2007 IEEE International Conference on
Robotics and Automation
Roma, Italy, 10-14 April 2007

Dynamic Programming for Creating Cooperative Behavior of Two Soccer Robots —Part 1: Computation of State-Action Map

Ryuichi Ueda, Kohei Sakamoto, Kazutaka Takeshita and Tamio Arai

***Abstract*— To solve decision making problems of multi-agent systems, researchers have devised complicated methods, which are expected to solve curse of dimensionality. In this paper, we go to the opposite extreme. We generate cooperative behavior of two soccer robots with a simple dynamic programming (DP), which was proposed in '50s. Through the example, the ability of DP on a recent computer is measured and evaluated both qualitatively and quantitatively. We then show that the simple structure of DP is useful in obtaining behavior of robots in a convincing way. In the implementation of DP, space that is spanned by eight variables for decision making is divided into 610 million states. DP solves the optimal actions of two robots in every division and creates a look-up table, which is called a state-action map. The ability of this state-action map is measured by simulation and the result is discussed.**

I. INTRODUCTION

In almost all of studies on cooperative robotics, the curse of dimensionality [1] is discussed implicitly and explicitly. In many studies of multi-agent systems, some kinds of basic behavior are planned for each robot beforehand, and cooperative behavior is planned as their combination. For example, there are many cases of such studies [2], [3], [4], [5] in RoboCup (robot soccer world cup [6]). In these cases, a problem is divided into each subproblems of each robot. State space of all robots are also divided into subspace of each robot. Those studies have made it possible to create various types of cooperative behavior of robots.

From the viewpoint of optimal control, however, excessive assist for emergence of cooperation is not optimal. In many cases, those methods find at most some behavioral patterns that are easy to understand for people.

In this paper, we daringly try creating a huge control policy without techniques on the study of multi-robot systems. Dynamic programming (DP), which was proposed by Bellman [1] in '50s, is applied to a cooperative task of two robots. An algorithm of DP for a finite Markov decision process (a finite MDP) [7] is applied to a continuous system with discretization.

The concrete purposes of this study are as follows.

- We verify that a reasonable result of DP can be obtained within feasible time, and that the result makes two robots cooperate effectively.
- A concept of virtual state transitions is introduced.
- We discuss and deal with a problem on the approximation of the continuous system with a finite MDP, especially in the case of multi-agent systems.
- The computational cost of DP is evaluated.
- The result of DP is evaluated by simulation.

In Sec. II, a scoring task with two robots is defined. Our implementation of DP is explained in Sec. III minutely. The novel concept, virtual state transitions, is introduced in this section. The obtained result is evaluated by simulation in Sec. IV. We conclude this paper in Sec. V.

II. SCORING TASK WITH TWO SOCCER ROBOT

A. RoboCup Four Legged Robot League

We assume an environment, which was used in RoboCup four-legged robot league until RoboCup 2003, for simulation. The size of the soccer field but two inside areas of goals is 4.2[m] by 2.7[m] as shown in Fig.1. Each goalmouth is 600[mm] across. Note that the field is surrounded by a sloped wall.

The current soccer field in the league is larger than the old one, and we would be better off using the new one. In this paper, however, we use the old one so as not to lead to poor continuity with our past works [8], [9], [10], [11]. Though a computation result is evaluated only in simulation in this paper, results of dynamic programming have been applied to actual robots in the past works.

ERS-210, shown in Fig.2, has been used in this league. We assume the use of this robot in simulation. An ERS-210 has three DoF in each leg and in the head. It has a MIPS 192[MHz] as its CPU and 32[MB] RAM. On its nose, there is a color CMOS camera.

B. Scoring Task with Two Robot on Simulation

There are two teammate ERS-210s on the field as shown in Fig. 1. Their task is to bring the ball into the sky-blue goal as soon as possible. As explained later, they can choose some kinds of walking actions and kicking actions. We want to compute and record the optimal pair of their actions for every state of the game. This task is named the scoring task. We never call it the *cooperative* scoring task because they do not have to cooperate with each other unnecessarily.

A robot can observe its position and orientation, which are collectively called *a pose*. Then it can measure the position of the ball when the ball can be observed by its camera. Though some errors or uncertainty are contained in the measurements in the real world, we do not consider them in this paper. They can exchange the measurements by using their wireless LAN

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

R. Ueda, K. Sakamoto, K. Takeshita, and T. Arai are with Department of Precision Engineering, School of Engineering, The University of Tokyo
ryuichi@robot.t.u-tokyo.ac.jp

devices. We assume that time for the exchanges are much shorter than the cycle of decision making.

For purposes of illustration, we define a field coordinate system, Σ_f, and a robot coordinate system, Σ_r, as shown in Fig.3. The measurement of a pose of a robot is obtained as the form of (x, y, θ) on Σ_f, where $|x| \leq 2100$[mm] and $|y| \leq 1350$[mm]. A measurement of the ball position is represented by polar coordinates (r, φ) from the origin of Σ_r. r is the distance between the robot and the ball. φ is then the direction of the ball from the robot. We assume that the robot can observe the ball when $150 \leq r < 3150$[mm] and $|\varphi| \leq 95$[deg]. We use millimeters and degrees for units of distance and angle respectively after this. The speed of the ball cannot be measured.

