Cooperatively Searching Objects Based on Mobile Agents

Takashi Nagata[1], Munehiro Takimoto[1], and Yasushi Kambayashi[2]

[1] Department of Information Sciences, Tokyo University of Science, Japan
[2] Department of Computer and Information Engineering, Nippon Institute of Technology, Japan

Abstract. This paper presents a framework for controlling multiple robots connected by communication networks. Instead of making multiple robots pursue several tasks simultaneously, the framework makes mobile software agents migrate from one robot to another to perform the tasks. Since mobile software agents can migrate to arbitrary robots by wireless communication networks, they can find the most suitably equipped and/or the most suitably located robots to perform their task. In this paper, we propose a multiple robot control approach based on mobile agents for searching targets as one of the effective examples. Though it is a simple task, it can be extended to any other more practial examples, or be used as an element of a real application because of its simplicity. We have conducted two kinds of experiments in order to demonstrate the effectiveness of our approach. One is an actual system with three real robots, and the other is a simulation system with a larger number of robots. The results of these experiments show that our approach achieves reducing the total time cost consumed by all robots while suppressing the energy consumption.

Keywords: Mobile agent, Dynamic software composition, Intelligent robot control.

1 Introduction

In the last decade, robot systems have made rapid progress not only in their behaviors but also in the way they are controlled. In particular, control systems based on multiple software agents have been playing important roles. Multi-agent systems introduced modularity, reconfigurability and extensibility to control systems, which had been traditionally monolithic. It has made easier the development of control systems on distributed environments such as multi-robot systems. On the other hand, excessive interactions among agents in the multi-agent system may cause problems in the multiple robot environments.

In order to mitigate the problems of excessive communication, mobile agent methodologies have been developed for distributed environments. In a mobile agent system, each agent can actively migrate from one site to another site. Since a mobile agent can bring the necessary functionalities with it and perform its tasks autonomously, it can reduce the necessity for interaction with other sites. In the minimal case, a mobile agent requires that the connection is established only when it performs migration [1].

The model of our system is a set of cooperative multiple mobile agents executing tasks for controlling a pool of multiple robots [2]. The property of inter-robot movement of the mobile agent contributes to the flexible and efficient use of the robot resources in addition to reducing the number of inter-communication as mentioned above. A mobile agent can migrate to the robot that is the most conveniently located to a given task, e.g. the closest robot to a physical object such as a soccer ball. Since the agent migration is much easier than the robot motion, the agent migration contributes to saving power consumption. Here, notice that any agents on a robot can be killed as soon as they finish their tasks. If the agent has a policy of choosing idle robots rather than busy ones in addition to the power-saving effect, it would result in more efficient use of robot resources.

In this paper, we show that the effectiveness of our control system in terms of resource usage and the efficiency of our system in terms of power consumption in two experimental setting. One setting consists of three physical robots cooperatively search targets [3] and the other setting consists of a larger number of virtual robots on a simulator also cooperatively search targets [4]. Since we implemented the simulator that reflects parameters given by the experiments with actual robots, we can expect the results of the simulation should be realistic enough. Although our example may look too simple, it could be extended to practical applications, or used as elements of them in a variety of applications because of its simplicity.

The structure of the balance of this paper is as follows. In the second section we describe the background. The third section describes the mobile agent class library that we have developed for controlling multiple robots. In our robot control system, the mobility that the software mobile agent system provides is the key feature that supports the ability of adding new functionalities to intelligent robots in action. The fourth section shows an example of intelligent robot systems in which robots search multiple target objects cooperatively. In the fifth section, we demonstrate how the properties of mobile agents can contribute to efficient use of robot resources through numerical experiments based on a simulator. Finally, we conclude in the sixth section.

2 Background

The traditional structure for the construction of intelligent robots is to make large, often monolithic, artificial intelligence software systems. The ALVINN autonomous driving system is one of the most successful such developments [5]. Putting intelligence into robots is, however, not an easy task. An intelligent robot that is able to work in the real world needs a large-scale knowledge base. The ALVINN system employs neural networks to acquire the knowledge semi-automatically [6]. One of the limitations of neural networks is that it assumes that the system is used in the same environment as that in which it was trained.

When the intelligent robot is expected to work in an unknown space or an extremely dynamic environment, it is not realistic to assume that the neural network is appropriately trained or can acquire additional knowledge with sufficient rapidity. Indeed, many intelligent robots lack a mechanism to adapt to a previously unknown environment.

On the other hand, multi-agent robotic systems are recently becoming popular in RoboCup or MIROSOT [7]. In traditional multi-agent systems, robots communicate with each other to achieve cooperative behaviors. The ALLIANCE architecture, developed in Oak Ridge National Laboratory, showed that cooperative intelligent systems could be achieved [8]. The architecture is, however, mainly designed to support self-adaptability. The robots in the system are expected to behave without external interference, and they show some intelligent behaviors. The observed intelligence, however, is limited due to the simple mechanism called *motivation*. Robots' behaviors are regulated by only two rules *robot impatience* and *robot acquiescence*. These rules are initially defined and do not evolve. In contrast, the goal of our system is to introduce intelligence and knowledge into the robots after they start to work [2]. Therefore, our system does not have any learning mechanism or knowledge acquiring mechanism. All the necessary knowledge is sent as mobile agents from other robots or the host computer.

