To Appear in Proc. of IEEE/RSJ/GI International Conference on Intelligent Robots and Systems, September 12-16, 1994, Munich, Germany

Coordination Of Multiple Behaviors Acquired By A Vision-Based Reinforcement Learning

Minoru Asada, Eiji Uchibe, Shoichi Noda, Sukoya Tawaratsumida, and Koh Hosoda
Dept. of Mech. Eng. for Computer-Controlled Machinery
Osaka University, 2-1, Yamadaoka, Suita, Osaka 565, Japan
asada@robotics.ccm.eng.osaka-u.ac.jp

Abstract

A method is proposed which accomplishes a whole task consisting of plural subtasks by coordinating multiple behaviors acquired by a vision-based reinforcement learning. First, individual behaviors which achieve the corresponding subtasks are independently acquired by Q-learning, a widely used reinforcement learning method. Each learned behavior can be represented by an action-value function in terms of state of the environment and robot action. Next, three kinds of coordinations of multiple behaviors are considered; simple summation of different action-value functions, switching action-value functions according to situations, and learning with previously obtained action-value functions as initial values of a new action-value function. A task of shooting a ball into the goal avoiding collisions with an enemy is examined. The task can be decomposed into a ball shooting subtask and a collision avoiding subtask. These subtasks should be accomplished simultaneously, but they are not independent of each other. Three kinds of coordinations are compared with each other by computer simulations and our on-going real experiments are explained.

1 Introduction

Reinforcement learning has recently 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 [1]. Fig.1 shows the basic model of robot-environment interaction, where a robot and environment are modeled by two synchronized finite state automatons interacting in a discrete time cyclical processes. The robot senses the current state of the environment and selects an action. Based on the state and 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.

Although the role of the reinforcement learning is very important to realize autonomous systems, the prominence of that role is largely determined by the extent to which it can be scaled to larger and complex robot learning tasks. However, the more complex and larger the robot task and the environment are, the much, much longer the learning rate is. Therefore, many theoretical works have argued the convergence

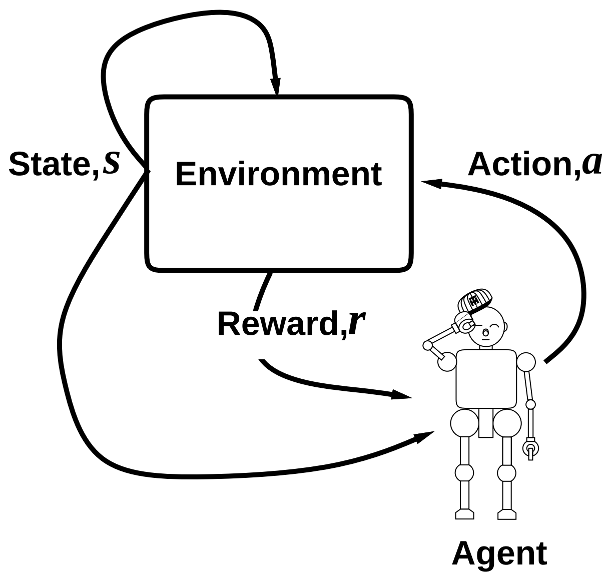


Figure 1: **The basic model of robot-environment interaction.**

time of the learning (ex. [2, 3]), and how to speed up it by using some heuristics such as modularity [4] and to extend these techniques from a single goal task to multiple ones [5]. Almost of them have only shown computer simulations, and only a few real robot applications are reported, which are simple and less dynamic [6, 7]. In order to make the role of the reinforcement learning evident in realizing autonomous agents, we need more applications in more dynamic and complex environments.

As one of these applications, we built a soccer robot [8] that tried to shoot a ball into the goal by applying the Q learning, a widely used reinforcement learning scheme [9]. The robot could learn a shooting behavior without world knowledge such as 3-D locations and sizes of the goal and ball in the field or the kinematics and dynamics of the robot itself. The information only the robot can capture is the image positions of the ball and/or goal which tell changes happened in the world caused by the robot actions.

In this paper, we attack more challenging problem of coordination of multiple behaviors which are concurrent with each other but not independent of each other. Such an example is to shoot a ball into a goal avoiding an enemy. The reason why challenging is twofold;

- from a viewpoint of building a real robot in a real situation, it is more dynamic and complicated environment than in [8], and
- from a viewpoint of robot learning, existing works have not demonstrated the ability to use previously learned knowledge to speed up the learning of a new policy [10].

