Searching Targets Using Mobile Agents in a Large Scale Multi-robot Environment

Tsukasa Abe[1], Munehiro Takimoto[1], and Yasushi Kambayashi[3]

[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 the previous papers, we have shown that this manner of controlling multiple robots can decrease the number of required robot resources in the three-robot case. The three robots demonstrated that they could suppress the energy consumption through a mobile agent based approach. In this paper, we pursue a case of large scale multiple robots. We have implemented a simulator for a large number of robots based on our framework, and have demonstrated the efficiency and the scalability. We report the observations we found in the numerical experiments.

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, a control system based on multiple software agents can control robots efficiently. 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,

J. O'Shea et al. (Eds.): KES-AMSTA 2011, LNAI 6682, pp. 211–220, 2011.

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 by 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. A mobile agent can migrate to the robot that is most conveniently located to a given task, e.g. 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.

We have proposed our model in the previous paper [3,4], and have also shown the effectiveness of saving power consumption and the efficiency of our system in a case of three physical robots [4]. In this paper, we focus our attention on the aggregate behaviors of a large number of robots based on our framework in order to investigate properties of our framework for practical use. We show the results through the numerical experiments on a simulator.

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 of 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. 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.

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 [9]. 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. We have extended AgentSpace and developed an agent library for controlling mobile robots.

We have implemented an agent library *Robot* as an extension of AgentSpace that includes methods for controlling robots. 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. 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.

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.

In order to construct a realistic simulator, we have extracted information by observing the behaviors of three wheeled mobile robots, called PIONEER 3-DX. Each robot has two servo-motors with wheels, one camera and sixteen sonic sensors. The power is supplied by a rechargeable battery. Each robot holds one notebook computer as its host computer. Our control agents migrate to these host computers by wireless LAN.

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 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 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 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 the more conveniently located robot. Take note 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 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 the number of agents' id's. 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. Otherwise, it 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

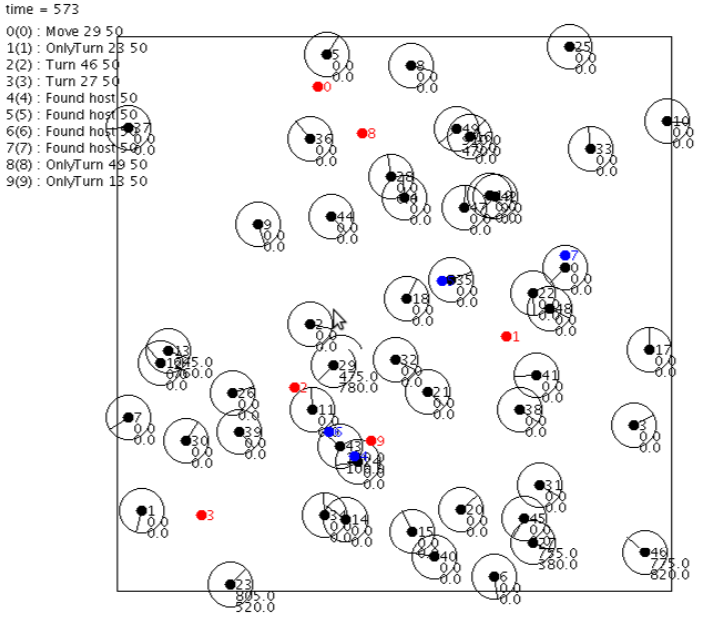


Fig. 1. A snapshot of a running simulator

robot walk randomly in order to find the target at different angles. Otherwise, it migrates to one of the other robots.

