Dynamic Task Assignment in a Multiagent/Multitask Environment based on Module Conflict Resolution

Eiji Uchibe, Tatsunori Kato, Minoru Asada, Koh Hosoda
Graduate School of Eng., Dept. of Adaptive Machine Systems
Osaka University, Suita, Osaka 565-0871, Japan
uchibe@er.ams.eng.osaka-u.ac.jp

Abstract

It is necessary to coordinate multiple tasks in order to cope with larger-scaled and more complicated tasks. However, it seems very hard to accomplish the multiple tasks at the same time. This paper proposes a method to resolve a conflict between task modules through the processes of their executions. Based on the proposed method, the robot can select an appropriate module according to the priority. In addition, we apply the module conflict resolution to a multiagent environment. Consequently, multiple tasks are automatically allocated to the multiple robots. As a task example, a soccer game is selected to show the validity of the proposed method. Real experiments are shown, and a discussion is given.

1 Introduction

In general, a robot is required to accomplish a variety of tasks in different environments. What is more important is to develop a method that can cope with multiple tasks at the same time. One approach to deal with such multiple tasks is to make some modules corresponding to the tasks. Each module has an ability to accomplish the corresponding task, that makes the programming easier. The most simple implementation is that one robot is allocated to the one task in advance. However, this approach seems inefficient from a viewpoint of the cost performance. We should not increase the number of robots even though the number of tasks increases.

Fontán and Matarić [3] proposed a method to divide a labor into exclusive *spatial* territories. That is, the working area is assigned to the robot. In order to realize their method, the knowledge about the working-field is given to the designer in advance.

Kuniyoshi [5] developed a framework called *Cooperation by Observation*. Although they realized a number of cooperation in the real environment, the conflict between the modules should be considered in advance. Parker [7] proposed an architecture called *L-ALLIANCE*. Castelpietra et al [2] has shown a dynamic role (task) assignment method based on explicit communication. Although they realized some cooperation in the real environment, the number of the tasks for each robot is one. Therefore, the relationship between robots and tasks is exactly one to one correspondence.

This paper proposes a method to assign the multiple tasks to the multiple robots in a time-varying environment. Unlike previous work [2, 3, 7], we focus on the incompatibility of the modules called *module conflict* through the interactions with the environment by considering to what extent the *executed* module affects the *accomplished* modules.

The robot can discriminate the conflict modules between the current *executed* module and the *accomplished module* automatically. Based on the proposed method, the robot can accomplish the tasks as many as possible. The module conflict resolution is also applied to a multiple robots environment using explicit communication.

As a task example, a simplified soccer situation including multiple robots is introduced. Because there are several tasks to perform the soccer game, each robot has to do its own tasks in a dynamically changing environment. Based on the proposed method, two robots show the defending behaviors while they change the roles according to the situation. Real experiments are shown and a discussion is given.

2 Dynamic Task Assignment

2.1 Definition of the Terminology

For reader's understanding, we introduce some important terms. A **multitask** is a composition of multi-

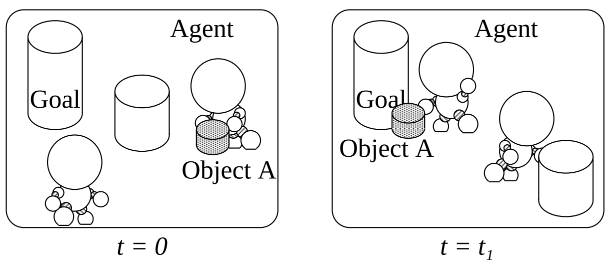
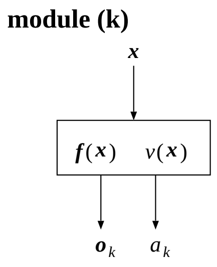
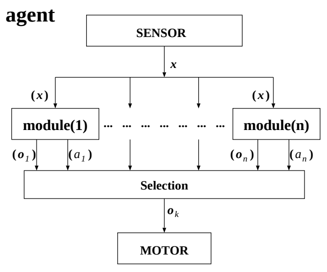


Figure 1: The definition of the task.



(a) module



(b) agent

Figure 2: An architecture of the proposed method.

ple **tasks**, where a task means transfer from the initial state to the desired state by robot. A **module** consists of a policy (controller) and an evaluation function which indicates to what extent the current task is accomplished.

