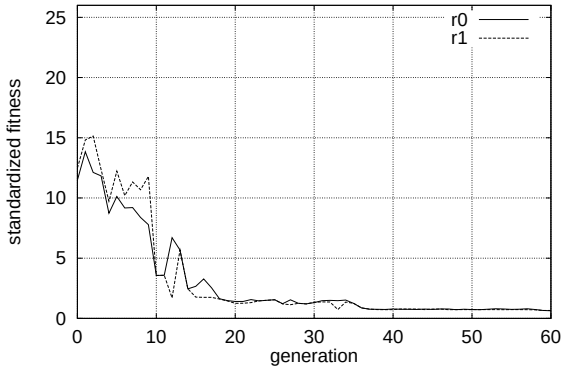


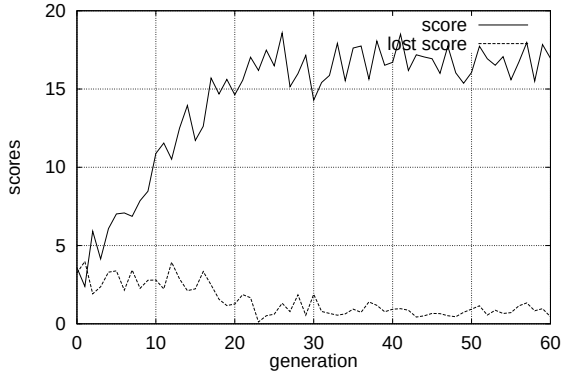
Fig.6.4 Two robots (**r0** and **r1**) succeed in shooting a ball into the goal



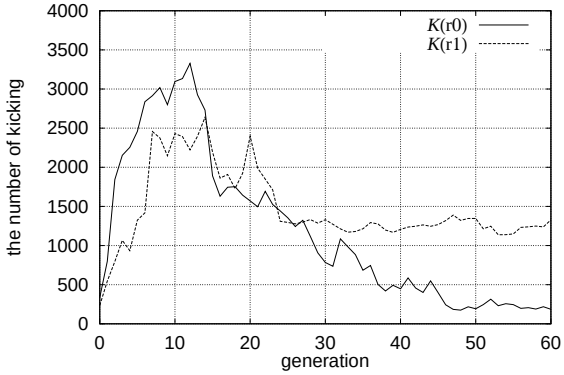
(a) the average fitness $f^f(i)$



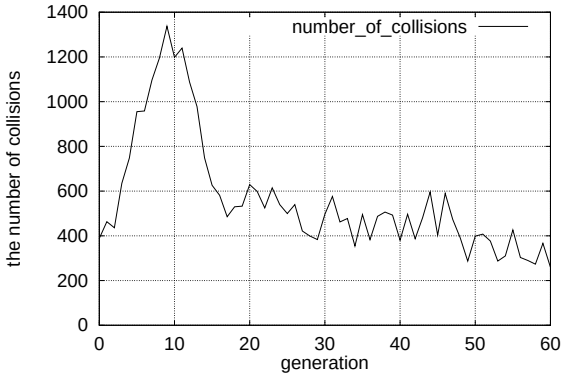
(b) the best fitness $f^f_{\min}(i)$



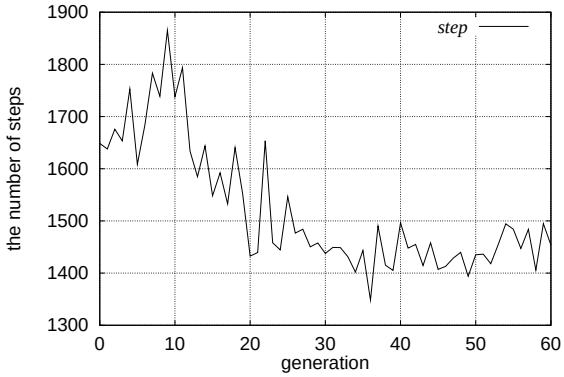
(c) the total number of scores $G(i)$ and $L(i)$



(d) the total number of kicking $K(i)$

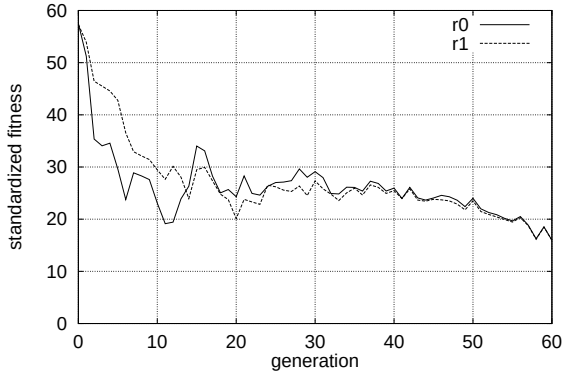


(e) the total number of collisions $C(i)$

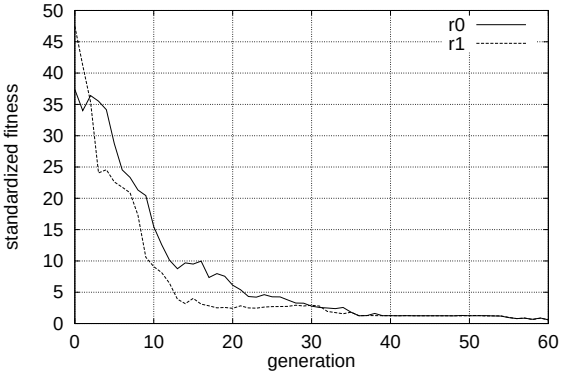


(f) the total number of steps *step*

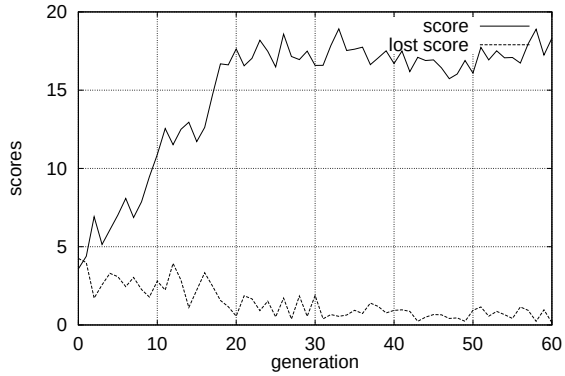
Fig.6.5 Experimental results in a case of two learners with fixed fitness function



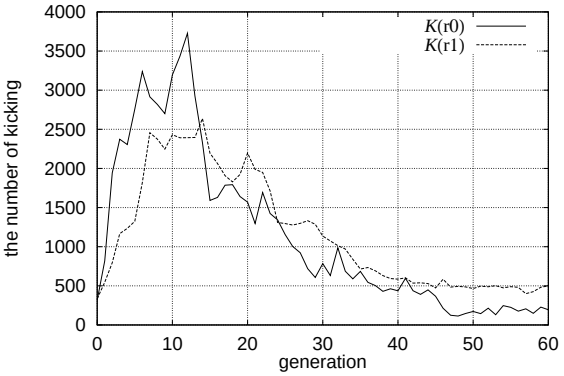
(a) the average fitness $f^v(i)$



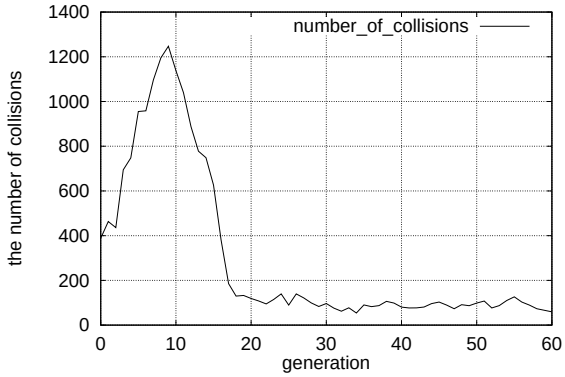
(b) the best fitness $f^v_{\min}(i)$



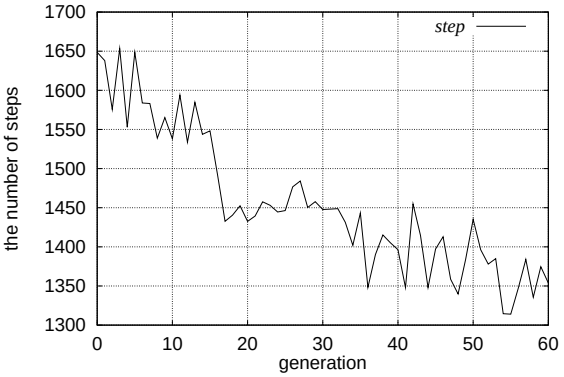
(c) the average number of scores $G(i)$ and $L(i)$



(d) the average number of kicking $K(i)$



(e) the average number of collisions $C(i)$



(f) the average number of steps *step*

Fig.6.6 Experimental results in a case of two learners with varying fitness function

6.4.2 Two Learners and One Stationary Robot

Next, we add one robot as a stationary obstacle to the environment described in Section 6.4.1. The number of function sets is 35 (= 7 (ball) $+2 \times 7$ (two goals) $+2 \times 7$ (teammate and opponent)).

Table 6.5 shows the tree depth and the number of nodes which are obtained by the fixed and varying fitness functions. The *GP* trees which **r1** acquires are more complicated than those of **r0**. The reason is that **r1** has to consider both of the shooting behavior and avoiding collisions with **r2**. On the other hand, the resultant behavior of **r0** is only to push the ball to **r1**. At the end of generations, the number of scores is not different between both fitness functions. However, there are difference with respect to the variety of the final population.

Although both learning robots are placed in the same way as in the previous experiments, the acquired cooperative behaviors are quite different. We found the following three patterns in a case of the fixed fitness function:

1. First pattern (ball rolling and accidental goal)
   Because **r0** is placed near the ball, **r0** pushes the ball more frequently than **r1**. Most of individuals of **r0** kick the ball towards **r1** owing to the initial placement. However, some individuals push the ball towards **r2** in the neighborhood of **r0**. Consequently, the ball rolls towards the goal by accident (See **Fig.6.7**).

2. Second pattern (goal after dribbling along the wall)
   Although both **r0** and **r1** kick the ball until generation 4, **r0** begins to pass the ball towards **r1**. However, **r1** can not shoot the ball from **r0** directly because **r0** cannot pass the ball to **r1** precisely. Therefore, **r1** kicks the ball to the wall and continues to kick the ball to the opponent's goal along the wall until generation 15 (See **Fig.6.8**). After that, the rank of this pattern dropped down.

3. Third pattern (mutual skill development)
   After a number of generations, both robots improve their own behaviors and acquire cooperative behaviors at the end of generations, where **r0** kicks the ball to the front of **r1**, then **r1** shoots the ball into the opponent's goal shown in **Fig.6.9**. As a result, both robots improve the cooperative behaviors synchronously. This is a kind of the mutual development described in Section 6.2.