III. Dynamic Programming for Scoring Task

A. Finite Markov Decision Process and Value Iteration

The scoring task is approximated to one of the finite Markov decision processes (finite MDPs) [7]. The scoring task is represented and approximated by the following formulation.

- An agent, which is the pair of robots in the scoring task, decides its action based on some state parameters $(x_1, x_2, \dots, x_n) = \boldsymbol{x}$. These parameters relate to the state of the agent and its surroundings. The space that is spanned by them is called state space $\mathcal{X}$.
- $\mathcal{X}$ is divided into N discrete states: $s_1, s_2, \dots, s_N$. The set of the discrete states is represented by $\mathcal{S}$. Some of them belong to the set of final states $\mathcal{S}_\mathrm{f}$.
- The agent chooses an action from $\mathcal{A} = \{a_1, a_2, \dots, a_m\}$ every decision making chance.
- Behavior of the agent is defined by state transition probabilities $\mathcal{P}^a_{ss'}$. This symbol denotes the probability that a continuous state $\boldsymbol{x}$ is transferred from s to s' by action a. $\mathcal{P}^a_{ss'}$ never changes in a Markov decision process.
- A reward $\mathcal{R}^a_{ss'} \in \Re$ is set for each set of s, s' and a. The reward is based on the purpose of the agent.

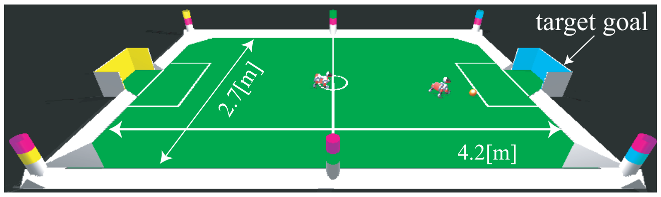


Fig. 1. The Field of RoboCup Four Legged Robot League

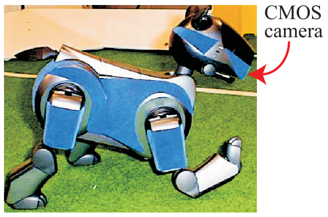


Fig. 2. ERS-210

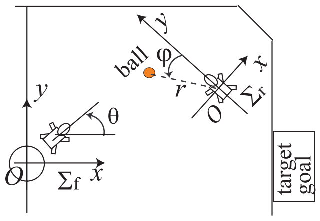


Fig. 3. Coordinate Systems

The agent chooses actions so as to maximize the sum of rewards from a state to a final state. In this finite MDP, we can assume the existence of the optimal mapping from every state to an action: $\pi^* : \mathcal{S} \to \mathcal{A}$. The agent can maximize the summation only by the repetition of recognition of state s and execution of action $a = \pi^*(s)$.

π^* is called the optimal policy. π^* is recorded on a memory array that contains every action $\pi^*(s)$ from s_1 to s_N in this paper. We call this array *a state-action map* (or *a map*).

The expected sum of rewards from s to a final state with π^* is symbolized as $V^*(s)$. It satisfies Bellman optimality equation [7], [1]:

$$V^*(s) = \max_a \sum_{s'} \mathcal{P}^a_{ss'} [\mathcal{R}^a_{ss'} + V^*(s')]. \tag{1}$$

$V^* : \mathcal{S} \to \Re$ is called the optimal state-value function. π^* is obtained from V^* as

$$\pi^*(s) = \operatorname*{argmax}_a \sum_{s'} \mathcal{P}^a_{ss'} [\mathcal{R}^a_{ss'} + V^*(s')]. \tag{2}$$

Value iteration is one of the popular methods of DP and it is used for obtaining V^* and π^* in this paper. This algorithm iterates the following procedure:

$$V(s) \longleftarrow \max_a \sum_{s'} \mathcal{P}^a_{ss'} [\mathcal{R}^a_{ss'} + V(s')] \ (\forall s \in \mathcal{S} - \mathcal{S}_\mathrm{f}) \tag{3}$$

toward a value function V, which is initialized arbitrarily. The value function comes to V^* by the iteration of the above procedure.

B. State Space for Scoring Task

We choose eight state variables: $x_1, y_1, \theta_1, x_2, y_2, \theta_2, r, \varphi$. A robot is called Robot1 and the other is called Robot2. Their poses are represented by (x_1, y_1, θ_1) and (x_2, y_2, θ_2) respectively. (r, φ) is a measurement of the ball from Robot2.

When both of the robots can observe the ball, one of them is regarded as Robot2. Robot2 is chosen by the following rule.

- Only one robot can observe the ball ($-150 \leq r < 3150$ and $|\varphi| \leq 95$), the robot can be Robot2.
- If both of them can observe it, the robot whose r is smaller than that of the other is chosen as Robot2.

When Robot1 and Robot2 trade their names each other, it means that a change of coordinate occurs.

We do not consider the case where the robots cannot observe the ball here. Their behavior in the case is coded by hand as explained later.