To the best of our knowledge, only a few works related to the problem have been presented. Whitehead et al. [5] proposed a method which learns a multiple goal behavior by decomposing a task into subtasks and merging policies independently obtained in these subtasks (subgoals). In their scheme, multiple subgoals (getting food and water which are not placed at the same place) are parallel but independent of each other in the sense of subgoal-directed behavior. That is, the state space is consistent with all the subtasks and there is almost no interferences between them. The problem we attack here is to obtain a new behavior based on the previously learned behaviors which are concurrent but not always independent of each other. Therefore, we cannot apply their method to the problem. Further, they have shown only the computer simulation results for the multiple goal problem in which the environment is simple and less dynamic.

Connell and Mahadevan [7] described a system that decompose a single task into a series of subtasks, each of which is learned by an individual module. Although they showed the real experimental results for the task of pushing a box, subtasks of "finding a box," "pushing a box," and "getting unwedged" are independent of each other, therefore the whole task can be easily achieved according to situations which are straightforwardly discriminated by robot sensors.

The subsumption architecture [11] might be useful because one behavior can be set at the upper level than other behavior in order to subsume the other. However, this control law when to subsume the lower behavior seems difficult to be determined because it seriously depends on the situation. A typical example is a case that a robot tries to shoot a ball by attacking an enemy (this means a collision with an enemy).

In this paper, we propose a method which obtains a coordinated behavior consisting of different behaviors previously learned. The difficulty of the problem the existing works have not faced with seems to coordinate different behaviors that are concurrent and interfered with each other, and therefore action selection might be conflict in the dynamic and complicated situations. We consider three kinds of coordinations: simple summation of different action value functions, switching action value functions according to situations, and learning a new behavior given the previously learned policies. We discuss the performance of these methods and the differences between them.

The remainder of this article is structured as follows: In the next section, we give a brief overview of the Q learning and three kinds of coordinations of multiple behaviors. We then explain the task, and how to construct state and action spaces in our method. Next, we show the experiments with computer simulations and on-going real robot system. Finally, we give discussions.

2 Basics of Reinforcement Learning

2.1 Q-learning

Before getting into the details of our system, we briefly review the basics of the Q-learning. For more through treatment, see [12]. We follow the explanation of the Q learning by Kaelbling [13].

We assume that the robot can discriminate the set $\boldsymbol{S}$ of distinct world states, and can take the set $\boldsymbol{A}$ 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 $T(s, a, s')$ be the probability that the world will transit to the next state s' from the current state-action pair (s, a). For each state-action pair (s, a), the *reward* $r(s, a)$ is defined.

The general reinforcement learning problem is typically stated as finding a policy that maximizes discounted sum of the reward received over time. A policy f is mapping from $\boldsymbol{S}$ to $\boldsymbol{A}$. This sum called the *return* and is defined as:

$$\sum_{n=0}^{\infty} \gamma^n r_{t+n}, \tag{1}$$

where r_t is the reward received at step t given that the agent started in state s and executed policy f. γ is the discounting factor, it controls to what degree rewards in the distant future affect the total value of a policy and is just slightly less than 1.

Given definitions of the transition probabilities and the reward distribution, we can solve the optimal policy, using methods from dynamic programming [14]. A more interesting case occurs when we wish to simultaneously learn the dynamics of the world and construct the policy. Watkin's Q-learning algorithm gives us an elegant method for doing this.

Let $Q^*(s, a)$ be the expected return or *action-value function* for taking action a in a situation s and continuing thereafter with the optimal policy. It can be recursively defined as:

$$Q^*(s, a) = r(s, a) + \gamma \sum_{s' \in S} T(s, a, s') \max_{a' \in A} Q^*(s', a'). \tag{2}$$

Because we do not know T and r initially, we construct incremental estimates of the Q values on line. Starting with $Q(s, a)$ at any value (usually 0), every time an action is taken, update the Q value as follows:

$$Q(s, a) \Leftarrow (1 - \alpha) Q(s, a) + \alpha (r(s, a) + \gamma \max_{a' \in A} Q(s', a')). \tag{3}$$

where r is the actual reward value received for taking action a in a situation s, s' is the next state, and α is a leaning rate (between 0 and 1). The following is a simple version of the 1-step Q-learning algorithm we used here.