An interesting research work of multi-agent robot control system was conducted at Tokyo University of Science [9]. Their work focused on the language aspect of robot control systems using multi-agents. They employed a hierarchical model of robot agents where the root agent indirectly manages all the agents. The lowest agents are physical devices and each has only one supervisor agent. Communications are performed through super-agent channels and subagent channels. Each robot has a hierarchically structured set of agents and the structure is rigidly constructed at the initial time. Therefore the structure of the control software is predetermined, and there is no concept of dynamic configuration of the structure of agents. The framework we present in this paper provides dynamic re-structuring of the set of agents and provides more flexibility in the real-world environments where any assumption cannot be expected.

For the communication aspect, they employ agent negotiation. In contrast, we employ agent migration so that our model can more suitably fit in a realistic multi-robot environment where the communication should be expected to be intermittent.

One notable feature of their system is the description language, called Multiagent Robot Language (MRL). This language is based on the committed-choice concurrent logic programming language and compiled into the guarded Horn clauses [10,11]. This feature has advantages of transparency of the agent descriptions over our framework that is based on Java. The efficiency problem of logic programming is overcome by recompiling into C language. We also implement a descriptive language based on functional languages, Objective Caml and Scheme, in order to achieve the transparency of the agent descriptions [12,13].

The work most closely related to ours is the distributed Port-Based Adaptable Agent Architecture developed at Carnegie Mellon University [14]. The Port-Based

Agents (PBAs) are mobile software modules that have input ports and output ports. All PBAs have the map of the port addresses so that they can move other robots and combine themselves with other PBAs to compose larger modules. The usefulness of PBA architecture is demonstrated by the Millibot project also at Carnegie Mellon University [15]. In a robot mapping application, PBA is used to control the mapping robots, and when the working robot has a hardware failure, the PBA on the robot detects it and moves to an idle robot.

Software composition is clearly possible using port-based modules. The dynamic extension capability of our mobile agent control system, however, is another strategy for the composition of larger software.

The PBA is derived from the concept of port-based objects, designed for real-time control applications [16]. Therefore it may have advantages as a robot control mechanism. The framework we present in this paper is an exploration of the applications of mobile agents and software compositions through mobility and extensibility. Constructing robot control software by mobile agents and its dynamic extension is not only novel but also flexible due to the migration which agents perform autonomously. It may be superior for extensibility of working software.

3 Mobile Agent Controlling Robots

We assume that a mobile agent system consists of mobile agents and places. Places provide runtime environments, through which mobile agents achieve migration from one environment to other environments. When a mobile agent migrates to another place, not only the program code of the agent but also the state of the agent can be transferred to the destination. Once an agent arrives at another place through migration, it can communicate with other mobile agents on that place.

The mobile agent system we have used to control robots is based on an existing mobile agent system, called AgentSpace, developed by I. Satoh [17]. AgentSpace provides the basic framework for mobile agents. It is built on the Java virtual machine, and agents are supposed to be programmed in Java language. The behaviors of an agents on AgentSpace are determined by the following methods:

create: is called when initializing the agent,
destroy: is called when killing the agent,
leave: is called when migrating to another site, and
arrive: is called when arriving at the new site.

These methods are call-back methods which are invoked by a place i.e. a runtime environment of AgentSpace. The `create` or `destroy` method is called when the user requires to create or to kill mobile agents through GUI. The `arrive` and `leave` method are called in the process of migration. All the behaviors of an agent are determined by programs described in these methods. In the case where an agent needs to communicate with other agents, first, the agent requires the context of the current place that the agent resides. The context includes information of other mobile agents on the place, and therefore through that context, any agents can extract information such as the reference to a specific agent through its own name from the context. The reference can be used to invoke the methods of other agents on the same place. The context also provides the functionality of agent migration. The agent migration is achieved through method *move*. The method *move* receives an instance of URL class with IP and port number as an argument, and *move* the agent that is referred by the context.

We have extended AgentSpace and developed an agent library *Robot* that includes methods as shown by Fig. 1. In order to implement the methods, we have taken advantage of primitives of ERSP. ERSP is a software development kit with high-level interfaces tailored for controlling robots. These interfaces provide several high-level means for control such as driving wheels, detecting objects through a camera, checking obstacles through supersonic sensors, and intelligent navigations. They are written in C++, while mobile agents are described as Java classes that extend `Agent` class of AgentSpace. Therefore, we have designed *Robot* library that uses these interfaces through JNI (Java Native Interface). The library *Robot* has interfaces that are supposed to be implemented for the following methods:

initialize initializes flags for inputs from a camera and sensors,
walk makes a robot move straight within required distance,
turn makes a robot turn within required degree,
setObjectEvent resets the flag for object recognition with a camera,
setObstacleEvent resets the flag for supersonic sensors,
getObject checks the flag for object recognition,
getObstacle checks the flag for the sensors, and
terminate halts all the behaviors of a robot.