C. Actions

An action of the agent in this finite MDP is defined as a pair of Robot1's action and Robot2's action. We assume that they take their actions synchronously. When Robot1 and Robot2 choose actions from $\mathcal{A}^{\mathrm{R1}} = \{a^{\mathrm{R1}}_i | i = 1, 2, \dots, M^{R1}\}$ and $\mathcal{A}^{\mathrm{R2}} = \{a^{\mathrm{R2}}_j | j = 1, 2, \dots, M^{R2}\}$ respectively, the action in the finite MDP is defined as their direct product: $\mathcal{A} = \mathcal{A}^{\mathrm{R1}} \times \mathcal{A}^{\mathrm{R2}}$.

We choose 14 walking actions as shown in Table I from the actions that were used for experiment with an actual ERS-210 in [8]. We assume that every robot executes each action once in a time step. Every walking action is parameterized by $(\delta_x, \delta_y, \delta_\theta)$. They mean the displacement to right direction, the displacement to the forward direction, and the displacement of θ value of a robot respectively.

TABLE I
WALKING ACTIONS

	name	δ_x[mm]	δ_y[mm]	δ_θ[deg]
only for Robot1	Stay	0.0	0.0	0.0
	Backward	0.0	-87.0	0.0
	RightSide	114.0	0.0	0.0
	LeftSide	-107.0	0.0	0.0
for both robots	Forward	0.0	113.0	0.0
	TurnRight	20.0	5.0	-32.5
	TurnLeft	-15.0	5.0	28.0
only for Robot2	ShortForward	0.0	66.0	0.0
	ShortRollRight	25.0	30.0	18.0
	ShortRollLeft	-25.0	30.0	-30.0
	RightForward15	16.0	84.0	2.0
	LeftForward15	-14.0	102.0	-2.6
	RightBackward15	36.0	-94.0	0.0
	LeftBackward15	-35.0	-95.0	0.0

TABLE II
KICKING ACTIONS

	name	r_{after}[mm]	φ_{after}[deg]
only for Robot2	KickForward	2000	0.0
	KickRight	2000	-75.0
	KickLeft	2000	75.0

As shown in the table, each robot uses a different set of actions. The set for Robot2, $\mathcal{A}^{\text{R2}}$, is chosen as a suitable set of actions for approaching to the ball. Moreover, three kicking actions belong to $\mathcal{A}^{\text{R2}}$ are defined as shown in Table II. A kicking action changes the position of the ball. $(r_{\text{after}}, \varphi_{\text{after}})$ in the table means the position of the ball (r, φ) after the kick by the robot. The pose of Robot2 does not change after any kicking action. Robot2 always chooses a kick action when $150 \le r < 250$[mm] and $-35 \le \varphi < 35$[deg]. In the case of an actual ERS-210, fine position adjustment is required for a kick from that range of (r, φ). In the simulation, however, we do not consider it for simplicity.

The actions of Robot1, $\mathcal{A}^{\text{R1}}$, are selected as the robot can run in four directions and turn to both directions. We fix Robot1's action to *Stay* when Robot2 kicks the ball. The number of actions in $\mathcal{A}$ is $M = 73$ with the above setting.

Though this setting looks like separation of roles of two robots, it is never a definite role assignment. The robots can switch their names each other. Robot1 can go to the ball so as to kick. In this case, Robot1 and Robot2 change their names when the distance of the ball from Robot1 becomes nearer than that of Robot2.

D. State Space Discretization and Final State Definition

We discretize the state space $\mathcal{X}$ as shown in Table III. $[\cdot]_i$ or $[\cdot]$ denote an interval on each axis. A discrete state is represented by a combination of the intervals as $s = ([x_1], [x_2], [y_1], [y_2], [\theta_1], [\theta_2], [r], [\varphi])$. The number of states reaches $14^2{\cdot}9^2{\cdot}15^2{\cdot}9{\cdot}19 = 610,829,100$ with this definition.

State s is chosen as a final state if it fulfills the two conditions: 1) Robot2 can kick the ball, and 2) there is more than 50% of probability of scoring with an appropriate kicking action. The probability of scoring is computed by the following Monte Carlo simulation:

1) ζ kinds of (x_2, y_2, θ_2) are chosen from state s at random.
2) Every kicking action is tried from all of the chosen poses and the number of goals are counted respectively.

If Robot2 can obtain more than $\zeta/2$ goals by one of the actions, state s is regarded as a final state.

E. State Transition

A state transition can be represented by

$$\mathcal{P}^a_{ss'} = P(s'|s, a) \quad \text{where, } a \in \mathcal{A}, \tag{4}$$
$$s = ([x_1], [x_2], [y_1], [y_2], [\theta_1], [\theta_2], [r], [\varphi]), \text{and}$$
$$s' = ([x_1]', [x_2]', [y_1]', [y_2]', [\theta_1]', [\theta_2]', [r]', [\varphi]').$$

All of the state transition probabilities are calculated and stored on memory before value iteration; otherwise a probability of the same state transition is calculated redundantly.

1) Reduction and Decomposition of State Transitions: By simple arithmetic, however, the combinations of (s, s', a) reaches $N^2 M = 610,829,100^2{\cdot}73 = 27,237,189,826,697,130,000$. Thus, whether this number can be reduced or not determines the feasibility of DP.