Figure 1 shows a simple example. There are two robots, one object (A), one obstacle, and, one goal (destination) in the environment. In this case, the multitask and the task are specified as follows:

multitask : To carry object A to the goal.
tasks : "Push object A", "Avoid an obstacle", "Approach an obstacle", "Push an obstacle", and so on.

2.2 Module

Figure 2 (a) shows a definition of a module. Each module $m_k \in \mathcal{M}$ has two functions. One is a *policy* $f_k(\boldsymbol{x})$ which maps from a state $\boldsymbol{x}$ to an action $\boldsymbol{o}_k$, and the other is an evaluation function $v_k(\boldsymbol{x})$. The module outputs $\boldsymbol{o}_k$ based on $f_k(\boldsymbol{x})$ in order to minimize the evaluation function $v_k(\boldsymbol{x})$. The module can be regarded as an actor-critic architecture [8]. In addition, we standardize $v_k(\boldsymbol{x})$ by

$$a_k = \begin{cases} 0 & (v_{k,\inf} < v_k(\boldsymbol{x})) \\ \dfrac{v_k(\boldsymbol{x}) - v_{k,\inf}}{v_{k,\sup} - v_{k,\inf}} & (v_{k,\sup} < v_k(\boldsymbol{x}) \leq v_{k,\inf}) \\ 1 & (0 \leq v_k(\boldsymbol{x}) < v_{k,\sup}) \end{cases}, \tag{1}$$

where $v_{k,\inf}$ and $v_{k,\sup}$ are thresholds. Hereafter, we call a_k *task accomplishment* $(0 \leq a_k \leq 1)$. Using the *task accomplishment*, the subset of modules $\mathcal{M}_a$ is defined by

$$\mathcal{M}_a = \{m_i | a_k = 1\}.$$

A module $m \in \mathcal{M}_a$ is no longer executed because it has been already *accomplished.* In this sense, some tasks might have been *accomplished* even if the corresponding modules have never been executed.

We define a new measure e_i by

$$e_i(t) = \begin{cases} 1 & \text{if } m_i \text{ is a selected module,} \\ a_i(t) & \text{otherwise.} \end{cases} \tag{2}$$

Hereafter, e_i is called *task execution*, which is a scalar value between 0 and 1. A case of $e_i = 1$ and $a_i \neq 0$ means that the robot can not minimize the evaluation function although the robot attempt to accomplish the i-th task. To deal with time-varying environment, the averaged *task execution* is calculated by

$$\bar{e_i}(t) = \rho\bar{e_i}(t-1) + (1-\rho)e_i(t), \tag{3}$$

where ρ is a forgetting factor between 0 and 1.

2.3 Module conflict resolution

Although it is desirable for a robot to accomplish given multiple tasks, there would be some modules incompatible to each other according to the situation. Therefore, it is important for a robot to resolve such conflicts between the modules automatically.

In order to detect the conflict between the modules m_i and m_j, a correlation of e_i and a_j is utilized. $\Delta r(e_i, a_j)$ is an index to measure how *task execution* of m_i influences *task accomplishment* of m_j. The modules that are in conflict with the module $m_j \in \mathcal{M}_a$ is determined by

$$\mathcal{M}_c = \{m_i | \Delta r(e_i, a_j) < 0, \quad m_j \in \mathcal{M}_a\}.$$

In case of $\Delta r(e_i, a_j) \geq 0$, the robot can cope with both the *execution* of the module m_i and the *accomplishment* of the module m_j. On the other hand, in case of $\Delta r(e_i, a_j) < 0$, the execution of the module m_i prevents the *accomplishment* of the module m_j.

The rest of the modules is determined as

$$\mathcal{M}_{ca} = \mathcal{M} - \mathcal{M}_a - \mathcal{M}_c.$$

At the beginning, $\mathcal{M}_{ca} = \mathcal{M}$, $\mathcal{M}_a = \phi$, and $\mathcal{M}_c = \phi$. The robot selects the module by

$$m_s = \arg \max_{m' \in \mathcal{M}_{ca}} U(m'), \tag{4}$$

where U is a priority which is given in advance. $\mathcal{M}_a$, $\mathcal{M}_c$ and $\mathcal{M}_{ca}$ are updated according to the changes of the environment in order to accomplish the tasks as many as possible.