4 Robot Controller Agents for Target Searcher

In this section, we demonstrate that our model, which is a set of cooperative multiple mobile agents, is an efficient way to control multiple robots. In our robot control system, each mobile agent plays a role in the software that controls one robot, and is responsible to complete its own task. One agent with one specific task migrates to one specific robot to perform that task. In this manner, an agent can achieve several tasks one by one through the migration to robots one by one. This scheme provides more idle time for each robot, and allows other agents to use the idle robots for incomplete tasks. In that way, this scheme contributes in decreasing the total time of completing all the tasks. We will show these advantages in the numerical experiments.

```
package robot;

public class Robot {

    static {
        System.loadLibrary("RobotStatic");
    }

    static public native void initialize();
    static public native void walk(double distance, double speed,
        int wait, double timeOut);
    static public void walk(double distance, double speed, double timeOut) {
        walk(distance, speed, 500, timeOut);
    }
    static public native void turn(double angle, double speed, double timeOut);
    static public native void setObstacleEvent(int dir, double threshold);
    static public native void setObjectEvent(int objId, double threshold);
    static public native int getObstacle();
    static public native int getObject();
    static public native void terminate();
}
```

Fig. 1. Class library *Robot*

4.1 Controlling Robots

An intelligent multi-robot system is expected to work in a distributed environment where communication is relatively unstable and therefore where fully remote control is hard to achieve. Also we cannot expect that we know everything in the environment beforehand. Therefore intelligent robot control software needs to have the following features: 1) It should be autonomous to some extent. 2) It should be extensible to accommodate the working environment. 3) It should be replaceable while in action. Our mobile agents satisfy all these functional requirements.

Our control software consists of autonomous mobile agents. Once an agent migrates to a remote site, it requires minimal communication to the original site. Mobile agents can communicate with other agents on the same place so that the user can construct a larger system by migration to the place. The newly arrived agent can communicate with agents that reside in the system before its arrival, and achieve new functionality with them. If we find that the constructed software is not good enough to satisfy our requirements in a remote environment, we can replace the unsuitable component (an agent) with new component (an- other agent) by using agent migrations.

In the first experiment, we employed three wheeled mobile robots, which are called PIONEER 3-DX, as the platform for our prototype system. Each robot has two servo-motors with tires, one camera and sixteen sonic sensors. The power is supplied by rechargeable battery. Fig. 2 shows the team of robots in action for searching targets.

In the second experiment, we constructed a realistic simulator following information we have extracted from the observation of the behaviors of PIONEER 3-DX to show the scalability of our control system.

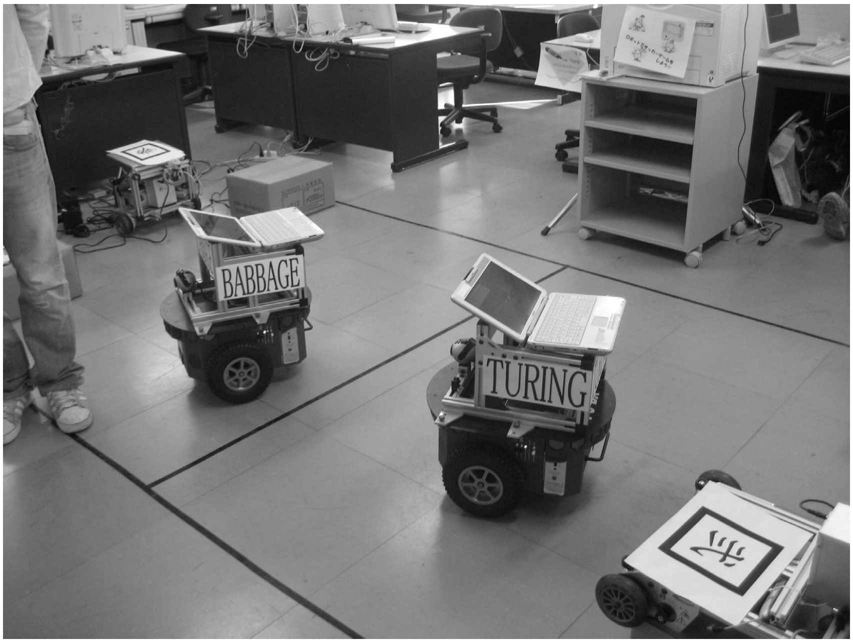


Fig. 2. A team of mobile robots are working under control of mobile agents

4.2 Searching a Target

Let us consider how to program a team of multiple robots to find a target. For such a task, the most straightforward solution would be to make all robots search for the target simultaneously. If the targets were comparatively fewer than the robots, however, most robots would move around in vain, consuming power without finding anything.

This problem can be more serious in our model where any robots can be shared by any agents, because the robots to which an agent with a new task is going to migrate may be already occupied by another agent with some different task. Especially, consider a case where the robots are working in an area where communications on wireless LAN are difficult. In such a case, even if one of the working robots finds the target, the other robot may not be able to know that fact. As a result, most robots continue to work to search that target in vain until time-out. Thus, this searching strategy could not only wastes the power but also increase the total costs of the multiple robots in aggregate. On the other hand, our strategy of using mobile agents achieves the suppression of the total cost due to the efficient use of idle resources as well as saving power consumption.