The individual of the third pattern obtained the high evaluation because it takes much shorter time to shoot the ball than the first and second patterns. **Fig.6.10** shows the results of evolutionary processes where a good synchronization between the best individuals of **r0** and **r1** can be seen. Since it becomes more difficult for **r1** to shoot the ball for itself because of the existence of **r2**, **r1** has to evolve behaviors with **r0** synchronously. In other words, the complexity of the task for **r1** increased around the same level of **r0**.

On the other hand, when we use the varying fitness function, most of the individuals that are obtained at the end of generations is the third pattern shown in Fig.6.9. We

can not find the first and the second patterns based on the varying fitness function. The reason is that the varying fitness function does not permit a wider variety of individuals. As we can see from **Fig.6.11**, the total number of collisions and the total steps decrease gradually because the corresponding terms become effective through evolutionary process. This leads to the extermination of the first and second patterns.

Table 6.5 The tree depths and the number of nodes in a case of two learners experiments each content is listed with respect to the fitness of (best, average, worst).

fitness function	the tree depth		the number of nodes	
	r0	**r1**	**r0**	**r1**
fixed	(13, 15.4, 4)	(23, 17.7, 17)	(989, 1126.9, 7)	(261, 239.7, 125)
varying	(14, 14.7, 8)	(24, 18.3, 12)	(1021, 1328.3, 251)	(1142, 1394.2, 860)

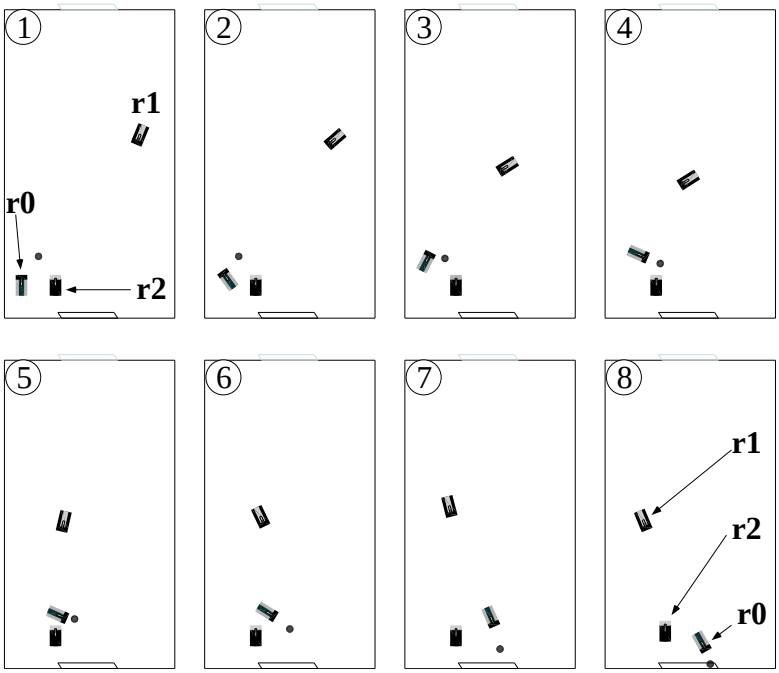


Fig.6.7 r0 shoots the ball into the goal

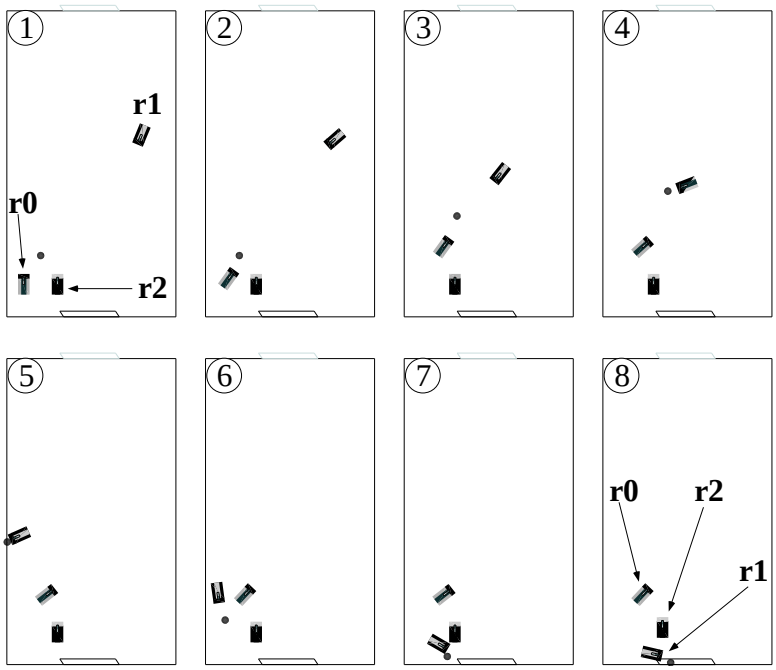


Fig.6.8 r1 shoots the ball into the goal along the wall at generation 15

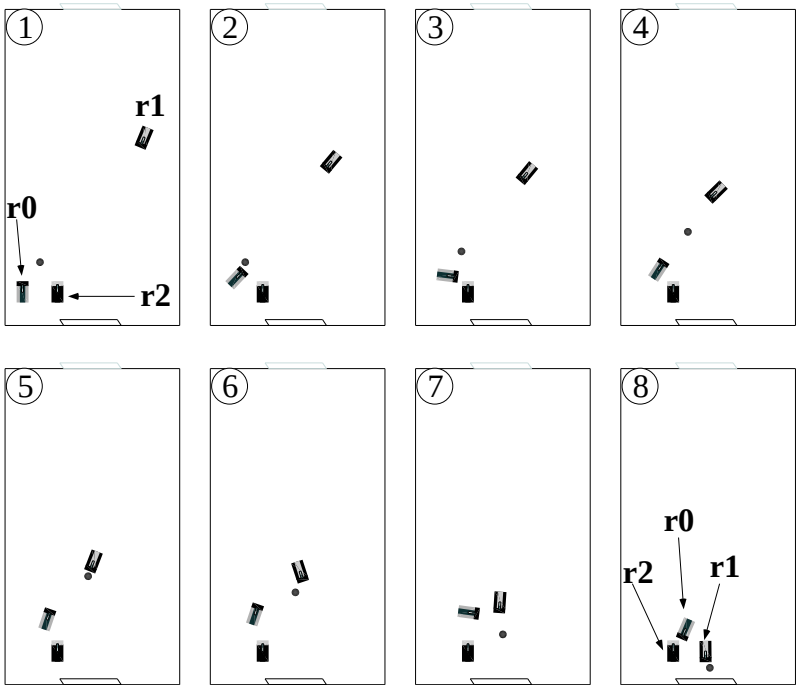
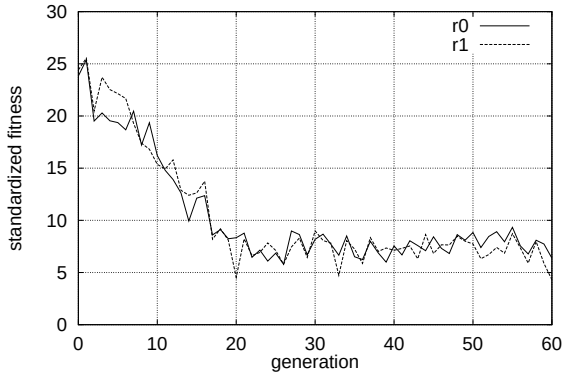
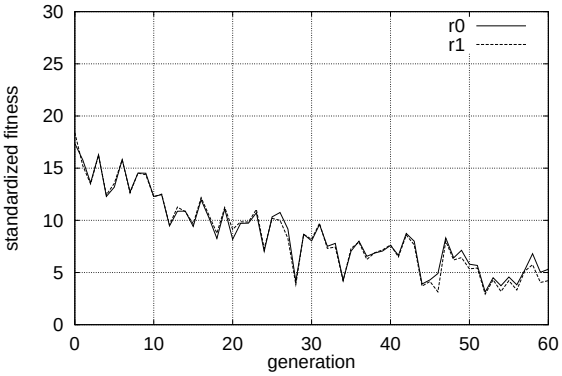


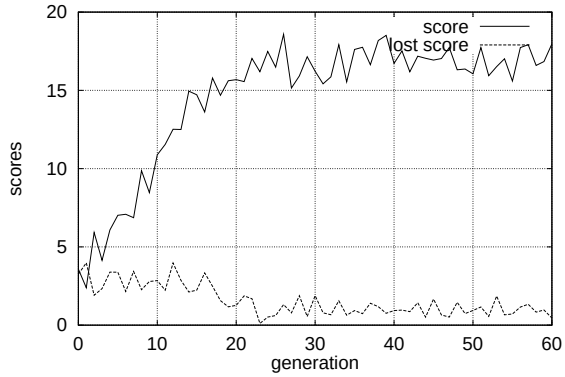
Fig.6.9 After **r0** pushes the ball toward the in front of **r1**, **r1** shoots the ball into the goal avoiding collision with **r2**



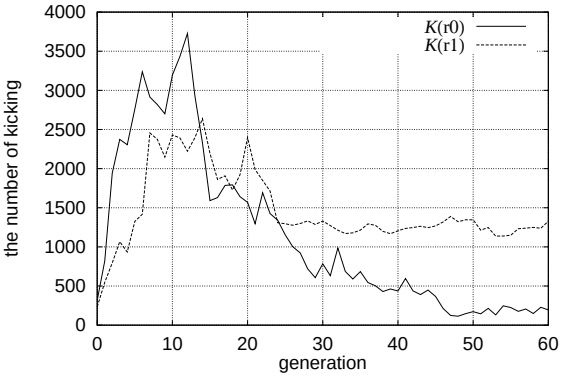
(a) the average fitness $f^f(i)$



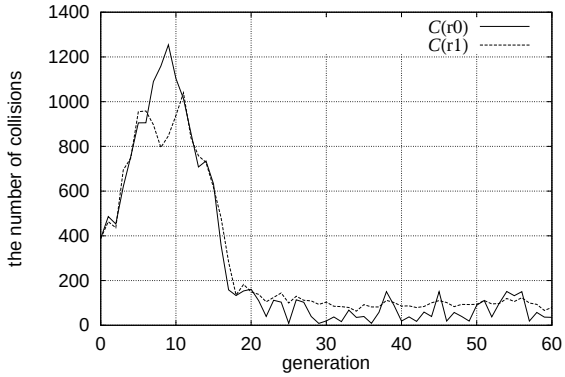
(b) the best fitness $f^f_{\min}(i)$



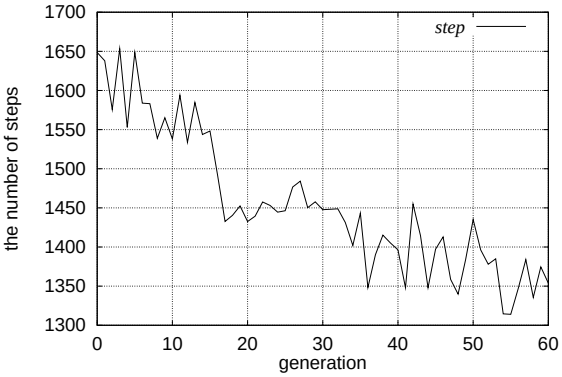
(c) the average number of scores $G(i)$ and $L(i)$