Each module outputs an action based on the policy. The problem is how the robot should integrate the several outputs from the modules. Simple realization is based on the weighted sum of outputs. However, the resultant behavior is not guaranteed as an optimal one, and it may lead to some serious situations. Then, we introduce a priority function U in order to make the priority. The robot follow the procedures every time step:

1. For the all modules, *task accomplishment* and *task execution* are computed using equations (1) and (2).
2. Calculate subsets $\mathcal{M}_a$, $\mathcal{M}_c$, and $\mathcal{M}_{ca}$.
3. Select a module m_s from $\mathcal{M}_{ca}$ using equation (4).
4. Execute an action $\boldsymbol{o}_s$ based on the policy of the module m_s.

Because the actual outputs of the robot is one of the outputs, the behavior of the robot has a physical meaning for the selected module.

3 Extension to a Multiagent Environment

As described in the previous section, we proposed a method to detect conflicts between modules. However, the modules belonging to $\mathcal{M}_c$ remain *un-executed* in the current system. Therefore, we extend our method to a multiagent system by adding multiple robots that take charge of $\mathcal{M}_c$. We assume that

- each robot has the same set of modules $\mathcal{M}$.
- each robot can get the information about *task accomplishment* and *task execution*.

This can be implemented by a blackboard system. Each robot has its own parameters such as *task accomplishment*, *task execution*, and module subsets ($\mathcal{M}_a$, $\mathcal{M}_c$, and $\mathcal{M}_{ca}$). Hereafter, "^{k}" denotes the parameter of the k-th robot ^kR. In order to measure how much the robot ^kR works, the load $^kL_j(t)$ is calculated by

$$^kL_j(t) = \sum_{i=j}^{n} {}^ka_i U(m_i), \tag{5}$$

where $U(m_i)$ is the priority of module m_i.

For the current module m_j determined by Eq. (4), the robot ^kR judges whether m_j should be *executed* or not. A procedure is described as follows:

1. Check *task accomplishment* of m_j.
   Go to step 4 if no robot has already accomplished the task.

2. Check the remaining power of ^kR.
   If $^kL(t) < L_{\max}$, go to step 4 ($L_{\max}$ is a threshold).

3. Entrust the module m_j to the other robot
   Subtract m_j from $^k\mathcal{M}_{ca}$, and add m_j to $^k\mathcal{M}_e$, where $^k\mathcal{M}_e$ denotes the subset of the modules that are accepted by other robots. Then, return to step 1.

4. Check the module conflict.
   Go to step 5 if the modules m_j and m_i are compatible each other. Otherwise, add m_j to $^k\mathcal{M}_c$. Then, go to step 1.

5. Execute m_j.
   If the robot R_k *accomplished* the module m_j, transfer m_j from $^k\mathcal{M}_c$ to $^k\mathcal{M}_a$. Then, go to step 1. Otherwise, execute m_j.

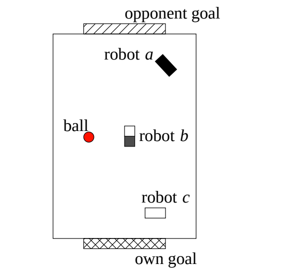
Each robot ^kR computes the module conflict and obtains the subsets $^k\mathcal{M}_a$, $^k\mathcal{M}_c$, $^k\mathcal{M}_{ca}$, and $^k\mathcal{M}_e$, separately.

4 Task and Assumptions

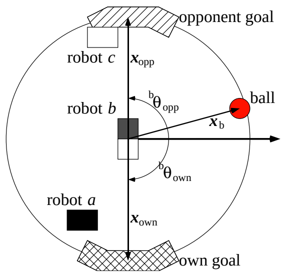
4.1 Environment and Robots

We apply the proposed method to a simplified soccer game including two mobile robots in the context of RoboCup [4]. RoboCup is an increasingly successful attempt to promote the full integration of AI and robotics research, and many researchers around the world have been attacking a wide range of research issues. Here, the multitask for the robots is to defend the own goal.