This number can be reduced by cutoff of state transitions that fulfill $\mathcal{P}^a_{ss'} < \eta$. In this case, the number of state transitions is not over $1/\eta$ toward a set of (s, a). We choose $\eta = 0.01$ for the scoring task. In this case, the number is reduced to $610,829,100 \cdot 73 \cdot 100 = 4,459,052,430,000$.

When we can decompose state transitions into some independent events, moreover, each state transition probability can be represented by the multiplication of the probabilities. In this case, the amount of memory for recording the state transitions can be reduced.

We can consider the following events in the task.

(i) displacement of a robot by an walking action

TABLE III
DISCRETIZATION OF THE STATE SPACE

	definition of intervals	
x_1	$[x_1]_i \equiv \big[300i - 2100, 300(i+1) - 2100\big)$	$(i = 0, 1, \ldots, 13)$
x_2	$[x_2]_i \equiv \big[300i - 2100, 300(i+1) - 2100\big)$	$(i = 0, 1, \ldots, 13)$
y_1	$[y_1]_i \equiv \big[300i - 1350, 300(i+1) - 1350\big)$	$(i = 0, 1, \ldots, 8)$
y_2	$[y_2]_i \equiv \big[300i - 1350, 300(i+1) - 1350\big)$	$(i = 0, 1, \ldots, 8)$
θ_1	$[\theta_1]_i \equiv \big[24i - 180, 24(i+1) - 180\big)$	$(i = 0, 1, \ldots, 14)$
θ_2	$[\theta_2]_i \equiv \big[24i - 180, 24(i+1) - 180\big)$	$(i = 0, 1, \ldots, 14)$
r	$[r]_i \equiv \begin{cases} \big[100i + 150, 100(i+1) + 150\big) & \text{—(1)} \\ \Big[50\,(i - 3/2)^2 + 675/2, 50\,(i - 1/2)^2 + 675/2\Big) & \text{—(2)} \end{cases}$ $(1) : (i = 0, 1, 2), (2) : (i = 3, 4, \ldots, 8)$	
φ	$[\varphi]_i \equiv \big[10i - 95, 10(i+1) - 95\big)$	$(i = 0, 1, \ldots, 18)$

(ii) relative displacement of the ball by Robot2's walk
(iii) displacement of the ball by Robot2's kick

When the robot chooses an walking action at $(x, y, \theta, r, \varphi)$, the posterior state $(x', y', \theta', r', \varphi')$ fulfills

$$\begin{pmatrix} x'-x \\ y'-y \\ \theta' \end{pmatrix} = \begin{pmatrix} \delta_x \cos\theta - \delta_y \sin\theta \\ \delta_x \sin\theta + \delta_y \cos\theta \\ \theta + \delta_\theta \end{pmatrix}, \text{ and} \tag{5}$$

$$\begin{pmatrix} r' \\ \varphi' \end{pmatrix} = \begin{pmatrix} \sqrt{(r\cos\varphi - \delta_x)^2 + (r\sin\varphi - \delta_y)^2} \\ \arctan[(r\sin\varphi - \delta_y)/(r\cos\varphi - \delta_x)] - \delta_\theta \end{pmatrix}. \tag{6}$$

if any collision of two objects are not considered.

Toward walking action a, Eq. (4) is decomposed into two kinds of probabilities: P_{robot} and P_{ball}. They denote a probability for event (i) and that for event (ii) respectively. Eq. (5) implies that the transition rule on x-axis and y-axis does not depend on the prior position (x, y) of the robot. Therefore, P_{robot} becomes a conditional probability $P_{\text{robot}}(\Delta[x_i], \Delta[y_i], [\theta_i]' | [\theta_i], a^{Ri})$ $(a^{Ri} \in \mathcal{A}^{Ri}, i = 1, 2)$, where $\Delta[x]$ and $\Delta[y]$ denote the relative position of the pairs: $([x_i], [x_i]')$ and $([y_i], [y_i]')$ respectively. P_{ball} is then represented by $P_{\text{ball}}([r]', [\varphi]' | [r], [\varphi], a^{R2})$ $(a^{R2} \in \mathcal{A}^{R2})$.

P_{robot} and P_{ball} can be obtained by Monte Carlo integrations respectively. P_{robot} and P_{ball} are then recorded on look-up tables respectively.

A state transition probability toward a set of walking actions can be calculated from P_{robot} and P_{ball} as

$$\mathcal{P}^a_{ss'} = P_{\text{robot}}(\Delta[x_1], \Delta[y_1], [\theta_i]' | [\theta_i], a^{R1}) \cdot P_{\text{robot}}(\Delta[x_2], \Delta[y_2], [\theta_2]' | [\theta_2], a^{R2}) \cdot P_{\text{ball}}([r]', [\varphi]' | [r], [\varphi], a^{R2}) \tag{7}$$

For the event (iii), $\mathcal{P}^a_{ss'}$ can be represented by

$$\mathcal{P}^a_{ss'} = P_{\text{kick}}([r]', [\varphi]' | a^{\text{R2}}) \quad (a^{\text{R2}}\text{: a kicking action}) \tag{8}$$

if we assume that the ball never collides with the wall or a robot. This equation means that the state transition depends only on kicking actions. Since we do not consider the distribution of the ball position after a kick, P_{kick} is deterministic in the discrete state space $\mathcal{S}$.