The core of our idea is finding the nearest robot to the target by using agent migration. Initially, an agent is dispatched from the host machine to a nearby

```
package agent.search;

import agentspace.*
import robot.Robot;

public class Search implements Agent, Mobile, Duplicatable, Preservable {

    public void arrive(AgentEvent evt, Context context) {
        //get ids of agents on the current place.
        AgentIdentifier[] aids = context.getAgents();

        if(aids.length == 1) {
            // If there is just itself on the current place,
            //execute  behavior() as the main behavior.
            behavior(context);
        }
        else {
            // If there are other agents on the current place,
            // it migrates to other robot.
            try {
                context.move(otherAddress());
            } catch(Exception e) { }
        }
    }


    public static void behavior(Context context) {
        while(true) {
            // It makes the robot rotate within 360 degrees.
            // If it finds a target, stop rotation and set the flag for detecting.
            Robot.turn(360.0, 5.0, 5.0);

            if (Robot.getObject().equals("TARGET1")) {
                // If detected thing is the target, it makes the robot go straight.
                Robot.walk(40.0, 3, 7.0);
            }
            else if (isExhausted()) {
                // If it has arrived at the last robot without finding anyting,
                // it makes the robot walk randomly.
                randomWalk();
            }
            else {
                // Otherwise, migrates to other robot.
                context.move (otherAddress());
            }
        }
    }
}
```

Fig. 3. *arrive()* method and *behavior()* method robot in the multi-robots system. Then, the agent hops among the robots one by one and checks the robot's vision in order to locate the target until it reaches the robot that is the closest to the target. Until this stage, robots in the multi-robot system do not move; only the mobile agent migrates around so that robots can save power.

Once the agent finds the target, it migrates to the closest robot and makes the robot move toward the target. In our strategy, since only one robot dedicates to a particular task at a time, it is essentially similar to making each robot special for each task. Since the migration time is negligible compared to robot motion, our strategy is more efficient than such as we described before. If the agent visits all the robots without finding the target, the agent makes the last one move around randomly with wheels in order to find the target.

In our current multi-robot system, the robots' vision does not cover 360 degrees. Therefore a robot has to rotate to check its circumstance. Rotating a robot at its current position may capture the target and another robot closer to the target. Then the agent migrates to that more conveniently located robot. Notice that the rotation requires much less movement of the wheels than exploration, and it contributes to the power saving.

Details of our searching algorithm are the followings: 1) an agent chooses an arbitrary robot to which it migrates, and performs the migration, 2) as the agent arrives on the robot, it makes that robot rotate,where if the robot to which the agent migrates has been occupied by another agent, it migrates to another robot further,3) if the target is found, the agent makes the robot move to that direction; otherwise, goes back to step 1, and 4) at this stage, if all robots have been tried without finding the target, the agent makes the last robot do random-walk until it can find a target.

We have implemented this algorithm as an AgentSpace agent *search* as shown by Fig. 3. As soon as *search* agent has migrated to a robot, its *arrive()* method is invoked. *Arrive()* checks whether there are any other agents on the current robot or not. That can be achieved by checking agents' ids. This is achieved by calling *context.getAgents()*. If it finds only its own agent id, it means that there are no other agents. If it finds no other agents, it invokes *behavior()* as the main behavior of the robot. If it finds another agent id, it means the robot is occupied by the other agent, and the newly arrived agent immediately migrates to another robot in order to avoid interferences with the other agents.

The method *behavior()* first makes the robot rotate within 360 degrees to look around its circumstance. If it finds something, it stops the rotation of the robot, and sets the flag that indicates it detects an object. At this time, what is found can be checked through *Robot.getObject()*. As a result, if the return value corresponds to "TARGET1", it makes the robot go straight through *Robot.walk()*. Otherwise, it checks whether it has visited all the robots through *isExhausted()*. If there is no more robots to visit, it invokes *randomWalk()*, and makes the robot walk randomly in order to find the target at different angles. Otherwise, it migrates to one of the other robots.

5 Experimental Results

In order to demonstrate the effectiveness of our system, we have conducted numerical experiments on the example of target search that is mentioned in the previous section. In the experiments, we set a condition where robots search several targets, where searching each target corresponds to distinct task.

We have compared our approach based on mobile agents with other two strategies as follows:

EachForEach: allocates specific target to each robot. Each robot searches its own target, and does not search any other things as shown by Fig. 4(a), and

AllForEach: makes all robots move around for each target as shown by Fig. 4(b) which is the snapshot for searching a target.

The EachForEach approach is the simplest method. If there is no way to replace the program on a robot in action, this method would be used as the most realistic solution.

The AllForEach approach seems to consume the least time in order to search one target. In some case, it may also be the most efficient method for searching several targets. However, this approach may consume a lot of time in the area where the condition of connections among robots is changeable, because even if one robot finds a target, other robots may not be able to know the fact. In such a case, the task for searching one target might exhaust the time allowed for the entire team of robots. In our mobile agent based approach (Fig. 4(c)), a mobile agent is fixed to one robot at a time, and only one robot with the mobile agent searches one particular target until robots to which it migrates are exhausted. Eventually, the behavior becomes the same as the EachForEach approach.