A mobile robot has an omnidirectional vision system. The robot moves around the field based on the



(a) top view of the field



(b) view captured by an omnidirectional vision

Figure 3: The experimental setting.

power wheeled steering system. As motor commands, each robot has two degrees of freedom. The environment consists of a ball and two goals. The sizes of the ball and the goals are the same as those of the middle-size real robot league in the RoboCup Competition.

4.2 Module description

Figure 3 (b) shows detected image features to extract the information of the environment, where $\boldsymbol{x}_b$, $\boldsymbol{x}_{own}$ and $\boldsymbol{x}_{opp}$ are the center positions of the ball, the own goal, and the opponent goal, respectively. The policy and the evaluation function are designed based on these features. Figure 4 shows typical behaviors generated by the four prepared modules. In order to realize the defending behavior, we design the following modules.

m_1**:** The purpose of this module is to push the ball.

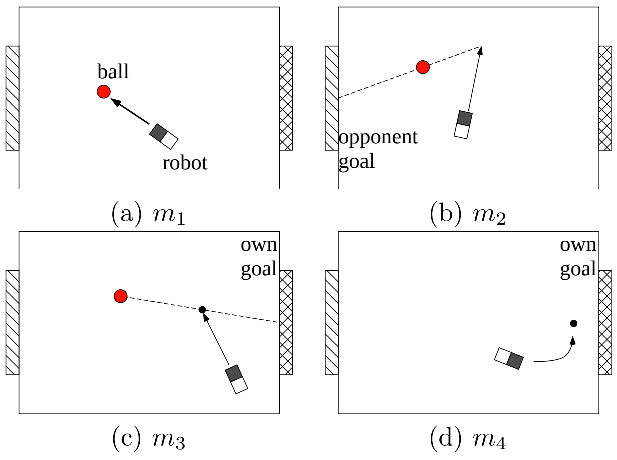
$$\begin{aligned} \boldsymbol{x}_d &= \boldsymbol{x}_b, \\ v &= ||\boldsymbol{x}_b||, \end{aligned}$$

where $\boldsymbol{x}_d$ and $||\cdot||$ denote the desired state and a norm of the vector, respectively. This module can be used when the robot attempts to shoot the ball into the goal or clear the ball.

m_2**:** The purpose of this module is to move to the opposite side of the opponent goal.

$$\begin{aligned} \boldsymbol{x}_d &= (b+1)\boldsymbol{x}_b - b\boldsymbol{x}_{opp} \quad (0 \le b \le 1), \\ v_r &= ||\boldsymbol{x}_b||, \\ v_\theta &= {}^b\theta_{opp}, \\ v &= wv_r + (1-w)v_\theta, \quad (0 \le w \le 1), \end{aligned}$$

This module can be used when the robot behave as a offensive player.



(a) m_1 (b) m_2 (c) m_3 (d) m_4

Figure 4: Typical behaviors generated by the modules

m_3**:** The purpose of this module is to move to the position between the ball and the own goal to save the own goal.

$$\begin{aligned} \boldsymbol{x}_d &= (1-a)\boldsymbol{x}_b + a\boldsymbol{x}_{own}, \\ v &= \pi/2 - |{}^b\theta_{own}|, \end{aligned}$$

m_4**:** The purpose of this module is to stay at the own goal.

$$\begin{aligned} \boldsymbol{x}_d &= \boldsymbol{x}_{own} \\ v &= ||\boldsymbol{x}_{own}||, \end{aligned}$$

The goalie should select this module in order to block the own goal with its own body.

In addition, we define the priority as $U(m_i) = i$ in this experiment. We prepare a controller which makes the features on the image plane converge to the desired values. For the desired state $\boldsymbol{x}_d = [x_d \, y_d]^T$, a motor command $\boldsymbol{u}$ is computed by

$$\boldsymbol{u} = \begin{bmatrix} u_r \\ u_l \end{bmatrix} = \begin{bmatrix} -1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_d \\ y_d \end{bmatrix}, \qquad (6)$$

where u_r and u_l are the velocities of the right and left wheels, respectively. All the policies of the modules are designed using Eq. (6). Although we implement the above *policies* using the servoing technique in this experiment, it is possible to acquire them based on the learning algorithms [9] as well.