(d) the average number of kicking $K(i)$

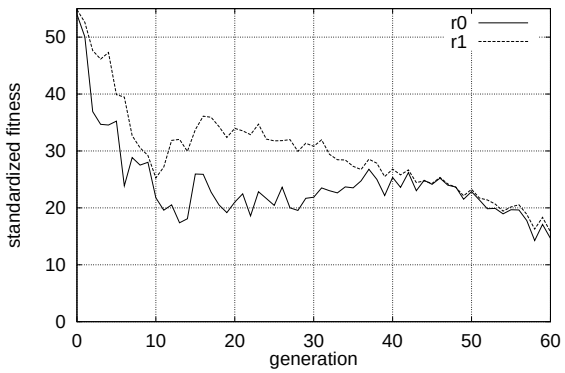


(e) the average number of collisions $C(i)$

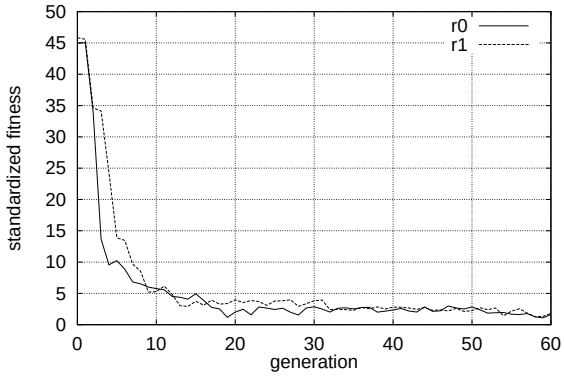


(f) the average number of steps *step*

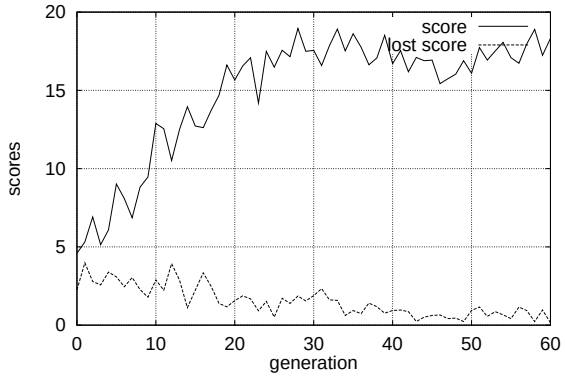
Fig.6.10 Experimental results in a case of two learners and one stationary agent with fixed fitness function



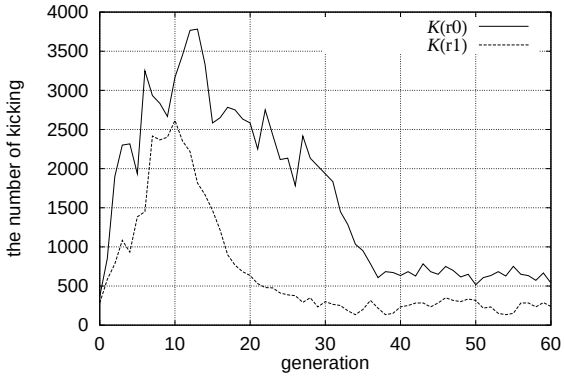
(a) the average fitness $f^v(i)$



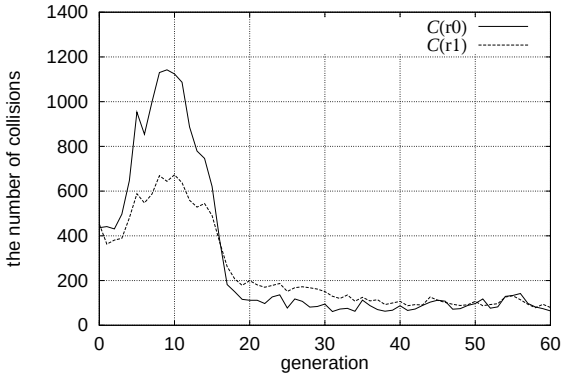
(b) the best fitness $f^v_{\min}(i)$



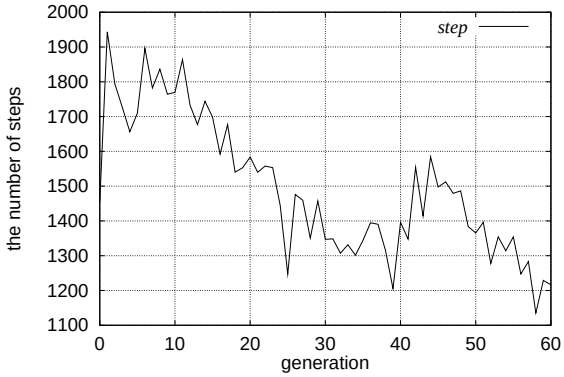
(c) the average number of scores $G(i)$ and $L(i)$



(d) the average number of kicking $K(i)$



(e) the average number of collisions $C(i)$



(f) the average number of steps *step*

Fig.6.11 Experimental results in a case of two learners and one stationary agent with varying fitness function

6.4.3 Three Learners

Finally, we test the co-evolution among three robots. That is, **r2** added in Section 6.4.2 evolves its behavior with **r0** and **r1** simultaneously. The difference from Sections 6.4.1 and 6.4.2 is involvement of competition between **r2** and a team of **r0** and **r1**. The number of function sets is the same as the case of Section 6.4.2.

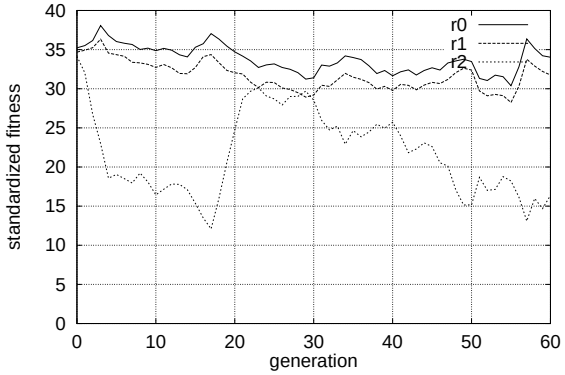
Table 6.6 shows the tree depths and the numbers of the nodes. The acquired *GP* tree of **r1** tends to be simple as compared with the cases of **r0** and **r2**. Furthermore, the results based on the varying fitness function is more complicated than those of the fixed one. However, the varying fitness function leads the *GP* agents to the local solutions as follows.

The results are shown in **Figs.6.12** and **6.13**. As compared with the only cooperative tasks in Section 6.4.2, fitness values rather oscillate than converge stably. Although $C(i)$ and *steps* decrease gradually through the evolution in a case of the varying fitness function, this game is dominated by **r2** at the 18th generation. This phenomenon is observed when we use the fixed fitness function, but the acquired performances of **r0** and **r1** are a little bit better than the a case of the varying fitness function.

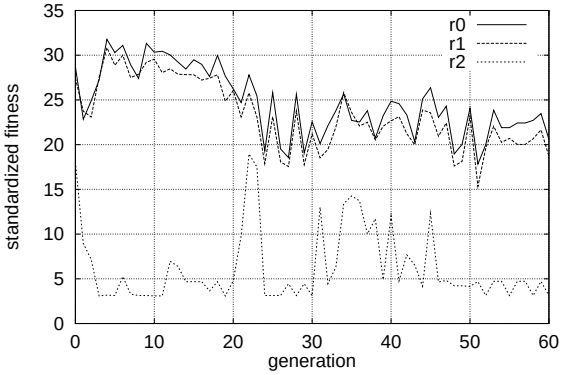
We can see two typical settlements in this three-robot soccer game. One is the same behaviors described in Section 6.4.2 : **r0** kicks the ball toward **r1**, then **r1** shoots the ball into the goal avoiding collisions with **r2** (See **Fig.6.14**). The other one is that **r2** intercepts the ball and shoots the ball into the goal (See **Fig.6.15**). The ratio between the former and the latter is about 25 % : 75 %. It depends on whether **r0** or **r2** achieves its goal. However, **r2** can observe the ball and the opponent's goal at the same time and it may shoot the ball by itself while **r0** needs to pass the ball to **r1**. We suppose that the predominance of **r2** may be caused by the different complexity of the given tasks, that is, task complexities for **r0** and **r1** is higher than that for **r2**.

Table 6.6 The tree depths and the number of nodes in a case of three learners experiments each content is listed with respect to the fitness of (best, average, worst).

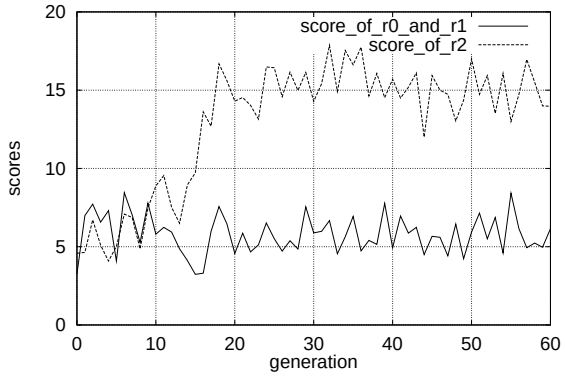
fitness function	the tree depth		
	r0	**r1**	**r2**
fixed	(24, 23.7, 25)	(15, 16.1, 18)	(21, 20.0, 21)
varying	(25, 27.3, 18)	(16, 17.3, 12)	(24, 19.8, 27)
fitness function	the number of nodes		
	r0	**r1**	**r2**
fixed	(1433, 1373.2, 1505)	(1093, 1081.8, 1265)	(749, 772,5, 733)
varying	(1810, 1635.7, 1206)	(1272, 1184.2, 1306)	(1762, 1083.5, 967)