5.1 Three Robots Experiment

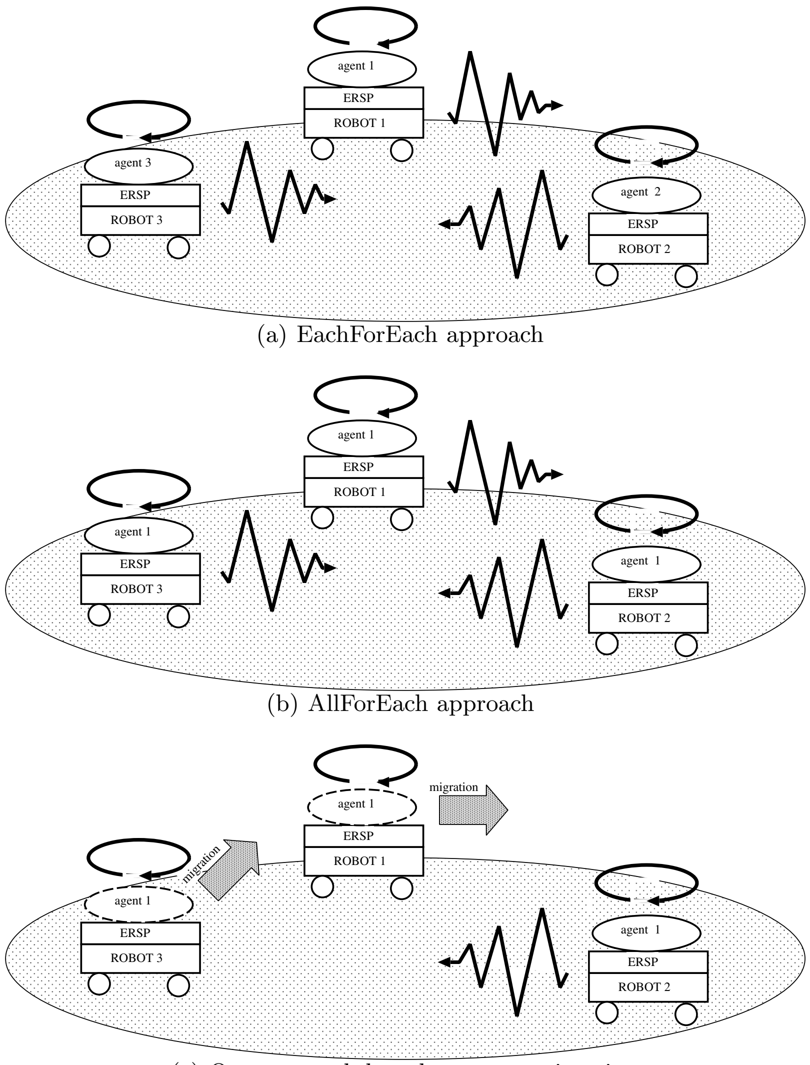
First, in order to show that our approach is physically effective, we conducted experiments based on actual three robots in $3.5m \times 5.0m$ rectangle area which is the largest possible area where at least two of three robots have overlapping views.

We deal with the case where every target is the close to the different robot. In order to simulate realistic situations, we set several variations for some approach as follows:

For the EachForEach: a) robots for target A, B and C are respectively close to A, B and C , b) only robot for target A is close to A, and the other ones are close to different targets, and c) every robot is close to different target.

For the AllForEach: making all robots search one target at a time, and repeat for three targets. There is no other variation.

For mobile agent based approach: a) agents for target A, B and C initially migrate to robots close to A, B and C, b) some agents migrate one time after initial migration, and c) some agents migrate twice.

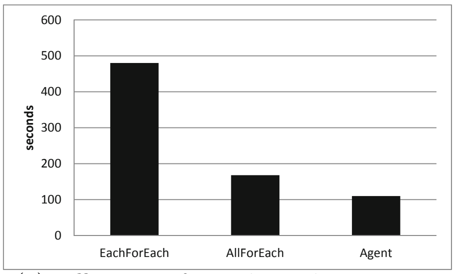


(a) EachForEach approach

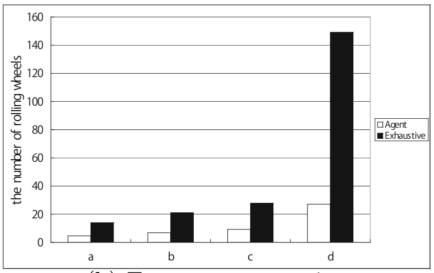
(b) AllForEach approach

(c) Our approach based on agent migration

Fig. 4. Experiments



(a) Efficiency of searching three targets



(b) Energy consumption

Fig. 5. Time cost and energy consumption based on phisical robots

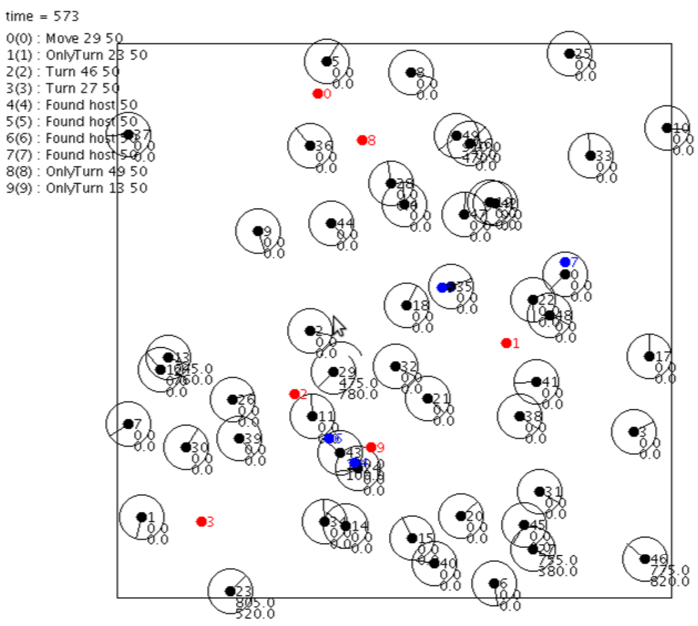
Fig. 5(a) shows the results of the experiments on efficiency. This graph shows how long the multiple robots take to find all the targets in seconds as the average of all settings for each approach. As shown by the figure, we can observe that our approach is the most efficient. We can reason that our approach makes mobile agents occupy robot resources not so long as the other approaches do, and produces more idle resources. Therefore the idle resources are effectively reused through migration of other new agents with other tasks.