2) Virtual State Transition: We have decomposed the state transition probabilities into $P_{\text{robot}}, P_{\text{ball}}$, and P_{kick}. On the other hand, we have neglected the existence of the following events that occur in the task.

(1) collision between the ball and the wall around the field
(2) change of Robot1/2 (abbreviation of Robot1 and Robot2)
(3) collision between the ball and a robot
(4) collision between Robot1 and Robot2

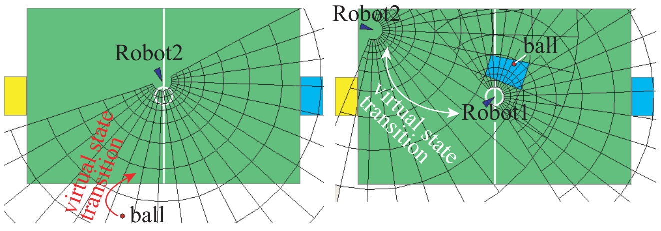
(1) and (2) should be especially considered in value iteration. If (1) is neglected, the ball is regarded as being out of the field. In this case, a state-value function obtained by value iteration will contain great errors. When (2) is not considered, the roles of the robots are fixed. If (3) and (4) are neglected in value iteration, the efficiency of the state-action map will decrease.

However, considerations of the above events increase the dependency of each state variable. If we consider (1), P_{kick} depends not only on a kicking action, but also the pose of Robot2. If Robot2 and Robot1 should be changed after the kick, the pose of Robot1 should also be considered. P_{robot} and P_{ball} also depend on all of the eight state variables if (2) is considered.

To solve this problem, we introduce *virtual states* and *virtual state transitions*. In the case of (1), we decompose a state transition into the following two processes: 1) the ball goes out of the field and stops, and 2) the ball returns from the outside of the field to the inside instantaneously. A ball out state, as shown in Fig. 4(a), is regarded as a virtual state. The exportation of the ball is regarded as a virtual state transition. Since virtual state transitions occur instantaneous, no reward is given. In the actual world, such a pair of state transitions never happens.

In a value iteration algorithm, however, we can consider them if we prepare memory space for recording the value of virtual states and their state-transition probabilities. There are 118, 788, 390 ball out states in the set of discrete states defined in Table III and the memory can be prepared.

In a virtual state transition from ball out state s, interval $[r]$ of state s moves to an inner interval, which contains a part of the field, as shown in Fig. 4(a). Since s' is fixed toward a ball out state s, $\mathcal{P}^a_{ss'} = P_{\text{ball out}} = 1$ is the state transition model for each of them. In this case, a is the teleportation. We call it *a virtual action*.



(a) a ball out state (b) a reverse state

Fig. 4. Virtual States and Virtual State Transitions

We also regard the event (2) as a virtual state transition. Actions of the robots and a change of Robot1/2 by the actions is decomposed into a set of two state transitions. We define reverse states, which are virtual states, as the states where Robot1/2 should be switched. An example is shown in Fig. 4(b). In a reverse state, the change of Robot1/2 occurs instantaneously. We implement a Monte Carlo algorithm to find a reverse state. In the algorithm, distances of the ball from Robot1 and Robot2 are compared from various continuous state $\boldsymbol{x}$ that belongs to s when both of the robots can observe the ball. If the average distance from Robot1 is smaller than the other, the state s is regarded as a reverse state.

Virtual state transition probabilities P_{rev} from reverse states can be represented as

$$P_{\text{rev}}([r]', [\varphi]' | \Delta[x], \Delta[y], [\theta_1], [\theta_2], [r], [\varphi]). \tag{9}$$

P_{rev} can be computed by a Monte Carlo sampling of values $(\Delta x, \Delta y, \theta_1, \theta_2, r, \varphi)$ from the intervals $(\Delta[x], \Delta[y],$

$[\theta_1], [\theta_2], [r], [\varphi]$). $(\Delta x, \Delta y)$ denotes a relative position of Robot1 and Robot2. In our algorithm for computing P_{rev}, the sampling is not at random but fixed. Three values (two boundary values and the mean value) are chosen from $\Delta[x]$ and $\Delta[y]$ and two boundary values are chosen from the other intervals. $45, 242, 367$ kinds of virtual state transitions are recorded on a look-up table by this implementation.

F. Reward

The reward is given as $\mathcal{R}^a_{ss'} = -1$[step] when the robots take their actions respectively. When both of the robots cannot observe the ball, $\mathcal{R}^a_{ss'} = -\infty$[step] is given. Therefore, the purpose of decision making is to reach a final state as small number of steps as possible without losing the ball.

G. Value Iteration

We implement the value iteration algorithm on a computer that has 3.0[GB] RAM, a 300[GB] hard disk drive (HDD) and 3.2[GHz] Pentium D CPU. The algorithm starts from the definition of state space $\mathcal{S}$, set of actions $\mathcal{A}$, and reward $\mathcal{R}^a_{ss'}$. After that, two look-up tables of a state-value function and a state-action map are created on HDD. Each look-up table is divided into $14^2 = 196$ files with respect to a set of $([x_1], [x_2])$.