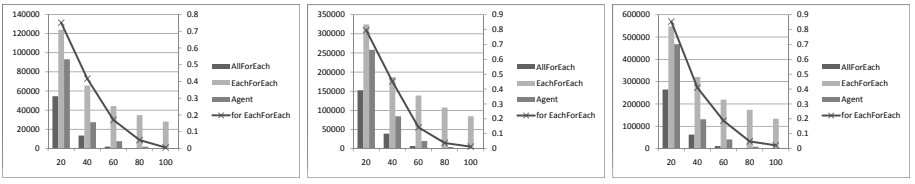
5 Experimental Results

In order to demonstrate the effectiveness of our system in a realistic 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 in the previous experiments using PIONEER 3-DX with ERSP [4]. 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 explained respectively as follows:

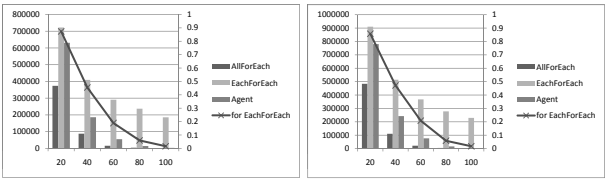
AllForEach: makes all robots move around for each target one by one, and

EachForEach: allocates specific targets to each robot. Each robot searches its own targets, and does not do any other tasks.

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



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

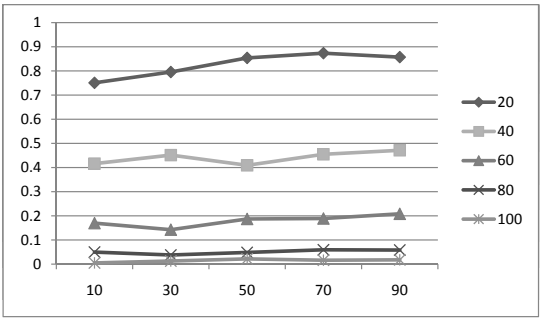
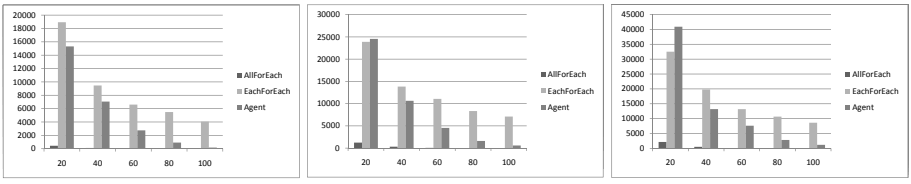


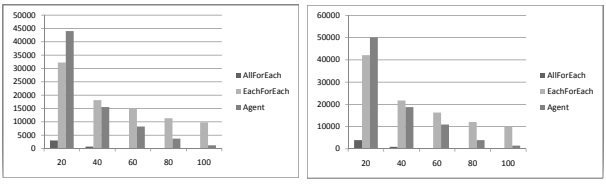
Fig. 3. The total moving distance over the numbers of targets

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. 1. It is reasonable to assume that energy consumption of servomotors is linear to moving distance.

Bar charts of Fig.2(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 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



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

not be able to know the fact. In that 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 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.2(a)–(d).

Meanwhile, Fig. 3 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.4(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.4(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.

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

6 Conclusions

We have presented a novel framework for controlling intelligent multi-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 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. 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. Hulaas, G., Binder, A.V.W.J.: 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. Takimoto, M., Mizuno, M., Kurio, M., Kambayashi, Y.: Saving energy consumption of multi-robots using higher-order mobile agents. In: Nguyen, N.T., Grzech, A., Howlett, R.J., Jain, L.C. (eds.) KES-AMSTA 2007. LNCS (LNAI), vol. 4496, pp. 549–558. Springer, Heidelberg (2007)
4. 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, Los Alamitos (2009)
5. Pomerleau, D.A.: 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.A.: Alvinn: An autonomous land vehicle in a neural network. In: Advances in Neural Information Processing System, vol. 1, pp. 305–313. Morgan Kaufmann, San Francisco (1989)
7. Murphy, R.R.: Introduction to AI robotics. MIT Press, Cambridge (2000)
8. Parker, L.E.: Aliance: An architecture for fault tolerant multirobot cooperation. IEEE Transaction on Robotics and Automation 14(2), 220–240 (1998)
9. Satoh, I.: A mobile agent-based framework for active networks. In: Proceedings of IEEE System, Man and Cybernetics Conference, pp. 71–76 (1999)