Initialization: $Q \quad \leftarrow$ a set of initial values for the action-value function (e.g., all zeros).
Repeat forever:

1. $s \leftarrow$ the current state

2. Select an action a that is usually consistent with the policy f but occasionally an alternate.

3. Execute action a, and let s' and r be the next state and the reward received, respectively.

4. Update $Q(s, a)$:

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha(r + \gamma \max_{a' \in A} Q(s', a')). \tag{4}$$

5. Update the policy f:

$$f(s) \leftarrow a \quad \text{such} \quad \text{that} \quad Q(s,a) = \max_{b \in \boldsymbol{A}} Q(s,b) \tag{5}$$

To speed up the learning time, we generate actions probabilistically based on Q values using a Boltzmann distribution. Given a situation s, we choose an action a with probability:

$$\frac{e^{Q(a,s)/T}}{\sum_{a \in \boldsymbol{A}} e^{Q(a,s)/T}} \tag{6}$$

This serves to make actions whose values are much better than the others be chosen with much greater likelihood. The *temperature* parameter T controls the amount of exploration (the degree to which actions other than the one with the best Q value are taken).

2.2 Learning a reflexive behavior

The Q-learning method can obtain not only goal-directed behaviors but also reflexive ones as well by slightly changing some parameters and updating equations. Unlike the goal-directed behaviors to find the path from the current state to the goal state, reflexive behaviors are reactive, and therefore, the discounting factor γ^r should be much smaller so that the action-value for the distant future action cannot be affected.

A typical example of such behaviors is "collision avoidance" which has another different property from that of goal-directed behaviors. That is, any action can be allowed to be taken unless it causes collisions with other objects (agents). In order to learn such a behavior, the negative reward should be assigned for the state-action pair which causes a collision with a moving obstacle, and such actions should be learned by using the following update equations instead of eqns.(4,5):

$$\begin{aligned} Q^r(s,a) &\Leftarrow (1-\alpha)Q^r(s,a) + \\ &\quad \alpha(r + \gamma^a \min_{a' \in A} Q^r(s',a')). \qquad (7) \\ f^r(s) &\leftarrow a \quad \text{such} \quad \text{that} \qquad (8) \\ &\quad Q^r(s,a) = \min_{b \in \boldsymbol{A}} Q^r(s,b). \qquad (9) \end{aligned}$$

After learning, the decision of action selection is done based on eqn.(5). That is, the agent tries to make collisions with other objects during the learning process, and it does not take such actions after the learning.

3 Coordinatin of Multiple Behaviors

We consider three kinds of coordinations in which the previously learned behaviors are combined; simple summation of different action value functions, switching action value functions according to situations, and learning given the learned policies as *a priori* knowledge. The state spaces $\boldsymbol{S}^c$ for the coordinated behavior in these coordinations are a little bit different from each other according to their methods. To simplify the following explanatins, let us consider to combine a goal-directed behavior $(Q^g(s^g, a))$ and a reflexive behavior $(Q^r(s^r, a))$ into a new one.

Basically, a state $s^c \in \boldsymbol{S}^c$ can be defined as a combined state of $\boldsymbol{S}^g$ and $\boldsymbol{S}^r$. We denote this combination as $\boldsymbol{S}^g \times \boldsymbol{S}^r$ or $(\boldsymbol{S}^g, \boldsymbol{S}^r)$. The number of $\boldsymbol{S}^c$ is theoretically a product of numbers of states of $\boldsymbol{S}^g$ and $\boldsymbol{S}^r$.

(a) Simple summation of different action value functions

The action value function of simple summation $Q^c_{ss}(s^c, a)$ for the coordinated behavior is given by;

$$Q^c_{ss}(s^c, a) = \max_{a \in \boldsymbol{A}} (Q^g((s^g, *), a) + Q^r((*, s^a), a)) \tag{10}$$

where $Q^g((s^g, *), a)$ and $Q^a((*, s^a), a)$ denote the extended action value functions for the goal-directed and reflexive behaviors in the new state space, respectively. $*$ means any states, therefore each of these functions considers only the original states and ignores the states of other behaviors. In this scheme, the selected action sometimes might not make any sense for both behaviors because the simple summation cannot consider combined new situations.

(b) Switching action value functions

The switching action value function $Q^c_{sw}(s^c, a)$ for the coordinated behavior is given by the following equation depending on a situation.

$$Q^c_{sw}(s^c, a) = \begin{cases} Q^r(s^r, a), & \text{in some situations} \\ Q^g(s^g, a), & \text{otherwise} \end{cases} \tag{11}$$

It seems hard to appropriately determine the situations to switch the functions $Q^g(s^g, a)$ and $Q^r(s^r, a)$. Therefore, we need a carefully designed decision rule to switch the policies. The following method provides us with this rule by learning a new policy coping with new situations.