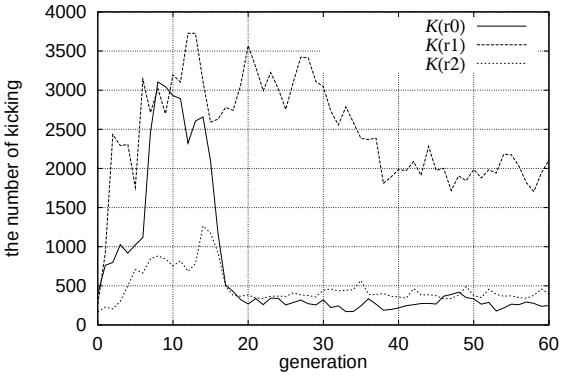
(a) the average fitness $f^f(i)$



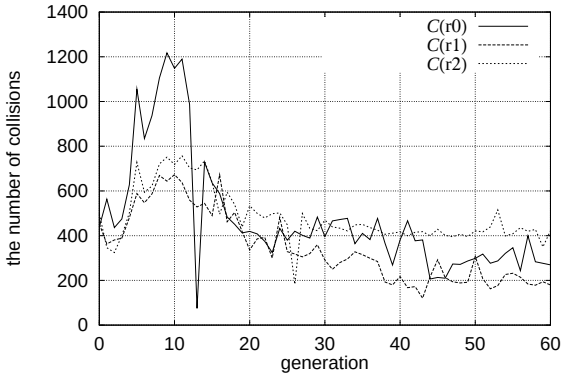
(b) the best fitness $f^f_{\min}(i)$



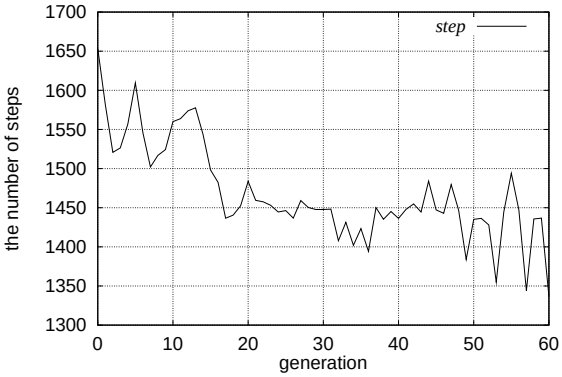
(c) the average number of scores $G(i)$ and $L(i)$



(d) the average number of kicking $K(i)$

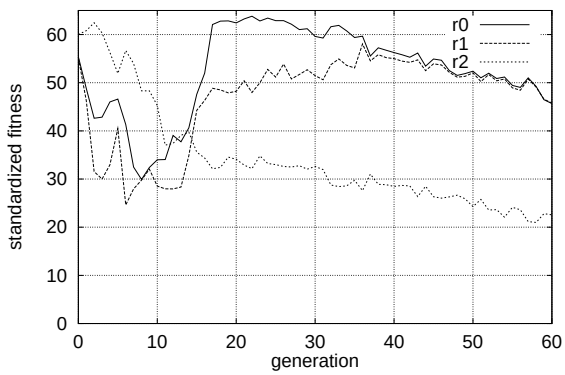


(e) the average number of collisions $C(i)$

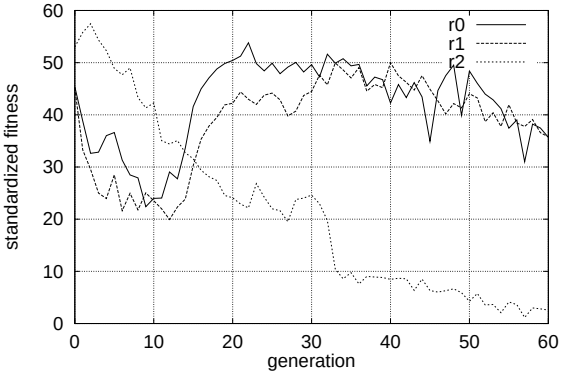


(f) the average number of steps *step*

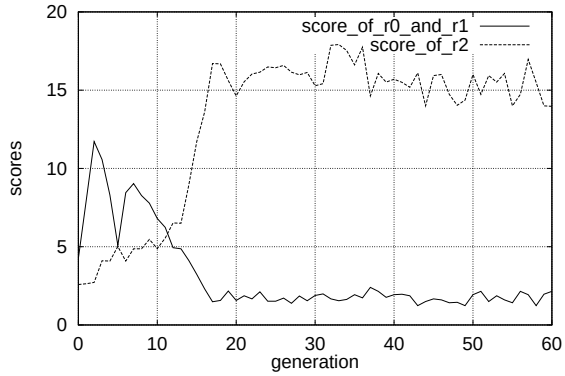
Fig.6.12 Experimental results in a case of three learners with fixed fitness function



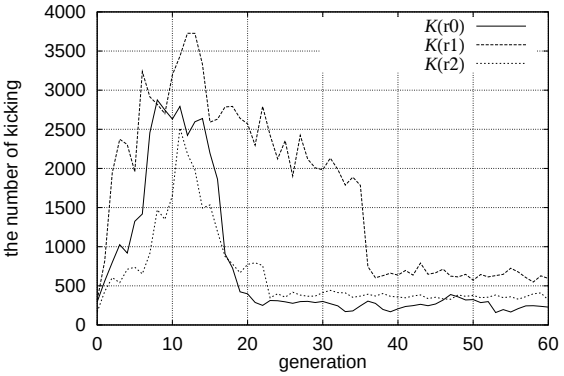
(a) the average fitness $f^v(i)$



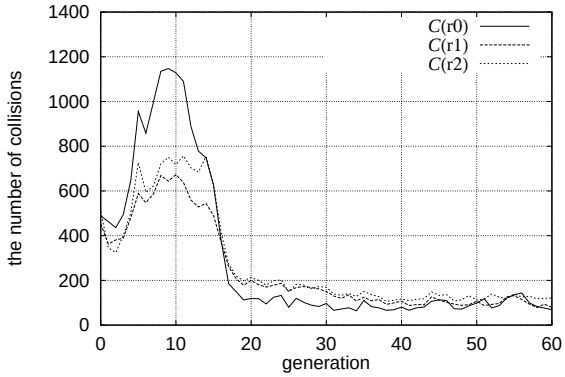
(b) the best fitness $f^f_{\min}(i)$



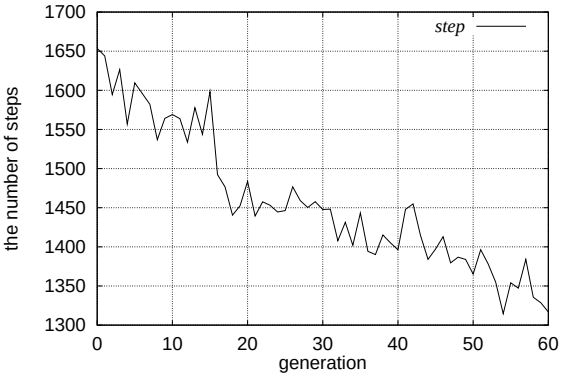
(c) the average number of scores $G(i)$ and $L(i)$



(d) the average number of kicking $K(i)$



(e) the average number of collisions $C(i)$



(f) the average number of steps *step*

Fig.6.13 Experimental results in a case of three learners with varying fitness function

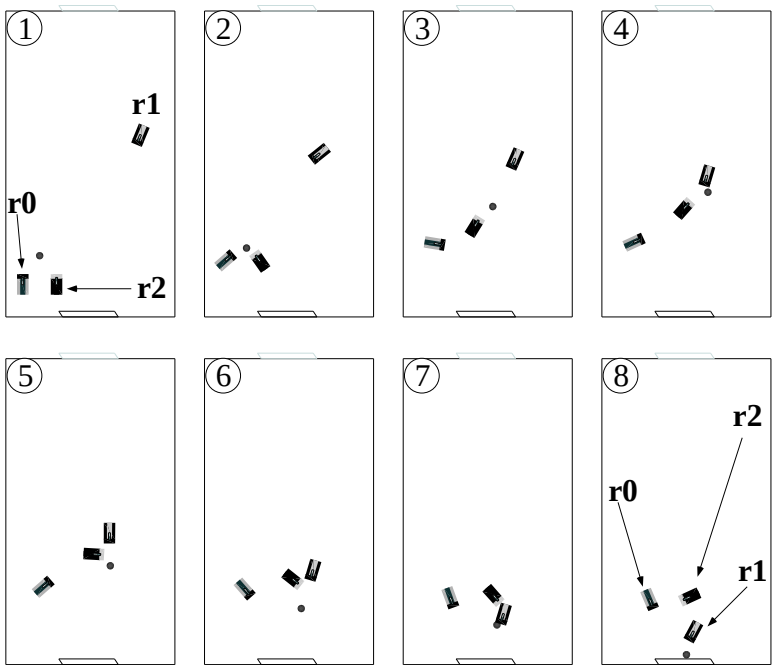


Fig.6.14 Two robots (**r0** and **r1**) succeed in shooting a ball into the goal against the defender (**r2**)

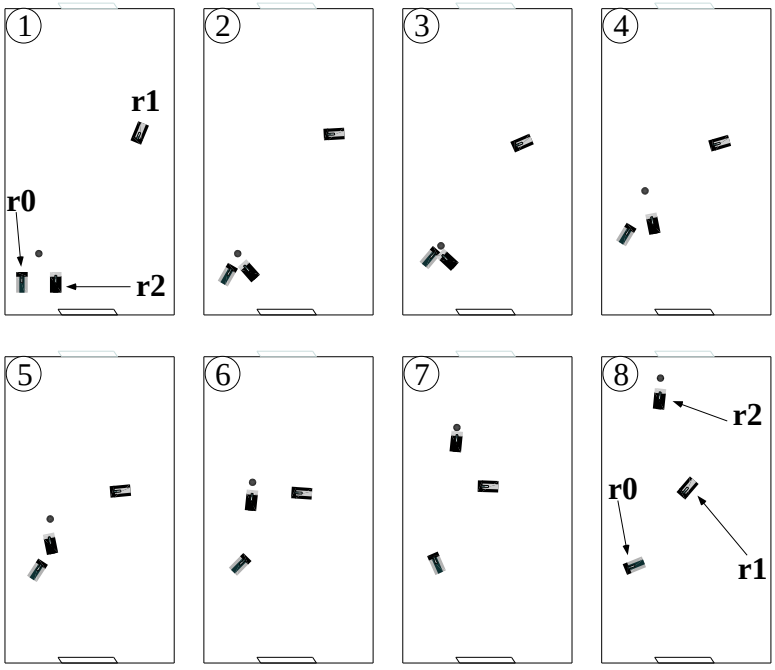


Fig.6.15 The defender (**r2**) succeeds in shoot a ball into the goal against the two robots (**r0** and **r1**)

6.5 Discussion and Future Works

This chapter showed how co-evolution technique could emerge not only competitive behaviors but also cooperative ones through a series of experiments in which two or three robots play a simplified soccer game. In order to co-evolve cooperative agents, it should be noted that robots must synchronize their evolutionary processes. Otherwise, there are many traps to local maxima (suboptimal strategies) as we can see in Section 6.4.1.

In a case of the most complicated situation (three agents and both cooperation and competition are involved), the task complexity should be equal to all agents so as to co-evolve cooperative and competitive agents simultaneously. This also suggests that the environment itself should co-evolve from simpler to more complicated situations to assist the development of desired skills of cooperations and competitions. Otherwise, co-evolution is prone to be settled into suboptimal strategies as shown in Section 6.4.3.