In this experiment, we dealt with the ideal case that does not cause random-walk for our approach. In general, such cases would not occur so often, and our approach also has to randomly walk as soon as the robot to which the agent migrates are exhausted. As mentioned before, the behavior of our approach after visiting all the robots is the same as the EachForEach approach, so our approach has an obvious advantage against the EachForEach approach.

On the other hand, the AllForEach approach may be more efficient in some cases. Consider that there are fewer targets than robots that search them. In this case, some robots successfully find the targets, but the other ones would move around in vain. That results in wastefully consuming energy. Fig. 5(b) shows the comparison of the times of rotating wheels for the AllForEach approach and our approach in the extreme case where three robots search one target. Each result shows the time in the different settings as follows:

Pattern a: the target is near the robot to which the *search* agent migrates first,
Pattern b: the target is near the robot to which the agent migrates second,
Pattern c: the target is near the robot to which the agent migrates last, and
Pattern d: the target is far from any robots.

It is reasonable to assume that energy consumption of servo-motors is linear to the wheel rotation times. It is clearly observed that, in all the settings, our approach consumes less energy than the AllForEach approach. Furthermore, as mentioned before, the AllForEach approach can waste plenty of time in the case where the condition of connections among robots is changeable.



```
time = 573
0(0) : Move 29 50
1(1) : OnlyTurn 23 50
2(2) : Turn 46 50
3(3) : Turn 27 50
4(4) : Found host 50
5(5) : Found host 50
6(6) : Found host 50
7(7) : Found host 50
8(8) : OnlyTurn 49 50
9(9) : OnlyTurn 13 50
```

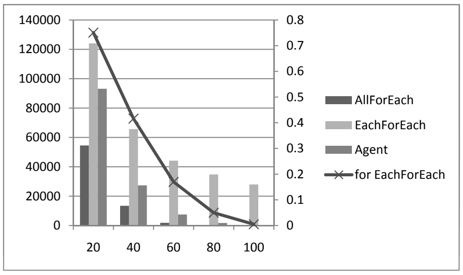
Fig. 6. A snapshot of a running simulator

5.2 Large Scale Simulation

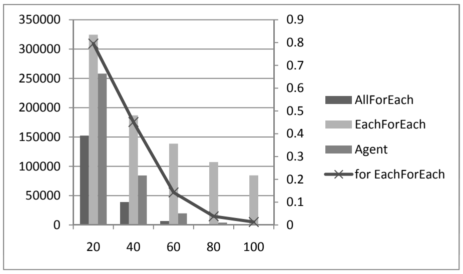
Second, in order to demonstrate the effectiveness of our system in a large scale environment, we have implemented a simulator of the target search based on the real multi-robot system described in the previous section, and have conducted numerical experiments on it. On the simulator, moving and rotating speed of robots, and lags required in agent migration and object recognition are based on real values obtained in the previous experiments using PIONEER 3-DX with ERSP [3]. In the experiments, we set a condition where fifty robots are scattered in a 500 × 500 square field in the simulator, where searching each target corresponds to a distinct task. We have compared our approach based on mobile agents with other two strategies, AllForEach and EachForEach as well as the experiments with actual robots.

We have recorded the total moving distance and the total time of the robots that perform all the strategies. We have evaluated the results by changing the two parameters. They are the number of targets and the width of a view. The view means a circle with a robot as a center, and the robot can detect any objects in the circle as shown in Fig. 6. It is reasonable to assume that energy consumption of servomotors is linear to moving distance.

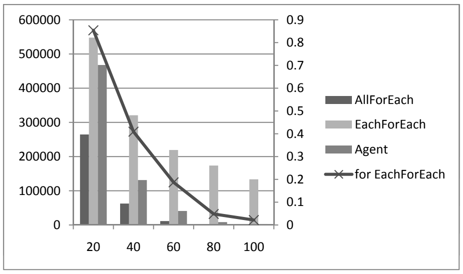
Bar charts of Fig. 7(a)–(d) show each of the total moving distances for 10, 30, 50, 70, and 90 targets respectively. The AllForEach strategy seems to achieve the least energy consumption over any number of targets. In some cases, it shows