(c) Learning a new behavior

In the above methods, the previously learned action value functions are simply summed or switched. Therefore these methods ignore some situations inconsistent with the state spaces $\boldsymbol{S}^g$ or $\boldsymbol{S}^r$. Eventually, an action suitable for these situations has never been learned. To cope with these new situations, the robot needs to learn a new behavior by using the previously learned behaviors. The method is as follows;

1. Construct a new state space $\boldsymbol{S}^c$:
   - (a) construct the directly combined state space $\boldsymbol{S}^g \times \boldsymbol{S}^r$.
   - (b) find such states that are inconsistent with $\boldsymbol{S}^g$ or $\boldsymbol{S}^r$.
   - (c) resolve the inconsistent states by adding new substates $s^c_{sub} \in \boldsymbol{S}^c$.
2. Learn a new behavior in the new state space $\boldsymbol{S}^c$:
   - (a) use the values of the action value function Q^c_{ss} as the initial values of Q^c_{rl} for both the normal states s^c and the new substates s^c_{sub}. For the new substates, we use the original value of $Q^c_{ss}(s^c, a)$ before generating these new states. That is,

$$\begin{aligned} Q^c_{rl}(s^c, a) &= Q^c_{ss}(s^c, a) \\ Q^c_{rl}(s^c_{sub}, a) &= \text{original value of } Q^c_{ss}(s^c, a) \end{aligned} \tag{12}$$

   - (b) control the temperature parameter T in eqn(6) for the action selection in such a way that low temperature (conservative) is used around the normal states s^c and high temperature (random) around the new substates s^c_{sub} in order to reduce the learning rate.

4 The Task and Assumptions

As mentioned in Introduction, complicated tasks in real world are not always decomposed into perfectly indepedent subtasks. As one of such tasks, we consider a task of a soccer robot which tries to shoot a ball into the goal avoiding an enemy as shown in **Fig.2**, where a robot and an enemy are indicated by a dark and bright rectangles, respectively. The problem we are attacking here is to develop a method which automatically acquires strategies how to do this. In [8], the robot has learned how to shoot a ball into the goal in the case of no enemy. Here, the robot tries to shoot a ball avoiding the collisions with an enemy. Except this point, the environment is the same as one in [8]. That is, the environment consists of a ball the robot can kick and a goal fixed on the ground.

If we know the exact three-dimensional parameters of the environment, kinematics/dynamics of the robot and the enemy, and sensing parameters such as internal and external camera parameters, we might be able to develop several methods to control it to shoot a ball into the goal avoiding the enemy. This is not our intention. We intend to start with only the visual information, that is, the image positions of the ball, the goal, and the enemy. That's all the robot captures from the environment. In order for the robot to take an action against the environment, it can select one action from several candidates. Note that the robot does not even know any physical meanings for them. The effects of an action against the environment can be informed to the robot only through the visual information. To enable to do that, the robot has to track the ball, the goal and/or the enemy in the image, continuously. A simple application of the Q-learning to the task here is not practical because the number of states drastically increases by a factor of more than ten compared with the number of states in our previous work, which means that non realistic number of trials are needed since the learning rate can be said exponential in the size of the state space [3].

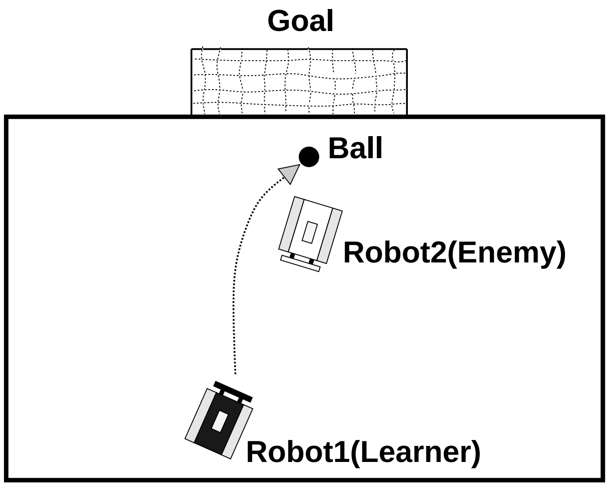


Figure 2: **The task is to shoot a ball into the goal avoiding an enemy.**

Here, we apply the method of coordination of multiple behavios to the task; shooting a ball into a goal avoiding collisions with an enemy (moving obstacle). The former has been learned in [8] as a goal-directed behavior. The latter which seems difficult in the sense of existing works (ex.[15]) is also acquired by Q-learning as a reflexive behavior. These two behaviors are combined into a new behabior by the three kinds of coordinations described in the provious section.

5 Constructing State and Action S-paces

In order to apply the Q-learning scheme to each of two subtasks, we define a number of sets and parameters for each of them. The existing applications of the reinforcement learning have constructed the state and action spaces in such a way that each action causes the state transition (ex. one action is forward, backward, left, or right, and states are encoded by the locations (coordinates) of the agent) in order to make the quantization problem (the structural credit assignment problem) easy. This makes a gap between the computer simulations and real robot systems. Each space should reflect the corresponding physical space in which a state or an action can be perceived or taken. Then, we construct these spaces considering the sensor resolution and control parameter resolution for the actuator as follows.