In the current system, the visual information processing is simplified by color region extraction. State quantization and terminal actions were fixed. Through the evolution process, the agent obtains its decision tree which tells how visual features are organized and connected to one terminal action. A different application of *GP* can be considered to extract motor outputs by a feature tree in which sensory inputs are combined and calculated. That is, evolutionary process can be used to express the motor outputs in terms of the sensor inputs. In this case, both motor and sensor spaces are continuous; therefore, it needs much more individuals and generations. Further, it seems difficult to extract meaningful modules from the final tree that can be re-usable in different contexts.

More systematic understanding is needed to make clear what are necessary and sufficient conditions to lead co-evolutionary processes to successful situations. Design issues of environments including agents, tasks, and fitness functions are our future work. Also, we are planning to implement real experiments to check the validity of the proposed method and the obtained behaviors.

Chapter 7

Conclusion

In this dissertation, we developed techniques for cooperative and/or competitive behavior acquisition in a multiagent environment. Multiagent systems are often required because of spatial or geographic distribution, or in situation where centralized information is not available or is not practical. Even when a distributed approach is not required, multiple agents may still provide an excellent way of scaling up to approximate solutions for very large problems by streamlining the search through the space of possible policies.

In order to demonstrate the power of multiagent reinforcement learning, we focused the research issues on the state representation and the reward function. Our work deals with the following three issues in reinforcement learning [41]

- incomplete and imperfect perception (Chapter 3),
- re-learning based on the previously acquired results (Chapter 4),
- coordination of multiple behaviors (Chapter 5),

in the context of multiagent learning.

In Chapter 3, we have proposed an idea of the *Local Prediction Model* (hereafter *LPM*) in order to apply reinforcement learning to the multiagent environments. In a dynamic environment including other robots, sensor outputs do not have any simple relationship to the learner's motion. Markovian assumption which is the necessary condition to apply reinforcement learning may be violated if the robots construct the state representation from only current observation. Therefore, the learning robot has to construct the complicated internal representation. When the real robots move in the real environment, all the physical objects including robots have the dynamics. Therefore, we regard the order as the a sort of criterion for state representation, that is we focus on the order of the state vector of the objects. The *LPM* estimates the local interaction between the learner and the other objects taking account of the tradeoff among the precision of prediction, the dimension of state vector, and the length of steps to predict.

In Chapter 4, we have shown the method of controlling the environmental complexity based on the *LPM*. If the representation become complicated, the learning time become

long. Therefore, we have to cope with the acceleration of the learning. However, existing work is based on the viewpoint of only learner. Therefore, we focus on the complexity of the interaction to accelerate the learning. In multiagent systems, human designer can control two issues to discuss the complexity of the interaction. One is that how the learner should represent other robots because sensor outputs have no simple relationship with the learner's action. The other is that how other robot should behave toward the learner because the goal achievement of the learner depends on the behavior of other robots. In case of the learner, the learner should change the internal representation of other robot. We regard the relative order of the state vector as a sort of the complexity of the interaction.

In Chapter 5, we have proposed a vector-valued reward function to cope with the multiple rewards. If we utilize the mixture of the several rewards, the estimation of the utility may become poor. This reward function is a straightforward extension of a scalar-valued reward function. We have shown the validity of the proposed method by performing a series of experiments in the context of robotic soccer.

In Chapter 6, we have shown how co-evolution technique could emerge not only competitive behaviors but also cooperative ones through a series of experiments in which two or three robots play a simplified soccer game. In order to co-evolve cooperative agents, it should be noted that robots must synchronize their evolutionary processes. Otherwise, there are many traps to local maxima (suboptimal strategies).

We do not intend to use explicit communication simply because in a game situation broadcasting might be overheard by opponent team members. A more serious reason is that a longer range research issue related to no use of explicit communication is to establish non-verbal communication based on observation and action, that is , eye-contact. This can be also related to emergence of language, that is, symbols emerged in one agent through learning or evolution processes might be shared by other agents to do some task together. Such symbols might be building blocks of the language of the agents. As a matter of fact, we might not be able to understand their language, but we may find the process of emergence of one language. However, it seems difficult to expect the current system to emerge such processes since both state variables and terminal actions are fixed. Therefore, we need to extend and modify it so that co-evolution can contribute to such processes.

Although there are some open problems discussed before , we suppose that the proposed method might be promising for the fundamental method.

Appendix A

Basics of Subspace State Space Identification

A.1 Problem Description

A number of algorithms to identify multi-input multi-output (MIMO) combined deterministic stochastic systems have been proposed. In contrast to 'classical' algorithms such as PEM (Prediction Error Method), the subspace system identification algorithms [29, 71] do not suffer from the problems caused by a priori parameterizations. Larimore's Canonical Variate Analysis (CVA) [36] is one of such algorithms, which uses canonical correlation analysis to construct a state estimator. Conceptually, CVA deals directly with the input and output time series.

Let $\boldsymbol{u}_t \in \Re^m$ and $\boldsymbol{y}_t \in \Re^q$ be the input and output generated by the unknown system

$$\begin{aligned} \boldsymbol{x}_{t+1} &= \boldsymbol{A}\boldsymbol{x}_t + \boldsymbol{B}\boldsymbol{u}_t + \boldsymbol{w}_t, \\ \boldsymbol{y}_t &= \boldsymbol{C}\boldsymbol{x}_t + \boldsymbol{D}\boldsymbol{u}_t + \boldsymbol{v}_t, \end{aligned} \tag{A.1}$$

with

$$E\left\{ \begin{bmatrix} \boldsymbol{w}_t \\ \boldsymbol{v}_t \end{bmatrix} \begin{bmatrix} \boldsymbol{w}_\tau^T & \boldsymbol{v}_\tau^T \end{bmatrix} \right\} = \begin{bmatrix} \boldsymbol{Q} & \boldsymbol{S} \\ \boldsymbol{S}^T & \boldsymbol{R} \end{bmatrix} \delta_{t\tau},$$

where $\boldsymbol{v}_t \in \Re^q$ and $\boldsymbol{w}_t \in \Re^n$ are unobserved, Gaussian-distributed, zero-mean, white noise vector sequences. $\boldsymbol{A} \in \Re^{n\times n}$ is called the (dynamical) system matrix. It describes the dynamics of the system (as completely characterized by its eigenvalues). $\boldsymbol{B} \in \Re^{n\times m}$ is the input matrix which represents the linear transformation by which the inputs influence the next state. $\boldsymbol{C} \in \Re^{q\times n}$ is the output matrix which describes how the internal state is transferred to the outside world in the measurements $\boldsymbol{y}$. The term with the matrix $\boldsymbol{D} \in \Re^{q\times m}$ is called the direct feedthrough term. In continuous time systems, this term is most often 0, which is not the case in discrete time systems due to the sampling. $\boldsymbol{Q} \in \Re^{n\times n}$, $\boldsymbol{S} \in \Re^{n\times q}$ and $\boldsymbol{R} \in \Re^{q\times q}$ are the covariance matrices of the noise sequences $\boldsymbol{w}$ and $\boldsymbol{v}$. $E\{\cdot\}$ denotes the expected value operator and $\delta_{t\tau}$ the Kronecker delta. Generally

speaking, CVA uses a estimated state vector $\hat{\boldsymbol{x}}$ which is a linear combination of the previous input-output sequences since it is difficult to determine the dimension of $\boldsymbol{x}$. Eq.(A.1) is transformed as follows:

$$\begin{bmatrix} \hat{\boldsymbol{x}}_{t+1} \\ \boldsymbol{y}_t \end{bmatrix} = \begin{bmatrix} \boldsymbol{T}^{-1}\boldsymbol{A}\boldsymbol{T} & \boldsymbol{T}^{-1}\boldsymbol{B} \\ \boldsymbol{C}\boldsymbol{T} & \boldsymbol{D} \end{bmatrix} \begin{bmatrix} \hat{\boldsymbol{x}}_t \\ \boldsymbol{u}_t \end{bmatrix} + \begin{bmatrix} \boldsymbol{T}^{-1}\boldsymbol{w}_t \\ \boldsymbol{v}_t, \end{bmatrix}, \tag{A.2}$$

and $\boldsymbol{x}_t = \boldsymbol{T}\hat{\boldsymbol{x}}_t$. Therefore, since we can regard $\boldsymbol{\mu}_t$ as a new state vector, we use $\hat{\boldsymbol{x}}$ as same as $\boldsymbol{x}$ hereafter. Strictly speaking, we require the pair $\{\boldsymbol{A},\ \boldsymbol{C}\}$ to be observable since only the modes that are observed can be identified. Furthermore, we require the pair $\{\boldsymbol{A}\ [\boldsymbol{B},\ \boldsymbol{Q}^{1/2}]\}$ to be controllable. This implies that all modes are excited by either the external input $\boldsymbol{u}$ and the process noise $\boldsymbol{w}$.

There are several subspace state space identification methods that assume the same conditions, and van Overschee and De Moor report the differences among them [72]. According to their results, CVA is more insensitive than other two methods to scaling of the inputs and/or outputs. They conclude that this is because only angles and normalized directions are considered in the CVA algorithm. Therefore, we choose CVA as an implementation of the *LPM*.

A.2 CVA Algorithm

Subspace Construction

At the beginning, we construct two subspaces $\boldsymbol{P}$ and $\boldsymbol{F}$ from the sequences of input (action) and output (perception). For $\{\boldsymbol{u}_t, \boldsymbol{y}_t\}$, $t = 1, \cdots N_{all}$, construct new vectors

$$\boldsymbol{p}_t = \begin{bmatrix} \boldsymbol{u}_{t-1} \\ \vdots \\ \boldsymbol{u}_{t-l} \\ \boldsymbol{y}_{t-1} \\ \vdots \\ \boldsymbol{y}_{t-l} \end{bmatrix}, \quad \boldsymbol{f}_t = \begin{bmatrix} \boldsymbol{y}_t \\ \boldsymbol{y}_{t+1} \\ \vdots \\ \boldsymbol{y}_{t+l-1} \end{bmatrix}. \tag{A.3}$$

Based on the vectors $\boldsymbol{p}$ and $\boldsymbol{f}$, we compute estimated covariance matrices $\hat{\boldsymbol{\Sigma}}_{pp}$, $\hat{\boldsymbol{\Sigma}}_{pf}$ and $\hat{\boldsymbol{\Sigma}}_{ff}$ as follows:

$$\hat{\boldsymbol{\Sigma}}_{pp} = \frac{1}{N'} \sum_{t=l+1}^{N_{all}-l+1} \tilde{\boldsymbol{p}}_t \tilde{\boldsymbol{p}}_t^T, \tag{A.4}$$

$$\hat{\boldsymbol{\Sigma}}_{ff} = \frac{1}{N'} \sum_{t=l+1}^{N_{all}-l+1} \tilde{\boldsymbol{f}}_t \tilde{\boldsymbol{f}}_t^T, \tag{A.5}$$

$$\hat{\boldsymbol{\Sigma}}_{pf} = \frac{1}{N'} \sum_{t=l+1}^{N_{all}-l+1} \tilde{\boldsymbol{p}}_t \tilde{\boldsymbol{f}}_t^T, \tag{A.6}$$

where $N' = N_{all} - 2l + 1$ and

$$\begin{aligned}\tilde{\boldsymbol{p}}_t &= \boldsymbol{p}_t - E\{\boldsymbol{p}_t\} = \boldsymbol{p}_t - \sum_{t=l+1}^{N_{all}-l+1} \boldsymbol{p}_t, \\ \tilde{\boldsymbol{f}}_t &= \boldsymbol{f}_t - E\{\boldsymbol{f}_t\} = \boldsymbol{f}_t - \sum_{t=l+1}^{N_{all}-l+1} \boldsymbol{f}_t,\end{aligned}$$

The rank of the subspace depends on the historical length l. Although the estimation is improved if l becomes longer and longer, it needs enough memory and time to maintain $\boldsymbol{P}$ and $\boldsymbol{F}$. As we can see, $\hat{\boldsymbol{\Sigma}}_{pp}$ and $\hat{\boldsymbol{\Sigma}}_{ff}$ are usually regular matrices if the observation vector $\boldsymbol{y}$ is appropriate.