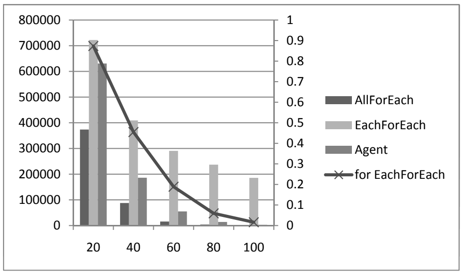
(a) Results for 10 targets



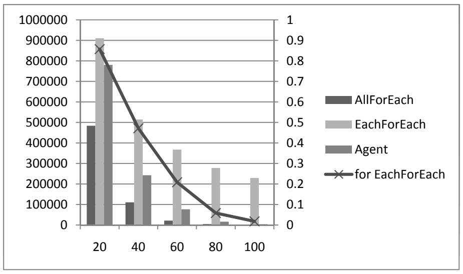
(b) Results for 30 targets



(c) Results for 50 targets



(d) Results for 70 targets



(e) Results for 90 targets

Fig. 7. The total moving distances for the view width: 20, 40, 60, 80, and 100

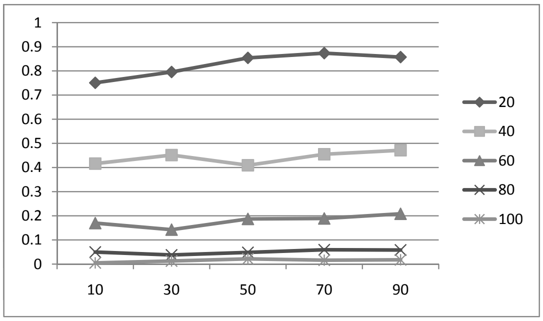


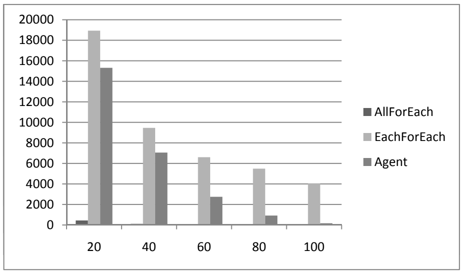
Fig. 8. The total moving distance over the numbers of targets

itself as the most efficient method for searching multiple targets. This approach, however, may consume a lot of energy when the condition of connections among robots is intermittent. Even though one robot finds a target, other robots may not be able to know the fact. In such a case, the task for searching one target might consume all the allowances for the entire team of robots. In our mobile agent based approach, on the other hand, a mobile agent is fixed on a certain robot after its migration, and as soon as the robot achieves the task, the entire multi-robot system turns to pursue the next object. As a result, the behavior of entire multi-robot system becomes effectively similar to the EachForEach strategy but it performs the same task just more efficiently.

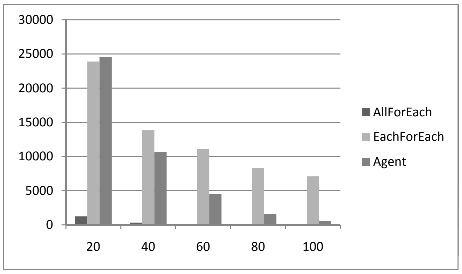
The mobile agent system displays a remarkable saving of energy consumption compared to the EachForEach strategy. Furthermore, the more the width of a view increases, the more efficiency Agent gains than EachForEach, as shown by the line chart representing the ratio of the Agent to the EachForEach in Fig. 7(a)–(d).

Meanwhile, Fig. 8 shows the ratio of the total moving distance of the Agent strategy to the EachForEach strategy for each view width over the various numbers of targets. The flat lines illustrate the constant advantage of the Agent strategy over the EachForEach strategy regardless of the number of targets.

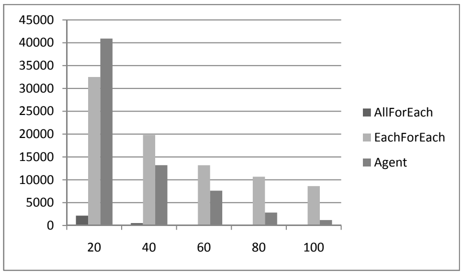
Fig. 9(a)–(d) shows the total time for searching out all the 10, 30, 50, 70, and 90 targets respectively. Since the total duration time is proportional to the total moving distance, the AllForEach strategy seems to be the most efficient among the three strategies. But it is not practical as mentioned above. On the other hand, the Agent strategy makes mobile agents occupy robot resources not so long as the other approaches do, and produces more idle resources. Then other new agents with other tasks can effectively use the idle resources through migration. We, however, observe that the Agent strategy shows less efficiency than that of the EachForEach where the width of a view is 20 shown in Fig. 9(b)-(d). In such cases, the Agent strategy often fails to find any target during the migration step due to the restricted view, and causes robots to randomly walk. We can conclude that the wider the view becomes, the more efficiently Agent works.



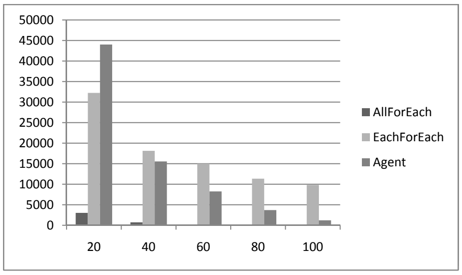
(a) Results for 10 targets



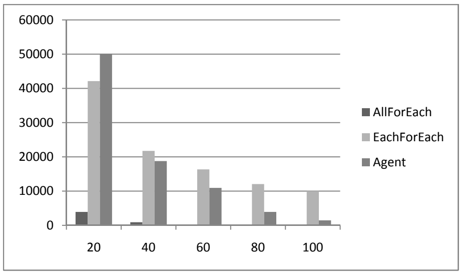
(b) Results for 30 targets



(c) Results for 50 targets



(d) Results for 70 targets



(e) Results for 90 targets

Fig. 9. The total time for each the view width: 20, 40, 60, 80, and 100

Thus, we believe that our approach is practical enough for controlling multiple robots in the real world in terms of the total cost and energy consumption.