Each action is represented by a unique number from 0 to 72. 1 byte is required for recording each of these numbers. Before value iteration, zero is filled in the state-action map except the elements of final states, ball out states, and reverse states. They are given unique numbers: 255, 254, and 253 respectively.

In the look-up table for the state-value function, final states are given zero as their value. Nonzero values are given to the other states. A 2-byte unsigned integer is used for representing a value. -1[step] is equivalent to 100 on the integer data.

In the next process, $P_{\text{robot}}, P_{\text{ball}}$, and P_{rev} are computed respectively. ($P_{\text{ball out}}$ and P_{kick} are deterministic.) As shown in Table IV, all of the state transition probabilities can be represented by much smaller kinds of probabilities than N^2M.

TABLE IV
NUMBER OF STATE TRANSITIONS

probabilities	number of combinations
P_{robot}	$1,024$
P_{ball}	$7,485$
P_{rev}	$45,242,367$

H. Computation of State-Action Map

The value iteration algorithm is then executed. Required files are loaded on memory, improved by Eq. (3), and rewritten. Since a Pentium D has two cores, computing speed is enhanced by multiprocessing. We divide our value iteration algorithm to Process 1 for $[x_1]_i$ $(i = 0, 1, \ldots, 6)$ and Process 2 for $[x_1]_i$ $(i = 7, 8, \ldots, 13)$ and execute them simultaneously.

1) Computation Result: We have obtained a state-action map by the above implementation. The processes has been executed for ten days (243 hours). Process 1 and 2 have been executed 51 and 55 sweeps respectively. A sweep denotes the execution of Eq. (3) for all $s \in \mathcal{S} - \mathcal{S}_{\text{f}}$. It took only five minutes to calculate $P_{\text{robot}}, P_{\text{ball}}$, and P_{rev}. The size of the map, which is called the 8D map hereafter, is $610, 829, 100$ bytes because 1 byte data is allocated to each state.

We have recorded the maximum difference of value $V(s)$ before and after the process Eq. (3) in each sweep. The differences are illustrated in Fig. 5. The horizontal axis indicates the computation time. The vertical axis is a logarithmic one for the difference of value. Roughly speaking, it takes twice processing time to reduce the difference to a tenth value as shown in this graph.

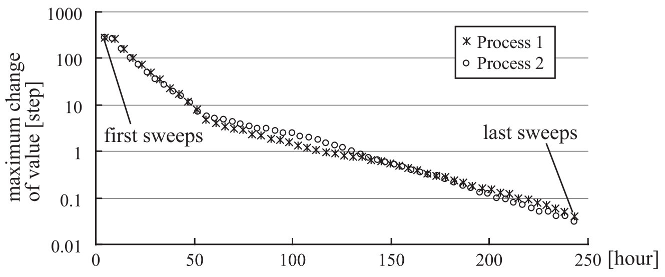


Fig. 5. Reduction of Maximum Change of Value

2) Discussion about Calculation Cost of Value Iteration: As shown above, we can obtain a huge state-action map within ten days. If there is one more state variable, the computation time will jump to 100 days. By definition, such a basic value iteration cannot deal with an infinite number of state variables. However, we think that the number of state variables will be increased one-by-one. Nowadays, there are two topics, which will accelerate the ability of value iteration, about commercial CPUs.

One of them is parallelization of CPUs. Though it took ten days to obtain the 8D map, this time can be reduced by additional multiprocessing. It would take 20 days if we had not executed the value iteration algorithm on two processes. Some kinds of CPU for personal computers have two cores nowadays and it will increase in future.

The other is the release of 64-bit CPUs for the public. They make coding of DP easy through the explosion of address space. The value iteration algorithm in this paper had to read and write data between RAM and HDD frequently due to the limitation of the memory space. Such a process becomes the cause of slow down and bugs. If we use a 64-bit CPU, the code for value iteration will be simple.

I. Behavior of Robots with The 8D Map

1) Emergence of Cooperative Behavior: We show two examples of the robots' behavior obtained by the 8D map. When both of the robots cannot observe the ball, RobotA chooses *TurnLeft* and the other chooses *TurnRight* so as to search it.

In Fig. 6(a), two robots, which are called RobotA and RobotB, started moving from the bottom left corner and in front of the sky-blue goal respectively. At first, RobotA is Robot1 and RobotB is Robot2. The ball is put at the center of the field. In this figure, some important positions of RobotA and RobotB are numbered as Ai and Bi respectively. The numbers are synchronized.