5.1 Preparations for the first task [8]

The first task to simply shoot a ball into the goal has been learned by using the following sets.

- a state set $\boldsymbol{S}^g$: only the information the robot can obtain about the environment is the image supposed to be capturing the ball and/or the goal. The ball image is quantized into 9 sub-states, combinations of three positions (left, center, and right) and three sizes (large (near), medium, and small (far)). The goal image has 27 sub-states, combinations of three parameters each of which is quantized three parts. Each sub-state corresponds to one posture of the robot toward the goal, that is, position and orientation of the robot in the field. In addition to these 243 (27×9) states, we add other states such as these cases in which only the ball or the goal is captured in the image.

  After some simulations, we realized that as long as the robot is capturing the ball and the goal positions in the image, it succeeds in shooting a ball. However, once it lost the ball, it randomly moves because it does not know to which direction it should move to find the ball. This causes because the ball-lost state is just one, therefore it cannot discriminate to which direction the ball is lost. Then, we separate the ball-lost state into two states; the ball-lost-into-right and the ball-lost-into-left states. Also, we set up goal-lost-into-right and goal-lost-into-left states, too. This improved the robot behavior much better. Eventually, we have 319 states in the set $\boldsymbol{S}^g$.

- an action set $\boldsymbol{A}$: In a real system, the robot moves around the field by a PWS (Power Wheeled Steering) system with two independent motors. Since we can send the motor control command to each of two motors independently, we quantized the action set in terms of two motor commands ω_l and ω_r, each of which has 3 sub-actions (forward, stop, and backward motions, respectively). Totally, we have 9 actions in the action set $\boldsymbol{A}$.

- a reward: we assign a reward value 3 when the ball was entered into the goal or 0 otherwise. This makes the learning very time-consuming. Although adopting a reward function in terms of distance to the goal state makes the learning time much shorter, it seems difficult to avoid the local maxima of the action-value function Q^g.

- a discounting factor γ^g is used to control to what degree rewards in the distant future affect the total value of a policy. In our case, we set the value a slightly less than 1 ($\gamma^g = 0.8$).

5.2 Preparations for the second task

The second subtask is to simply avoid a moving obstacle. The action set is the same as in the first one, but the number of states is much smaller because the state space consists of the image of only the moving obstacle, which is quantized by the same manner as for the ball image in the first task. That is, combinations of the position (left, center, and right) and the size (small, medium, and large) are used in the state space $\boldsymbol{S}^r$.

Since collision avoiding behavior is a reflexive one, we apply the method described in 2.2. In order to reduce the effects on the action-value function for the distant future actions, we set the discounting factor $\gamma^r = 0.1$. The robot tries to make collisions with a moving obstacle during the learning process, and to take any actions except collision after the learning process. To realize such a behavior, we set the negative reward (-1) for the state-action pair which causes a collision with a moving obstacle and use the update eqns.(7,9).

Another important issue is the enemy's behavior which tries to keep the ball outside the goal. If the enemy has learned the professional techniques to keep the goal, the robot might not be able to learn how to shoot a ball into the goal anymore because of almost no goals it achieves. From a viewpoint of teaching, the enemy's behavior should be idle in part so that the robot can succeed in shooting a ball into the goal. Then, we set the enemy's behavior in such a way that it randomly moves with probability of 50% and tends to chase after the robot in order to interfere its shooting behavior with probability of 50%.

6 Coordination of Learned Behaviors

We consider three kinds of coordinations in which the previously learned behaviors are combined. Since the numbers of $\boldsymbol{S}^g$ and $\boldsymbol{S}^r$ are 319 and 11 respectively, the number of $\boldsymbol{S}^c$ is basically 3509.

(a) Simple summation of two action value functions

By using eqn.(10), we decide an action for the coordinated behavior. In this scheme, the selected action sometimes might not make any sense for both behaviors because the simple sum cannot consider combined new situations.

(b) Switching action value functions

By using eqn.(11), we decide an action for the coordinated behavior. It seems hard to appropriately determine the situations to switch the functions $Q^g(s^g, a)$ and $Q^r(s^r, a)$. Simple situations we tried are the cases where only an enemy can be seen or where an enemy can be seen. In the former, the robot does not care about collisions with the enemy when the ball or the goal can be observed, while in the latter the robot tries to avoid the enemy even if it is likely able to shoot a ball into the goal. Therefore, we need a carefully designed decision rule to switch the policies.