6 Conclusions

We have presented a novel framework for controlling intelligent multiple robots. The framework helps users to construct intelligent robot control software by migration of mobile agents. Since the migrating agents can dynamically change the functionalities of the robot which they control, the control software can be flexibly assembled while they are running. Such a dynamically extending the control software by the migration of mobile agents enables us to make the base control software relatively simple, and to add functionalities one by one as we know the working environment. Thus we do not have to make the intelligent robot smart from the beginning or make the robot learn by itself. We can send intelligence later as new agents.

We have conducted experiments using three real robots. Through the experiments, we have successfully shown that our framework for controlling multiple robots can reduce energy consumption under the realistic circumstances. We also implemented a simulator that simulates the behaviors of a large scale team of cooperative search robots to show the effectiveness of our framework, and demonstrated that our framework contributes to suppressing the total cost of a multi-robot system in large scale cases. The numerical experiments show the volume of saved energy is significant. They demonstrate the superiority of our approach over more traditional non-agent based approaches.

Our future directions for research will include the addition of security features, refinement of the implementation of dynamic extension, additional proof of concept for dynamic addition of new functionality, and additional work on scheduling of conflicting tasks.

Acknowledgement. This work is supported in part by Japan Society for Promotion of Science (JSPS), with the basic research program (C) (No. 20510141), Grant-in-Aid for Scientific Research.

References

1. Binder, W.J., Hulaas, G., Villazon, A.: Portable resource control in the j-seal2 mobile agent system. In: Proceedings of International Conference on Autonomous Agents, pp. 222–223 (2001)
2. Kambayashi, Y., Takimoto, M.: Higher-order mobile agents for controlling intelligent robots. International Journal of Intelligent Information Technologies 1(2), 28–42 (2005)
3. Nagata, T., Takimoto, M., Kambayashi, Y.: Suppressing the total costs of executing tasks using mobile agents. In: Proceedings of the 42nd Hawaii International Conference on System Sciences. IEEE Computer Society CD-ROM (2009)

4. Abe, T., Takimoto, M., Kambayashi, Y.: Searching targets using mobile agents in a large scale multi-robot environment. In: O'Shea, J., Nguyen, N.T., Crockett, K., Howlett, R.J., Jain, L.C. (eds.) KES-AMSTA 2011. LNCS, vol. 6682, pp. 211–220. Springer, Heidelberg (2011)
5. Pomerleau, D.: Defense and civilian applications of the alvinn robot driving system. In: Proceedings of 1994 Government Microcircuit Applications Conference, pp. 358–362 (1994)
6. Pomerleau, D.: Alvinn: An autonomous land vehicle in a neural network. In: Advances in Neural Information Processing System 1, pp. 305–313. Morgan Kaufmann (1989)
7. Murphy, R.: Introduction to AI robotics. MIT Press, Cambridge (2000)
8. Parker, L.: Aliance: An architecture for fault tolerant multirobot cooperation. IEEE Transaction on Robotics and Automation 14(2), 220–240 (1998)
9. Nishiyama, H., Ohwada, H., Mizoguchi, F.: A multiagent robot language for communication and concurrency control. In: Proceedings of International Conference on Multi-Agent Systems, pp. 206–213 (1998)
10. Shapiro, E.: Concurrent Prolog: Collected Papers. MIT Press, Cambridge (1987)
11. Ueda, K.: Guarded Horn Clauses. PhD Thesis, University of Tokyo (1986)
12. Kambayashi, Y., Takimoto, M.: A functional language for mobile agents with dynamic extension. In: Negoita, M.G., Howlett, R.J., Jain, L.C. (eds.) KES 2004. LNCS (LNAI), vol. 3214, pp. 1010–1017. Springer, Heidelberg (2004)
13. Kambayashi, Y., Takimoto, M.: Scheme implementation of the functional language for mobile agents with dynamic extension. In: Proceedings of IEEE International Conference on Intelligent Engineering Systems, pp. 151–156 (2005)
14. Pham, T., Dixon, K.R., Jackson, J., Khosla, P.: Software systems facilitating self-adaptive control software. In: Proceedings of IEEE International Conference on Intelligent Robots and Systems, pp. 1094–1100 (2000)
15. Grabowski, R., Navarro-Serment, L., Paredis, C., Khosla, P.: Heterogeneous teams of modular robots for mapping and exploration. Autonomous Robots 8(3), 293–308 (2000)
16. Stewart, D., Khosla, P.: The chimera methodology: Designing dynamically reconfigurable and reusable real-time software using port-based objects. International Journal of Software Engineering and Knowledge Engineering 6(2), 249–277 (1996)
17. Satoh, I.: A mobile agent-based framework for active networks. In: Proceedings of IEEE System, Man and Cybernetics Conference, pp. 71–76 (1999)