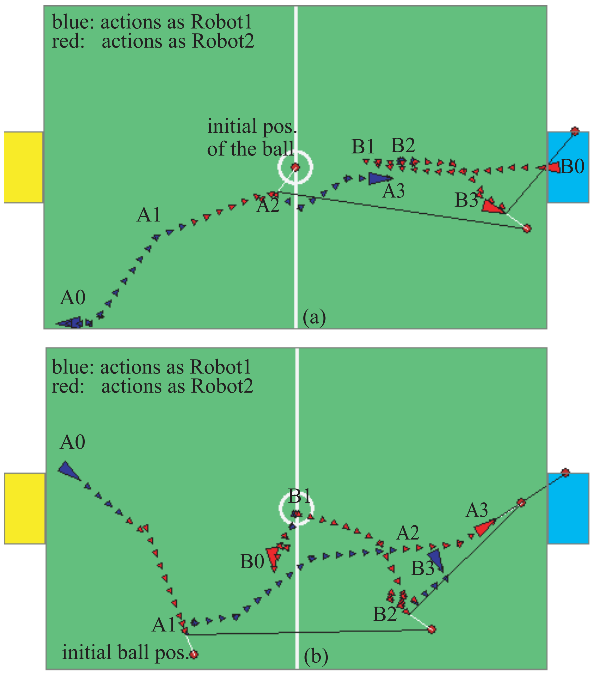


Fig. 6. Examples of Cooperative Behavior

Both of the robots go to the ball until the state (A1,B1). RobotB, however, goes back the way at B1. RobotB hands over the ball to RobotA at the moment though it is Robot2. After that, RobotA becomes Robot2 and RobotB changes into Robot1. As shown this example, Robot2 does not always go to the ball and the role of a robot is never fixed by whether the robot is Robot1 or Robot2. At A2, RobotA kicks the ball. RobotB waits for the kick at B2. After the pass, RobotB reaches B3, which is a final state. At B3, *KickLeft* is chosen as a final shot by an algorithm, which is explained later. The ball is kicked into the goal and the task is finished.

In a precise sense, the pair of RobotA's kick at A2 and RobotB's wait at B2 is only a way to reduce the number of steps. However, it is cooperative behavior because the reduction of steps and the role of each robot emerge. The kick by RobotA at A2 can be regarded as *a pass*. In this case, RobotA is the passer and RobotB is the receiver.

Another example with the 8D map is shown in Fig. 6(b). We can see two passes in one trial. RobotB waited for RobotA's kick at B1 at the first pass. RobotA then waited for RobotB's kick at A2.

2) Countermeasure against Ill-Effect of Discretization: On the other hand, the cooperative behavior given by the 8D map is not perfect due to the coarse discretization of the state space. As shown in Fig. 6(a, b), the receiver waits for the ball at a distant point. The margin is necessary because poses of the robots and the position of the ball are unspecified in the discrete state space.

Another problem is observed in the 8D map. In some trials, both of the robots stop walking due to infinite virtual state transitions between two reverse states. That is because some virtual state transitions change a reverse state to another reverse state. This kind of deadlock is inevitable under the discretization.

We think that reduction of the ill-effects can be possible by additional algorithms, which are coded by hand. We do not have to chase the completeness of a state-action map. However, a tenuous ad-hoc method should not be chosen.

Our proposition toward this problem is as follows. When decision making in continuous state space $\mathcal{X}$ has certain advantage over that in discrete space, we should add some algorithms that decide actions in $\mathcal{X}$.

We therefore implement the following additional algorithms.

(a) **Decision of Shot:** When Robot2 judges whether a score is possible or not by the calculation in $\mathcal{X}$ without the 8D map. When it is possible, Robot2 choose an appropriate kicking action.
(b) **Detection of Reverse State:** When a state is a reverse state in $\mathcal{X}$, a virtual state transition occurs without relation to the 8D map. When a non-reverse state in $\mathcal{X}$ is regarded as a reverse state in the 8D map, there is no information for decision making. In this case, a change of Robot1/2 occurs and new Robot2 chooses *Forward* in order to avoid the deadlock.

IV. Simulation for Evaluation

We evaluate the 8D map with a simulator. In the evaluation, $10,000$ initial states are chosen. Each trial starts from one of them. When the number of decisions is over 180 times, the trial is regarded as a failure trial. In simulation, the wall reflects the ball with some extent of reflection.

A. Effectiveness of Cooperation

At first, we evaluate whether the cooperation by the 8D map is effective or not. We create another state-action map, called the 5D map here, under the consideration of $(x_2, y_2, \theta_2, r, \varphi)$. A robot with this map repeats a set of some walking actions and a kicking action so as to score. The discretization interval of each axis is identical with that of the 8D map.

We try the following cases: 1) two robots with the 8D map, 2) two robots with the 5D map, and 3) one robot with the 5D map. We measure their success rates. Only when the robot(s) can score from a common initial state in all of three cases, the sum of rewards is recorded respectively.

We also evaluate another case where the collision of two robots are considered in simulation. When the distance between two robots is shorter than 50[mm], we regard this state as a collision. The trial is then regarded as a failure trial.

TABLE V
EFFICIENCY OF COOPERATION

cases	success rate	average of steps	success rate (stop by collision)
8D Map	97.4[%]	36.3[step]	95.9[%]
5D Map (two robots)	93.8[%]	40.9[step]	70.9[%]
5D Map (one robot)	83.2[%]	50.5[step]	—

Table V shows the results. When the collisions are not simulated, both the success rate and the number of steps by the 8D map are better than those by the 5D map. 4.6[step] were reduced by the pass behavior. The 8D map is more effective when the collision of the robots is considered. It is interesting that the robots with the 8D map tend to move apart from each other though their collision is not considered in the value iteration. The reason is simply because the robots became a passer and a receiver.