(c) Learning a new behavior

In the above methods, the previously learned action value functions are simply summed or switched.

Therefore these methods ignore some situations inconsistent with the state spaces $\boldsymbol{S}^g$ or $\boldsymbol{S}^r$. Eventually, an action suitable for these situations has never been learned. To cope with these new situations, the robot needs to learn a new behavior by using the previously learned behaviors (the method is described in 3**(c)**).

A typical example is the case where a ball and the enemy are located at the same area and the ball is occluded by the enemy from the viewpoint of the robot. In this case, the robot cannot observe the ball, and therefore the corresponding state $s^g \in \boldsymbol{S}^g$ might be the state of "ball-lost," but it is not correct. Of course, if both the ball and the enemy can be observed, this situation can be considered consistent. This problem is resolved by adding new substates $s^c_{sub} \in \boldsymbol{S}^c$. In the above example, a new situation "occluded" is added, and the corresponding new substates are generated.

The learning scheme is applied to both normal states and newly generated ones with different temperature parameters T in eqn(6) for the action selection in such a way that low temperature (conservative) is used around the normal states s^c and high temperature (random) around the new substates s^c_{sub} in order to reduce the learning time.

7 Experiments

The experiment consists of two phases: first, learning the optimal policy f through the computer simulation, then apply the learned policy to a real situation. The merit of the computer simulation is not only to check the validity of the algorithm but also to save the running cost of the real robot during the learning process. However, still real experiments is necessary because the computer simulation cannot completely simulate the real world [10]. We have done the real experiments for the first task [8], that is, the robot learned how to shoot a ball into the goal without any enemy. Now, we are developing the real experiments for the coordinated behavior. Therefore we show the simulation results of the coordination of multiple behaviors by the simple summation, the switching, and the learning, and as a real system, we show the system configuration and the image processing results. At the conference, we will be able to presents the whole experimental results.

7.1 Simulation

We performed the computer simulation with the following specifications (the unit is an arbitrary-scaled length). The field is a square of which side length is 200. The goal post is located at the center of the top line of the square (see Fig.2) and its height and width are 10 and 50, respectively. The robot is 16 wide and 20 long and kicks a ball of which diameter is 6. The camera is horizontally mounted on the robot (no tilt), and its visual angle is 30 degrees. These and other parameters such as friction between the floor and the crawler and bounding factor between the robot and the ball are chosen to simulate the real world (for more details, see [8]).

In addition to three kinds of coordination methods, we show the performance data by only using the policy Q^g which completely ignores the existence of the enemy. Table 1 shows the simulation result where the rate of shooting per trial, the average of collision with the enemy, and the average steps needed to get a shoot. In the case of only using Q^g, the robot tries to shoot a ball ignoring the enemy, and therefore it collides with the enemy many times and needs much more steps to get a shoot although the rate is as good as the learning method. The simple sum seems better in collision because Q^a becomes dominant when the enemy approaches to it. However, it sometimes settles at the local maxima near the goal where Q^g and Q^a are balanced, and therefore the shooting rate is the worst. The switching condition we set is to use Q^g unless only the enemy can be observed very largely. The robot got more shoots than the simple sum because it can avoid the local maxima. However, when it uses Q^a, many actions not related to shooting behavior are chosen, and therefore it takes longest time step to get a shoot as a result. The learning method is the best in shooting rate, collision avoidance, and speed of shooting per trial.

Table 1: Simulation result

combination method	rate of shooting(%)	average of collisions/steps	average of steps
only Q^g	46.7	0.0232	286.9
simple sum	33.2	0.0129	231.2
switching	39.2	0.0102	414.4
learning	46.7	0.0042	128.3