5 Experimental Results

We show a result to demonstrate how the proposed method works. Figure 5 shows a configuration of the

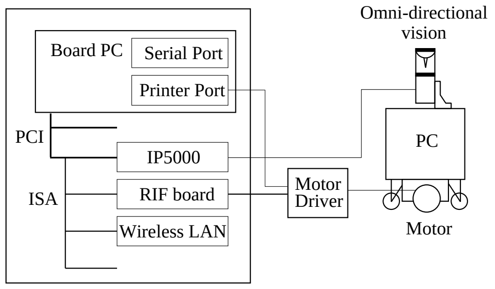


Figure 5: An overview of the robot system

real mobile robot. A simple color image processor (Hitachi IP5000) is applied to detect the ball and the goal area in the image in real-time (33 [msec]). Communication between the robots are realized using the Wireless LAN. Since the bandwidth of communication is very low, the robot can be controlled in real-time.

Figure 6 shows the averaged *task accomplishment* and *task execution* calculated by Eq. (3). From the beginning, m_4 is accomplished ($a_4 = 1$), while r0 seldom executes m_1. On the other hand, according to the situation, r1 pushed the ball toward the opponent goal using the module m_1.

Figure 7 shows an example sequence of the robots' behaviors. For the sake of convenience, the robot colored in black (white) is called r0 (r1). In Figure 7 (a), r0 is the robot in the right side. From these figures,

(a), (b) : Both robots moved to the own goal because the module m_4 (stay at the own goal) is the most important.

(c) : Since r0 accomplished the module m_4, r1 neglected m_4 and checked the *task accomplishment* of a_i $(i = 1, 2, 3)$.

(d) : r0 stayed in front of the own goal while r1 attempted to clear the ball.

(e) : They swapped their roles.

(f), (g) : r0 attempted to push the ball by m_1. In this case, m_2 was also accomplished by r0. On the other hand, r1 went to the own goal. However r1 failed to accomplish m_1 due to its poor policy.

(h) : r0 moved to the own goal again because r1 could not reach the own goal. As a result, r1 attempted to clear the ball again since r1 was not needed to stay at the own goal any more.

Because of the low image resolution, the robot sometimes failed to detect the objects at a long distance. In this case, the module could not be performed appropriately. In spite of those troubles, both robots were complementary to each other. The defense behavior could be accomplished based on our method.

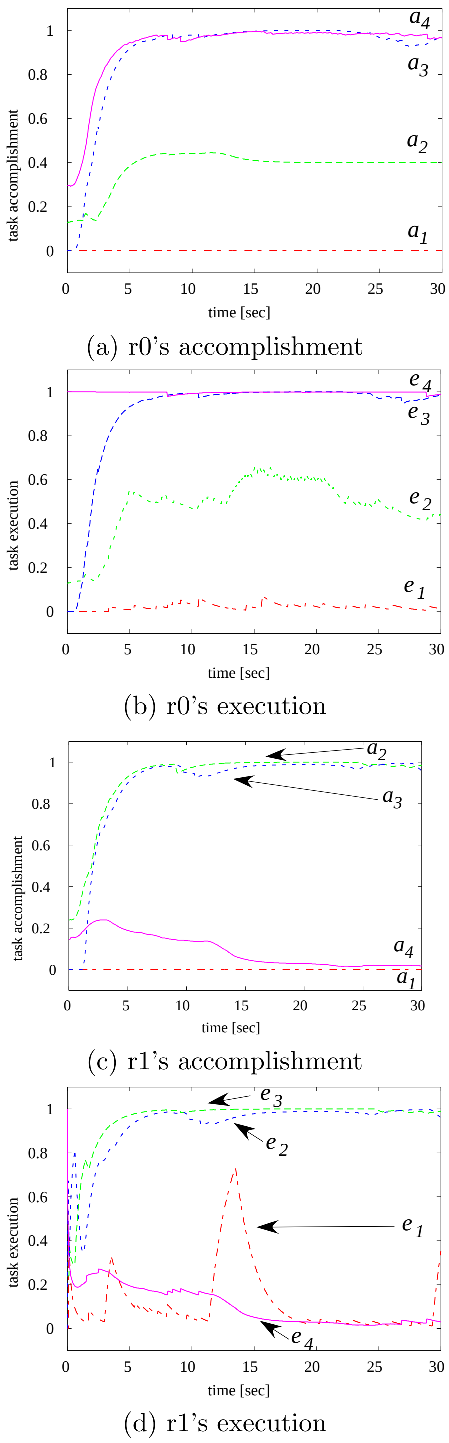


Figure 6: The averaged *task accomplishment* $\bar{a}_i$ and *task execution* $\bar{e}_i$