B. Effectiveness of the Additional Algorithms

We evaluate the additional algorithms: (a) and (b) mentioned at the end of Sec. III. In Table VI, we show the ability of algorithm (a). It can reduce the number of steps surely though the reduction of steps is only 0.5[step].

The result in Table VII has two faces. The 8D map is incomplete without algorithm (b). The percentage only with the 8D map, 62.7[%], is worse than any result with the 5D map. On the other hand, if the incompleteness is compensated by (b), the 8D map can output cooperative behavior with the high percentage. When (b) is used, there is no failure by the deadlock at reverse states in the $10,000$ trials. The 2.6[%] of failures at the use of (b) happen when the ball stops by the wall, or when the ball cannot be found by the ball search algorithm. In the former cases, robots cannot approach to the ball for fear that they collide with the wall. This is the ill-effect of coarse discretization of robots' positions.

TABLE VI
EFFICIENCY OF SHOT DECISION IN $\mathcal{X}$

cases	success rate	average of steps
without (a)	97.3[%]	39.3[step]
with (a)	97.4[%]	38.8[step]

TABLE VII
AVOIDANCE OF DEADLOCK

cases (with (a))	success rate
without (b)	62.7[%]
with (b)	97.4[%]

V. CONCLUSION AND FUTURE WORKS

In this paper, A state-action map for the scoring task with two robots on RoboCup is created by dynamic programming (DP). With the map and the two algorithms that decide actions in the continuous state space, the two simulated robots can reduce the number of steps for a score with their cooperative behavior. They can change the roles of a passer and a receiver flexibly though we never give them a role assignment in advance.

The concrete data obtained in this study is as follows.

- Time for creating the state-action map, which contains 610 million elements, is ten days with a computer that has a 3.2[GHz] Pentium D CPU, 3.0[GB] RAM, and 300[GB] HDD.
- $27 \cdot 10^{18}$ kinds of state transitions are reduced into $4.5 \cdot 10^{7}$ kinds of partial state transitions. The reduction is achieved not only by the cutoff probability η and decomposition of independent events, but also by the introduction of virtual state transitions.
- When collisions of the robots are not considered, the cooperative state-action map (the 8D map) can make 4.6[step] advantage over the non-cooperative state-action map (the 5D map). When the collision is considered, there is a 25.0[%] disparity between the success rate of the 8D map and that of the 5D map. It means that the 8D map tends to make the robots play at a distance.
- The deadlock at reverse states can be eliminated by decision making in the continuous state space. In simulation, the additional algorithm for reverse states can enhance the success rate from 62.7[%] to 97.4[%]. It is a meaningful result for the use of DP in multi-agent systems.

When we want to use the 8D map for the actual ERS-210s, some problems should be solved. First of all, the robot must be able to behave as the map instructs. We have confirmed that ERS-210 and ERS-7 can approach to the ball based on a state-action map [8], [12]. The accuracy of kicking actions will be a serious problem for the robots. Then, the required amount of memory must be reduced since the 8D map is too large to be stored on the memory of ERS-210. This problem is dealt with the next paper of this series.

REFERENCES

[1] R. Bellman, *Dynamic Programming*. Princeton University Press, 1957.
[2] G. Nitschke, "Emergent cooperation in robocup: A review," in *RoboCup 2005: Robot Soccer World Cup IX*, 2006, pp. 512–520.
[3] M. R. Khojastech *et al.*, "Using Learning Automata in Cooperation among Agents in a Team," in *Proc. of International RoboCup Symposium*, 2004, pp. CD–ROM.
[4] G. Fraser *et al.*, "Cooperative Planning and Plan Execution in Partially Observable Dynamic Domains," in *RoboCup 2004: Robot Soccer World Cup VIII*, 2005, pp. 524–531.
[5] H. Fujii, "Cooperative Control Method Using Evaluation Information on Objective Achievement," in *Proc. of DARS*, 2004, pp. 201–210.
[6] T. Laue *et al.*, "SimRobot –A General Physical Robot Simulator and Its Application in RoboCup," in *In RoboCup 2005: Robot Soccer World Cup IX*, 2006, pp. 173–183.
[7] R. S. Sutton, "Generalization in Reinforcement Learning: Successful Examples Using Space Coarse Coding," in *Neural Information Processing Systems*, 1996, pp. 1038–1044.
[8] R. Ueda *et al.*, "Vector Quantization for State-Action Map Compression," in *Proc. of ICRA*, 2003, pp. 2356–2361.
[9] R. Ueda *et al.*, "Mobile Robot Navigation based on Expected State Value under Uncertainty of Self-localization," in *Proc. of IROS*, 2003, pp. 473–478.
[10] R. Ueda *et al.*, "Expansion Resetting for Recovery from Fatal Error in Monte Carlo Localization – Comparison with Sensor Resetting Methods," in *Proc. of IROS*, 2004, pp. 2481–2486.
[11] R. Ueda *et al.*, "Real-Time Decision Making with State-Value Function under Uncertainty of State Estimation," in *Proc. of ICRA*, 2005.
[12] K. Takeshita *et al.*, "Fast Vector Quantization for State-Action Map Compression," in *Proc. of IAS-9*, 2006, pp. 694–701.