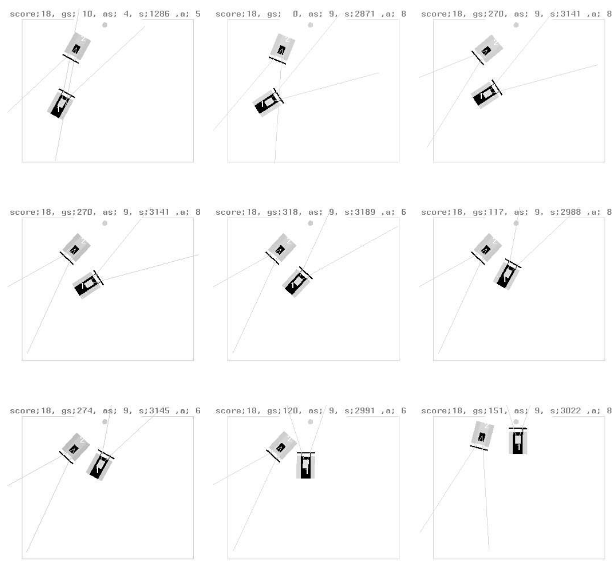

score;18, gs; 10, as; 4, s;1286 ,a; 5
score;18, gs; 0, as; 9, s;2871 ,a; 8
score;18, gs;270, as; 9, s;3141 ,a; 8
score;18, gs;270, as; 9, s;3141 ,a; 8
score;18, gs;318, as; 9, s;3189 ,a; 6
score;18, gs;117, as; 9, s;2988 ,a; 8
score;18, gs;274, as; 9, s;3145 ,a; 6
score;18, gs;120, as; 9, s;2991 ,a; 6
score;18, gs;151, as; 9, s;3022 ,a; 8
Figure 3: **A shooting behavior of the learning method**

Fig.3 shows a sequence of shooting behavior by the learning method. In these figures, the robot and the enemy are numbered 1 and 2, and colored in black and gray, respectively. The lines emerged from them shows their visual angles. The enemy tries to chase after the robot with the probability of 50% as long as it can see the robot. Otherwise, it randomly moves.

7.2 Real System

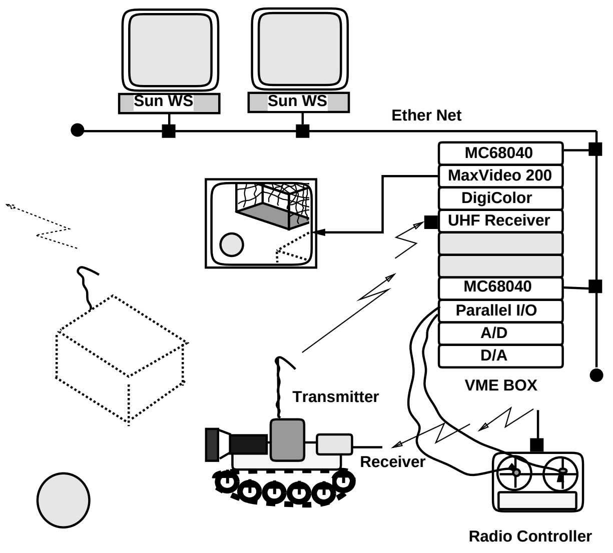


Figure 4: **A configuration of the real system.**

Fig.4 shows a configuration of the real mobile robot system some parts of which we have built. 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 enemy in red, blue, and yellow, respectively. We have constructed the radio control system of the robot, following the remote-brain project by Inaba et al. [16], but not yet for the enemy. The image processing and the vehicle control system are operated by VxWorks OS on MC68040 CPU which are connected with host Sun wrokstations via Ether net. Fig.5 shows a picture of the real robot with a TV camera (Sony handy-cam TR-3) and video transmitter.

Fig.6 shows the result of the image processing where a ball (front left), a goal (center-back), and an enemy (right) are detected and their positions are calculated in real time (1/30 seconds). **Fig.7** shows a sequence of images where the robot achieved the goal avoiding an enemy that is currently static.

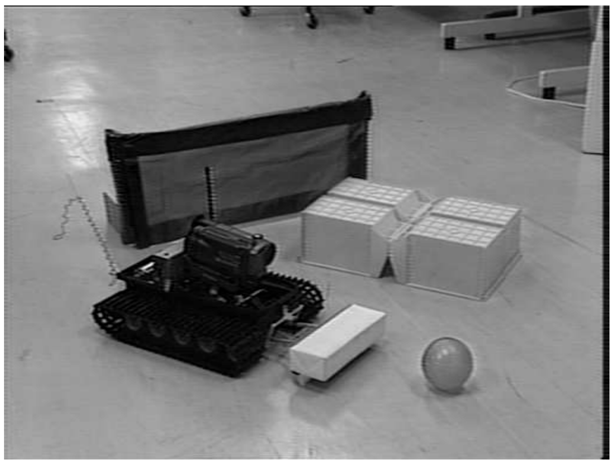
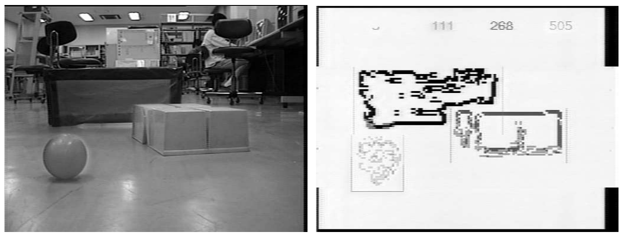