Generalized Singular Value Decomposition

Next, the relation between calculated subspaces $\boldsymbol{P}$ and $\boldsymbol{F}$ should be obtained. This relation can be solved by the following the generalized singular value decomposition

$$\begin{bmatrix} \boldsymbol{U} & \mathbf{o} \\ \mathbf{o} & \boldsymbol{V} \end{bmatrix} \begin{bmatrix} \boldsymbol{\Sigma}_{pp} & \boldsymbol{\Sigma}_{pf} \\ \boldsymbol{\Sigma}_{fp} & \boldsymbol{\Sigma}_{ff} \end{bmatrix} \begin{bmatrix} \boldsymbol{U}^T & \mathbf{o} \\ \mathbf{o} & \boldsymbol{V}^T \end{bmatrix} = \begin{bmatrix} \boldsymbol{I}_{l(m+q)} & \boldsymbol{S}_{aux} \\ \boldsymbol{S}_{aux}^T & \boldsymbol{I}_{lq} \end{bmatrix}. \tag{A.7}$$

Actually, we have to compute following singular value decomposition

$$\hat{\boldsymbol{\Sigma}}_{pp}^{-1/2} \hat{\boldsymbol{\Sigma}}_{pf} \hat{\boldsymbol{\Sigma}}_{ff}^{-1/2} = \boldsymbol{U}_{aux} \boldsymbol{S}_{aux} \boldsymbol{V}_{aux}^T, \tag{A.8}$$

$$\boldsymbol{U}_{aux} \boldsymbol{U}_{aux}^T = \boldsymbol{I}_{l(m+q)}, \quad \boldsymbol{V}_{aux} \boldsymbol{V}_{aux}^T = \boldsymbol{I}_{kq}.$$

Generally speaking, in order to obtain the matrices $\hat{\boldsymbol{\Sigma}}_{pp}^{-1/2}$ and $\hat{\boldsymbol{\Sigma}}_{ff}^{-1/2}$, we have to solve the eigen value decomposition. Given the symmetric matrix $\boldsymbol{A} \in \Re^{n \times n}$, the eigen value decomposition of $\boldsymbol{A}$ is

$$\boldsymbol{A} = \boldsymbol{V} \boldsymbol{\Delta}^2 \boldsymbol{V}^T = \lambda_1 \boldsymbol{v}_1 \boldsymbol{v}_1^T + \cdots \lambda_n \boldsymbol{v}_n \boldsymbol{v}_n^T,$$

where $\boldsymbol{v}$ is an eigenvector of the matrix $\boldsymbol{A}$, and

$$\begin{aligned}\boldsymbol{\Delta}^2 &= \mathrm{diag}\begin{bmatrix} \lambda_1 & \lambda_2 & \cdots & \lambda_n \end{bmatrix}, \\ \boldsymbol{V} &= \begin{bmatrix} \boldsymbol{v}_1 & \boldsymbol{v}_2 & \cdots & \boldsymbol{v}_n \end{bmatrix},\end{aligned}$$

where λ is an eigen value of the matrix $\boldsymbol{A}$. After all, the matrix $\boldsymbol{A}^r$ can be calculated by $\boldsymbol{V} \boldsymbol{\Delta}^{2r} \boldsymbol{V}^T$. Based on this spectral decomposition, we can compute both $\hat{\boldsymbol{\Sigma}}_{pp}^{-1/2}$ and $\hat{\boldsymbol{\Sigma}}_{ff}^{-1/2}$.

After we solve Eq.(A.8), the n dimensional estimated vector $\boldsymbol{x}_t$ is defined as:

$$\boldsymbol{x}_t = [\boldsymbol{I}_n \ \ 0] \boldsymbol{M} \tilde{\boldsymbol{p}}_t, \quad t = l+1, \cdots, N_{all} - l + 1, \tag{A.9}$$

where the matrix $\boldsymbol{M}$ is calculated by

$$\boldsymbol{M} := \boldsymbol{U}_{aux}^T \hat{\boldsymbol{\Sigma}}_{pp}^{-1/2}. \tag{A.10}$$

Estimation of parameter matrices

After we define the state vector by Eq.(A.9), the parameter matrix $\boldsymbol{\Theta}$ can be estimated applying the least square method to Eq (A.2) as follows.

$$\hat{\boldsymbol{\Theta}} = \left(\frac{1}{N'} \sum_{t=l+1}^{N-k+1} \begin{bmatrix} \boldsymbol{x}_{t+1} \\ \boldsymbol{y}_t \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_t^T & \boldsymbol{u}_t^T \end{bmatrix} \right) \left(\frac{1}{N'} \sum_{t=l+1}^{N-k+1} \begin{bmatrix} \boldsymbol{x}_t \\ \boldsymbol{u}_t \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_t^T & \boldsymbol{u}_t^T \end{bmatrix} \right)^{-1}, \qquad \text{(A.11)}$$

where $\boldsymbol{\Theta}$ is the composed parameter matrix

$$\boldsymbol{\Theta} = \begin{bmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{C} & \boldsymbol{D} \end{bmatrix}.$$

This parameter matrix $\boldsymbol{\Theta}$ is used to predict the future observation and prediction error. We implement this algorithm using C++.

Appendix B

Basics of Reinforcement Learning

B.1 Standard Reinforcement Learning

B.1.1 Problem Description

Reinforcement learning is a kind of unsupervised learning, and improves its policy based on the delayed reinforcement without explicit state transition probabilities. In a typical reinforcement learning method, a robot and its environment are modeled by two synchronized finite state automatons interacting in discrete time cyclical processes. The robot senses the current state of the environment and selects an action. Based on the state and the action, the environment makes a transition to a new state and generates a reward that is passed back to the robot. Through these interactions, the robot learns a purposive behavior to achieve a given goal.

We assume that the robot can discriminate the set $\boldsymbol{X}$ of distinct world states, and can take the set $\boldsymbol{U}$ of actions on the world. The world is modeled as a Markov process, making stochastic transitions based on its current state and the action taken by the robot. Let $T(x, u, x')$ be the probability that the world will transit to the next state x' from the current state-action pair (x, u). For each state-action pair (x, u), the reward $r(x, u)$ is defined.

The general reinforcement learning problem is typically stated as finding a policy that maximizes discounted sum of the reward received over time. A policy f maps the state space $\boldsymbol{X}$ to the action space $\boldsymbol{U}$. This sum is so called the *return* and is defined as:

$$v_t(\boldsymbol{x}) = E\left\{\sum_{n=0}^{\infty} \gamma^n r_{t+n} \middle| \boldsymbol{x}_t = \boldsymbol{x}\right\}, \tag{B.1}$$

where $r(t)$ is the reward received at step t given that the agent started in state x and executed policy f. γ $(0 \leq \gamma < 1)$ is the discounting factor, which controls to what degree rewards in the distant future affect the total value of a policy and is slightly less than 1.

B.1.2 Q Learning

Watkins and Dayan [73] proposed Q learning which is a form of model-free reinforcement learning based on stochastic dynamic programming, and provides robots with the capability of learning to act optimally in a Markovian environment. A simple version of tabular Q learning algorithm is shown as follows:

1. Initialize $Q(x, u)$ to 0s for all combination of $\boldsymbol{X}$ and $\boldsymbol{U}$.
2. Perceive current state x.
3. Choose an action u according to the action value function.
4. Execute an action u in the environment. Let the next state be x' and immediate reward be r.
5. Update the action value function from x, u, x', and r,

$$Q_{t+1}(x, u) = Q_t(x, u) + \alpha \left[r + \gamma \max_{b \in \boldsymbol{U}} Q_t(x', b) - Q_t(x, u) \right], \tag{B.2}$$

   where α is a learning rate parameter and γ is a fixed discounting factor between 0 and 1.
6. Return to 2.

B.2 Coordination of Interfering Multiple Behaviors based on Q Learning

B.2.1 Background

Simple application of the reinforcement learning method to multiple robot tasks seems hard because of enormous amount of learning time. Several methods are proposed to cope with large scaled robot tasks. Singh [58] defined a composite task as sequentially concatenating multiple elemental tasks, and rewards are generated only when the system achieves a subtask in a prescribed order. Whitehead *et al.* [75] proposed a modular architecture to coordinate multiple behaviors. Subtasks are independent of each other, and therefore, their execution order can be arbitrary. The validity of these methods has been shown by computer simulations where the action and state spaces are too idealized and the task seems simple and straightforward.

Connel and Mahadevan [14] proposed a rapid task learning for real robots by decomposing the whole task (box-pushing) into subtasks (finding, pushing a box, and unwedging) independent of each other. However, decomposition and switching conditions

between subtasks are designed by the programmer. Gachet *et al.* [19] realized a coordinated behavior which is a linear combination of basic behaviors. However, the resultant behavior is not guaranteed as an optimal one.

These methods explained above assume that the state spaces of subtasks do not interfere with each other or they are completely independent of each other. Asada *et al.* [8] proposed a method for behavior coordination in a case that the state spaces interfere with each other. However, it takes still long time to acquire the coordinated behaviors because their method learns the whole state space.

B.2.2 Finding Inconsistent States among Interfering Multiple Behaviors

In order to overcome with the problems described above, the whole state space is classified into two categories based on the action values separately obtained by Q learning as follows [70] :

- **no more learning area** : the set of states at which one of the learned behaviors is directly applicable. The maximum number of **no more learning areas** is the number of behaviors to be coordinated. In this area, the agent utilize the learning results with no modification.
- **re-learning area** : the set of states at which it is necessary to learn again because of the competition of multiple behaviors.