6 Conclusion

We have proposed a method to resolve the conflict between the given modules to handle the multiple tasks in a multiagent environment. We have applied the proposed method to several simplified soccer games, and show the validity of our method in a dynamically changing environment. We have already checked the simulation results in case of three robots, and we are planning to perform the experiments using three mobile robots in the real environment.

As future work there are following issues. One is a study about the priority function U because the total performance depends on the given priority function in the current system. Now, we are developing the method to change the priority function using the value of evaluation function. The other is an implementation of the proposed method without explicit communication since behavior understanding is also an important issue [6]. We have developed some algorithms to estimate the other's behavior from sequences of perception and action [1]. The integration of them leads the proposed method to be scaled to larger and more complicated situations.

Acknowledgment

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).

References

[1] M. Asada, E. Uchibe, and K. Hosoda. Cooperative behavior acquisition for mobile robots in dynamically changing real worlds via vision-based reinforcement learning and development. *Artificial Intelligence*, 110:275–292, 1999.

[2] C. Castelpietra et al. Communication and Coordination among heterogeneous Mid-size players: ART99. In *The Fourth International Workshop on RoboCup*, pages 149–158, 2000.

[3] M. S. Fontán and M. J. Matarić. Territorial multi-robot task division. *IEEE Transaction on Robotics and Automation*, 14(5):815–822, 1998.

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

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

[6] T. Matsuyama. Cooperative Distributed Vision – Dynamic Integration of Visual Perception, Action, and Communication –. In *Proc. of Image Understanding Workshop*, Monterey CA, 11 1998.

[7] L. E. Parker. Lifelong Adaptation in Heterogeneous Multi-Robot Teams: Response to Continual Variation in Individual Robot Performance. *Autonomous Robots*, 8:239–267, 2000.

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

[9] S. Suzuki, T. Kato, M. Asada, and K. Hosoda. 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.

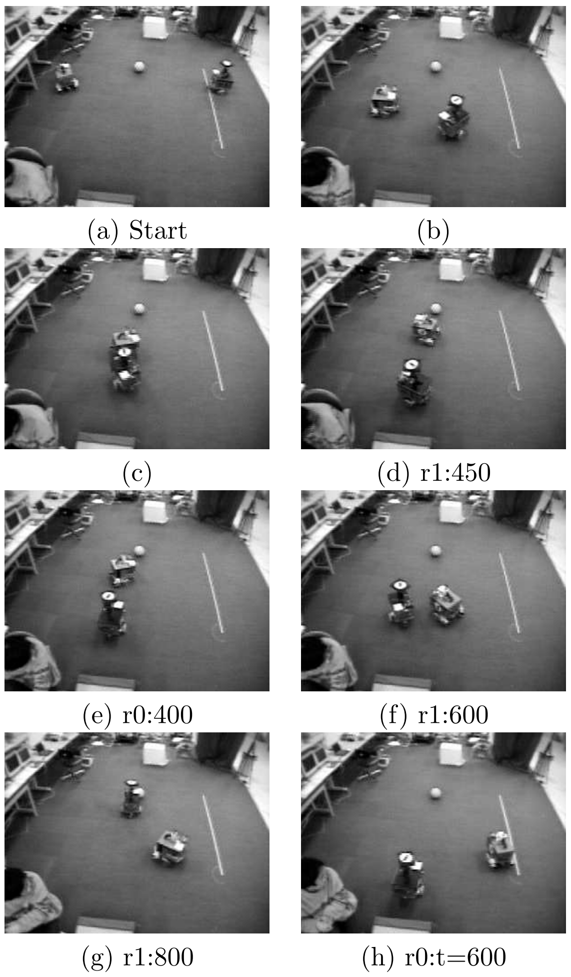


Figure 7: The result of real space experiment (robot ID:time)