Figure 5: **A picture of the radio-controlled vehicle near the goal with a ball and an enemy (a group of four boxes).**



(a) input image (b) detected image

Figure 6: **Detection of a ball (front left), a goal (center back) and an enemy (right).**

8 Discussion and Future Works

We have proposed a method which acquires a new behavior by coordinating behaviors previously learned. Three kinds of coordinations are considered: simple summation of action value functions, switching policies according to situations, and learning a new policy. Although it is time-consuming, the learning method to obtain a new policy was the best one because the simple sum and the switchg methods do not learn anymore to cope with new situations. In the simulation experiments, the enemy's moved towards the robot with the probability of 50%, but it can behave much better to keep the goal. In order to obtain more proficient behavior, the robot should have the capability of learning a new policy every time.

About the real robot experiments, we have to finish it first. Now, the enemy's behavior is radio-controlled by human operator. We need another set of remote brain which has a realtime vision system and control system operated by the host CPU.

The future works in long term includes from the competition between single agents to that of multi-agents. We are planning to solve many challenging problems of multi- agents coordination and competition by using the vision-based reinforcement learning.

Acknowledgment

The authors thanks Mr. Yasutake Takahashi for his efforts of implementing the experiments.

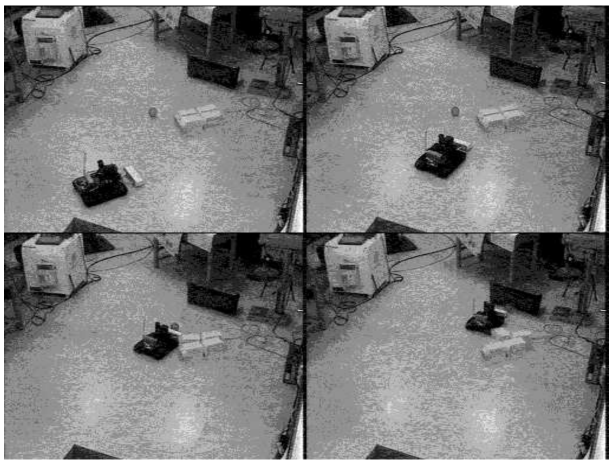


Figure 7: **The robot succeeded in shooting a ball into a goal avoiding an enemy.**

References

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

[2] C. Sammut and J. Cribb. "Is learning rate a good perfomance criterion for learning?". In *Proc. of Conf. on Machine Learning-1990*, pages 170–178, 1990.

[3] S. D. Whitehead. "A complexity analysis of co-operative mechanisms in reinforcement learning". In *Proc. AAAI-91*, pages 607–613, 1991.

[4] L. E. Wixson. "Scaling reinforcement learning techniques via modularity". In *Proc. of Workshop on Machine Learning-1991*, pages 368–372, 1991.

[5] S. Whitehead, J. Karlsson, and J. Tenenberg. "Learning multiple goal behavior via task decomposition and dynamic policy merging". In J. H. Connel and S. Mahadevan, editors, *Robot Learning*, chapter 3. Kluwer Academic Publishers, 1993.

[6] P. Maes and R. A. Brooks. "Learning to coordinate behaviors". In *Proc. of AAAI-90*, pages 796–802, 1990.

[7] J. H. Connel and S. Mahadevan. "Rapid task learning for real robot". In J. H. Connel and S. Mahadevan, editors, *Robot Learning*, chapter 5. Kluwer Academic Publishers, 1993.

[8] M. Asada, S. Noda, S. Tawaratsumida, and K. Hosoda. "Vision-based behavior acquisition for a shooting robot by using a reinforcement learning". In *Proc. of IAPR / IEEE Workshop on Visual Behaviors-1994*, pages 112–118, 1994.

[9] C. J. C. H. Watkins and P. Dayan. "Technical note: Q-learning". *Machine Learning*, 8:279–292, 1992.

[10] R. A. Brooks and M. J. Mataric. "Real robot, real learning problems". In J. H. Connel and S. Mahadevan, editors, *Robot Learning*, chapter 8. Kluwer Academic Publishers, 1993.

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

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

[13] L. P. Kaelbling. "Learning to achieve goals". In *Proc. of IJCAI-93*, pages 1094–1098, 1993.

[14] R. Bellman. *Dynamic Programming.* Princeton University Press, Princeton, NJ, 1957.

[15] J. DEL R. Millan and C. Torras. "A reinforcement connectionist approach to robot path finding in non-maze-like environments". *Machine Learning*, 8:363–395, 1992.

[16] M. Inaba. "Remote-brained robotics: Interfacing ai with real world behaviors". In *Preprints of ISRR'93*, Pitsuburg, 1993.