Fig.B.1 shows the basic idea of coordination of multiple interfering behaviors based on reinforcement learning, where the number of the tasks n is two for the sake of reader's understanding. First, a new state space ${}^{c}\boldsymbol{X}$ is composed by direct product of all the state spaces ${}^{i}\boldsymbol{X}$ of the i-th behavior ($i = 1, 2, \cdots,$). We define the kernel state ${}^{c}x_{k,i}$ of i-th behavior in order to prepare for clustering the composite state space ${}^{c}\boldsymbol{X}$. We calculate the maximum action value function

$$ {}^{i}q_{max}({}^{i}x) = \max_{a' \in \boldsymbol{A}} {}^{i}Q({}^{i}x, a'). $$

for the state ${}^{i}x \in {}^{i}\boldsymbol{X}$. If the state ${}^{c}x$ satisfies

$$ {}^{c}x = \arg \max_{{}^{i}x \in {}^{i}\boldsymbol{X}} {}^{i}q_{max}({}^{i}x), \tag{B.3} $$

and for all j ($j \neq i$).

$$ {}^{c}x = \arg \min_{{}^{j}x \in {}^{j}\boldsymbol{X}} {}^{j}q_{max}({}^{j}x) \tag{B.4} $$

In these cases, we regard the state ${}^{c}x$ as the kernel state ${}^{c}x_{k,i}$ of the i-th behavior. We apply ISODATA clustering algorithm [9], which is an iterative and non-hierarchical clustering method, to classify these action values.

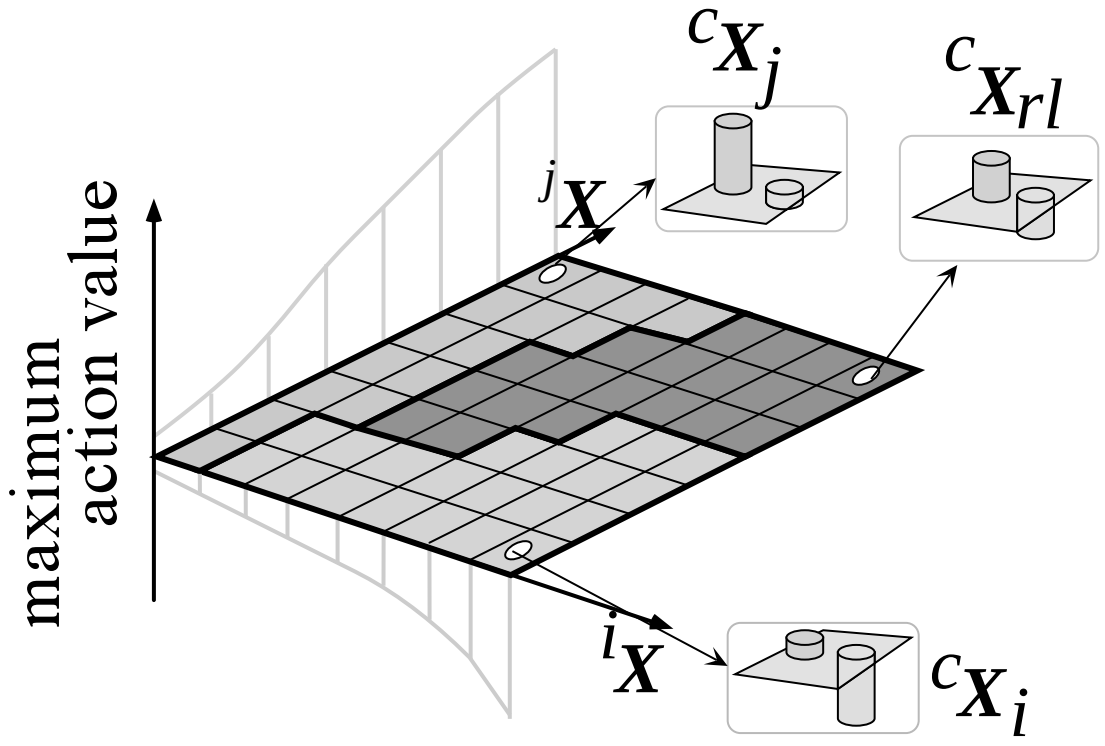


Fig.B.1 Basic idea for coordination of interfering multiple behaviors based on reinforcement learning

Then all states ${}^{c}x \in {}^{c}\boldsymbol{X}$ are classified according to the weighted distance between the non-kernel state and the kernel state. We utilize the weighted distance in the optimal action value space which is affected by the results of learning to classify the state. The distance between a composite state ${}^{c}x_j$ and the kernel state ${}^{c}x_k$ is calculated by

$$d_{j,k} = (\boldsymbol{q}({}^{c}x_j) - \boldsymbol{q}({}^{c}x_k))^T \boldsymbol{W} (\boldsymbol{q}({}^{c}x_j) - \boldsymbol{q}({}^{c}x_k)), \tag{B.5}$$

where $\boldsymbol{W}$ denotes the weighted matrix, and

$$\boldsymbol{q}({}^{c}x)^T = [{}^{1}q_{\max}({}^{1}x) \ \cdots \ {}^{n}q_{\max}({}^{n}x)].$$

Altogether, we summarize the classification algorithm as follows when the learning agent has n learned behaviors.

1. **Definition of the kernel states** : Based on Eq (B.3) and (B.4), determine the kernel state ${}^c x_{k,i}$ $(i = 1 \cdots n)$.
2. **Definition of initial areas** : Compute the weighted distance between n kernel states and other states. According to the distance, categorize all the states into one of the area.
3. **Rearrangement of areas** : Based on ISODATA algorithm, all the areas (no-more learning areas and re-learning areas).

All together, the composite state space ${}^c\boldsymbol{X}$ is classified into n no-more learning area ${}^c\boldsymbol{X}_i$, $i = 1 \cdots n$ and one re-learning area ${}^c\boldsymbol{X}_{rl}$.

B.2.3 Learning Rules

If both the current state x and the next state x' belong to the re-learning area, the normal Q learning algorithm can be applied. On the other hand both states x and x' belong to the no more learning area, it is not necessary to update the action value functions any more. The problem is the estimation of discounted sum of the reward to update the action value function, if the state x belong to the re-learning area while the state x' belong to the no more learning area. In general, the action values before and after coordination might not be consistent between different areas. Because, the action values before coordination are acquired independently by different subtasks, and therefore direct use of the action values simply brings to local maxima. Therefore, we have to adjust the action value function.

To overcome with this problem, we calculate $steps(x)$, the expected physical number of steps to goal, given that the process begins in the state x and follows the optimal policy thereafter. The action value functions are appropriately discounted using $steps({}^i s)$ as a discount factor. Eventually, we show the learning rules of this algorithm as follows:

1. if $x, x' \in {}^c\boldsymbol{X}_{rl}$: Apply normal update rule Eq.(B.2).
2. if $x \in {}^c\boldsymbol{X}_{rl}$, $x' \in {}^i\boldsymbol{X}$: Update ${}^cQ_{rl}(x, a)$ as follows.

$$ {}^cQ_{rl,t+1}(x, a) = {}^cQ_{rl,t}(x, a) + \alpha \left[r + \gamma {}^iV(x') - {}^cQ_{rl,t}(x, a) \right], \tag{B.6} $$

where ${}^iV(x')$ is value function of the no more learning are of i-th behavior. That is,

$$ {}^iV(x') = {}^c\gamma \max_{b \in \boldsymbol{A}} {}^iQ(x', b), \tag{B.7} $$

3. if $x \in {}^i\boldsymbol{X}$: Do not update.

Bibliography

[1] Akaike, H. A New Look on the Statistical Model Identification. *IEEE Transaction on Automatic Control*, 19:716–723, 1974.

[2] Angeline, P. J., and Pollack, J. B. Competitive Environments Evolve Better Solutions for Complex Tasks. In *Proc. of International Conference on Genetic Algorithm*, pages 264–270, 1993.

[3] Arai, S., Miyazaki, K., and Kobayashi, S. Generating Cooperative Behavior by Multi-Agent Reinforcement Learning. In *Proc. of the Sixth European Workshop on Learning Robots*, pages 111–120, 1997.

[4] Asada, M. An Agent and an Environment: A View of "Having Bodies" – A Case Study on Behavior Learning for Vision-Based Mobile Robot –. In *Proc. of 1996 IROS Workshop on Towards Real Autonomy*, pages 19–24, 1996.

[5] Asada, M., et al. Purposive Behavior Acquisition for a Real Robot by Vision-Based Reinforcement Learning. *Machine Learning*, 23:279–303, 1996.

[6] Asada, M., Noda, S., and Hosoda, K. Action-Based Sensor Space Categorization for Robot Learning. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1996.

[7] Asada, M., Uchibe, E., and Hosoda, K. Agents That Learn from Other Competitive Agents. In *Proc. of Machine Learning Conference Workshop on Agents That Learn from Other Agents*, 1995.

[8] Asada, M., Uchibe, E., Noda, S., Tawaratsumida, S., and Hosoda, K. Coordination of Multiple Behaviors Acquired By A Vision-Based Reinforcement Learning. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 2, pages 917–924, 1994.

[9] Ball, G. H., and Hall, D. J. ISODATA, a novel method of data analysis and pattern classification. *Stanford Research Institute, AD-699616*, 1965.

[10] Boyan, J. A., and Moore, A. W. Generalization in Reinforcement Learning : Safely Approximating the Value Function. In *Advances in Neural Information Processing Systems*. MIT Press, Cambridge, MA, 1995.

[11] Brooks, R. A. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2:14–23, 1986.

[12] Cliff, D., and Miller, G. F. Co-evolution of Pursuit and Evasion II : Simulation Methods and Results. In *Proc. of the Fourth International Conference on Simulation of Adaptive Behavior: From Animals to Animats 4*, pages 506–515, 1996.

[13] Clouse, J. A., and Utogoff, P. E. A Teaching Method for Reinforcement Learning. In *Proc. of the Nineth International Conference on Machine Learning*, pages 92–101, 1992.

[14] Connel, J. H., and Mahadevan, S. Rapid Task Learning for Real Robot. In *Robot Learning* [15], chapter 5, pages 105–140.

[15] Connel, J. H., and Mahadevan, S. *Robot Learning.* Kluwer Academic Publishers, 1993.

[16] Demiris, J., and Hayes, G. Imitative Learning Mechanisms in Robots and Humans. In Klingspor, V., editor, *Proc. of the Fifth European Workshop on Learning Robots*, Bari, Italy, 1996.

[17] Dorigo, M., and Colombetti, M. Robot Shaping: Developing Autonomous Agents through Learning. *Artificial Intelligence*, 71(2):321–370, 1994.

[18] Floreano, D., and Nolfi, S. Adaptive Behavior in Competing Co-Evolving Species. In *Proc. of the Fourth European Conference on Artificial Life*, pages 378–387, 1997.

[19] Gachet, D., Salichs, M. A., Moreno, L., and Pimentel, J. R. Learning Emergent Tasks for an Autonomous Mobile Robot. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 290–297, 1994.

[20] Goldberg, D. E. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison–Wesley, 1989.

[21] H. Kitano, ed. *RoboCup-97 : Robot Soccer World Cup I.* Springer Verlag, 1997.

[22] Held, R., and Hein, A. Movement-produced stimulation in the development of visually guided behaviors. *Journal of Comparative and Physiological Psycology*, 56:5:872–876, 1963.

[23] Horridge, G. A. The evolution of visual proceeding and the construction of seeing systems. In *Proc. of Royal Soc. London B230*, pages 279–292, 1987.

[24] Hosoda, K., and Asada, M. Versatile Visual Servoing without Knowledge of True Jacobian. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 186–193, 1994.

[25] Hu, J., and Wellman, M. P. Multiagent Reinforcement Learning : Theoretical Framework and an Algorithm. In *Proc. of the Fifteenth International Conference on Machine Learning*, 1998.

[26] Ishida, T., Katagiri, Y., and Kuwabara, K. *Distributed Artificial Intelligence*. Corona Publishing CO., LTD., Tokyo, Japan, 1996. (in Japanese).

[27] Kaelbling, L. P. *Learning in Embedded Systems*. MIT Press, 1993.

[28] Kanerva, P. Sparse distributed memory and related models. In Hassoun, M. H., editor, *Associative Neural Memories*, pages 50–76, New York, 1993. Oxford University Press.

[29] Katayama, T. *Introduction to System Identification*. Asakura Publishing, 1994. (in Japanese).

[30] Kimura, H., and Kobayashi, S. An Analysis of Actor/Critic Algorithms using Eligibility Traces: Reinforcement Learning with Imperfect Value Function. In *Proc. of the Fifteenth International Conference on Machine Learning*, 1998.

[31] Kohri, T., Matsubayashi, K., and Tokoro, M. An Adaptive Architecture for Modular Q-Learning. In *Fifteenth International Joint Conference on Artificial Intelligence*, pages 820–825. Morgan Kaufmann, 1997.

[32] Koza, J. R. *Genetic Programming I : On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

[33] Koza, J. R. *Genetic Programming II : Automatic Discovery of Reusable SubPrograms*. MIT Press, 1994.

[34] Kuniyoshi, Y. Behavior Matching by Observation for Multi-Robot Cooperation. In *International Symposium of Robotics Research*, 1995.

[35] Kuniyoshi, Y., Inaba, M., and Inoue, H. Learning by watching: Extracting reusable task knowledge from visual observation of human performance. *IEEE Transaction on Robotics and Automation*, 10(5), 1994.

[36] Larimore, W. E. Canonical Variate Analysis in Identification, Filtering, and Adaptive Control. In *Proc. 29th IEEE Conference on Decision and Control*, pages 596–604, Honolulu, Hawaii, December 1990.

[37] Lin, L.-J., and Mitchell, T. M. Reinforcement Learning With Hidden States. In *Proc. of the Second International Conference on Simulation of Adaptive Behavior: From Animals to Animats 2*, pages 271–280, 1992.

[38] Littman, M. L. Markov games as a framework for multi-agent reinforcement learning. In *Proc. of the Eleventh International Conference on Machine Learning*, pages 157–163, 1994.

[39] Luke, S., Hohn, C., Farris, J., Jackson, G., and Hendler, J. Co-Evolving Soccer Softbot Team Coordination with Genetic Programming. In *Proc. of the First RoboCup-97 Workshop at IJCAI'97*, pages 115–118, 1997.

[40] Maes, P., and Brooks, R. A. Learning to coordinate behaviors. In *Proc. of the Nineth National Conference on Artificial Intelligence*, pages 796–802, 1990.

[41] Mahadevan, S., and Kaelbling, L. P. The NSF Workshop on reinforcement learning: Summary and observations. *AI Magazine*, page 1996.

[42] Mataric, M. Learning to Behave Socially. In *Proc. of the Third International Conference on Simulation of Adaptive Behavior: From Animals to Animats 3*, pages 453–462, 1994.

[43] Mataric, M. Reward Functions for Accelerated Learning. In *Proc. of the Eleventh International Conference on Machine Learning*, pages 181–189, 1994.

[44] McCallum, A. K. Learning to Use Selective Attention and Short-Term Memory in Sequential Tasks. In *Proc. of the Fourth International Conference on Simulation of Adaptive Behavior: From Animals to Animats 4*, 1996.

[45] Michaud, F., and Mataric, M. Learning from History for Behavior-Based Mobile Robots in Non-Stationary Conditions. *Machine Learning*, 31(1–3):141–167, 1998.

[46] Minato, T., and Asada, M. Skill Acquisition and Self-Improvement for Environmental Change Adaptation of Mobile Robot. In *Proc. of the Fifth International Conference on Simulation of Adaptive Behavior: From Animals to Animats 5*, pages 360–365, 1998.

[47] Miyazaki, K., Yamamura, M., and Kobayashi, S. On the Rationality of Profit Sharing in Reinforcement Learning. In *Proc. of the 3rd International Conference on Fuzzy Logic, Neural Nets and Soft Computing*, pages 285–288, 1994.

[48] Moore, A. W., and Atkeson, C. G. The Parti-game Algorithm for Variable Resolution Reinforcement Learning in Multidimensional State-spaces. *Machine Learning*, 21:199–233, 1995.

[49] Nakamura, T., and Asada, M. Motion Sketch: Acquisition of Visual Motion Guided Behaviors. In *Fourteenth International Joint Conference on Artificial Intelligence*, pages 126–132. Morgan Kaufmann, 1995.

[50] Nottebohm, F. Laterality, seasons and space govern the learning of a motor skill. *Neuroscience*, 4:104–106, 1981.

[51] Ohko, T., Hiraki, K., and Anzai, Y. Reducing Communication Load on Contract Net by Case-Based Reasoning — Extension with Directed Contract and Forgetting. In *Proc. of the Second International Conference on Multi-Agent Systems*, pages 244–251, 1996.

[52] Omata, T. Learning with Assistance based on Evolutionary Computation. In *Proc. of the IEEE International Conference on Robotics and Automation*, pages 2180–2186, 1998.

[53] Ono, N., Ikeda, O., and Rahmani, A. T. Synthesis of Organization Behavior by Modular Q-learning Agents. In *Proc. of 1995 IEEE/Nagoya Univ. WWW'95 on Fuzzy Logic and Neural Networks / Evolutionary Computation*, pages 76–80, 1995.

[54] Quinlan, J. R. *C4.5: Programs for Machine Learning*. Morgan Kaufman, San Mateo, CA, 1993.

[55] Rao, R. P. N., and Fuentes, O. Hierarchical Learning of Navigational Behaviors in an Autonomous Robot using a Predictive Sparse Distributed Memory. *Machine Learning*, 31(1–3):87–113, 1998.

[56] Schwartz, B. *Psychology of Learning and Behavior: Third Edition*. W. W. Norton, NY, London, 1989.

[57] Sen, S., Sekaran, M., and Hale, J. Learning to coordinate without sharing information. In *Proc. of the Twelfth National Conference on Artificial Intelligence*, pages 426–431, 1994.

[58] Singh, S. P. Transfer of Learning by Composing Solution of Elemental Sequential Tasks. *Machine Learning*, 8:99–115, 1992.

[59] Smith, R. G., and Davis, R. Frameworks for Cooperation in Distributed Problem Solving. *IEEE Transaction on Systems, Man and Cybernetics*, 11(1):61–70, 1981.

[60] Steels, L. Structural coupling of cognitive memories through adaptive language games. In *Proc. of the Fifth International Conference on Simulation of Adaptive Behavior: From Animals to Animats 5*, pages 263–269, 1998.

[61] Steels, L., and Vogt, P. Grounding adaptive language games in robotic agents. In *Proc. of the Fourth European Conference on Artificial Life*, pages 474–482, 1997.

[62] Stone, P., and Veloso, M. Using Machine Learning in the Soccer Server. In *Proc. of IROS-96 Workshop on Robocup*, 1996.

[63] Stone, P., and Veloso, M. Team-Partitioned, Opaque-Transition Reinforcement Learning. In Asada, M., editor, *Proc. of the First RoboCup-97 Workshop at IC-MAS'98*, pages 221–235, 1998.

[64] Sugita, Y., and Tani, J. Emergence of Cooperative/Competitive Behavior in Two Robots' Games : Plans or Skills? In *SAB-98 Workshop 1 : Adaptive Behavior using Dynamic Reccurent Neural Nets*, 1998.

[65] Sutton, R. Generalization in Reinforcement Learning: Successful Examples Using Sparse Coarse Coding. In *Advances in Neural Information Processing Systems*, 1996.

[66] Sutton, R. S., and Barto, A. G. *Reinforcement Learning*. MIT Press/Bradford Books, March 1998.

[67] Suzuki, S., Kato, T., Asada, M., and Hosoda, K. Behavior Learning for a Mobile Robot with Omnidirectional Vision Enhanced by an Active Zoom Mechanism. In *Proc. of Intelligent Autonomous System 5(IAS-5)*, pages 242–249, 1998.

[68] Takahashi, Y., Asada, M., and Hosoda, K. Reasonable Performance in Less Learning Time by Real Robot Based on Incremental State Space Segmentation. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1996.

[69] Tani, J. Cognition of Robots from Dynamical Systems Perspective. In *Proc. of 1996 IROS Workshop on Towards Real Autonomy*, pages 51–59, 1996.

[70] Uchibe, E., Asada, M., and Hosoda, K. Behavior Coordination for a Mobile Robot Using Modular Reinforcement Learning. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1329–1336, 1996.

[71] van Overschee, P., and De Moor, B. A Unifying Theorem for Three Subspace System Identification Algorithms. *Automatica*, 31(12):1853–1864, 1995.

[72] van Overschee, P., and De Moor, B. *Subspace Identification for Linear Systems*. Kluwer Academic Publishers, 1996.

[73] Watkins, C. J. C. H., and Dayan, P. Technical note: Q-learning. *Machine Learning*, pages 279–292, 1992.

[74] Whitehead, S. D. Complexity and Coordination in Q-Learning. In *Proc. of the Eighth International Workshop on Machine Learning*, pages 363–367, Evanston, Illinois, 1991. Morgan Kaufmann.

[75] Whitehead, S. D., Karlsson, J., and Tenenberg, J. Learning Multiple Goal Behavior Via Task Decomposition And Dynamic Policy Merging. In Connel and Mahadevan [15], chapter 3.

[76] Yamaguchi, T., Miura, M., and Yachida, M. Multi-agent Reinforcement Learning with Adaptive Mimetism. In *5th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA-96)*, pages 288–294, 1996.

[77] Yang, B.-H., and Asada, H. Progressive Learning for Robotic Assembly: Learning Impedance with an Excitation Scheduling Method. In *Proc. of the IEEE International Conference on Robotics and Automation*, pages 2538–2544, 1995.

[78] Zlotkin, G., and Rosenschein, J. S. Cooperation and Conflict Resolution via Negotiation Among Autonomous Agents in Noncooperative Domains. *IEEE Transaction on Systems, Man and Cybernetics*, 21(6):1317–1324, 1991.