Development of Agent System for Multi-robot Search

Masashi Omiya[1], Munehiro Takimoto[2] and Yasushi Kambayashi[1]
[1]*Department of Computer and Information Engineering, Nippon Institute of Technology, 4-1 Gakuendai, Miyashiro-machi, Minamisaitama-gun, Saitama 345-8510, Japan*
[2]*Department of Information Sciences, Tokyo University of Science, 2641 Yamazaki, Noda 278-8510, Japan*

Keywords: Mobile Agent, Ad-hoc Network, Multi Robot, Particle Swarm Optimization.

Abstract: In this paper, we propose an agent system that controls multiple mobile robots. We describe the agent system as well as an example multi-robot system as an application of this agent system. The aim of the multi-robot system is providing efficient searches for a given target. Therefore, it is necessary to mutually communicate and cooperate among robots. Taking account of the delay of communication, cost, fault tolerance, and robustness, we have chosen mobile agent system for the information transmission method. We have also chosen ad-hoc method for the communication mode, where each robot communicates directly without going through the Internet. Even though agent systems are often implemented using Java language or Python language, we have chosen C++ language for developing our agent system with the hope for gaining performance efficiency. We have modularized each function and it does not depend on other modules. We have shown the feasibility of our multi-agent system by applying the system to a multi-robot system that implement particle swarm optimization.

1 INTRODUCTION

Recently we are observing scenes where robots play an active part in places where people can hardly work. For example, multiple autonomous robots may be used for search and rescue operations in urban areas after a disaster. In the previous study, we have shown that a team of multiple robots search efficiently using mobile agents (Oda et al., 2018).

A mobile agent is a program that migrates between devices and performs a given task at an arbitrary target device (Kambayashi and Takimoto, 2005). The agent platform need to transmit and receive agents between devices. When communicating via an access point that controls the network, the access point allocates IP addresses to each devices. In the previous study, we have specified the destination of agent using the assigned IP address (Oda et al., 2018).

Considering after disaster cases, disconnection may occur frequently between the access point and the communication device due to collapse of the communication infrastructures. Therefore, communication through the access points becomes hard to achieve.

On the other hand, in ad-hoc mode which devices communicate directly, the network can be constructed dynamically (Nishiyama et al., 2014). Since a mobile agent is a partially self-sufficient program, it does not need to keep continuous communication. They can continue to work with intermittent communication, since they only require connection when they move (Kambayashi et al., 2016).

We have three objects in this study; 1) employing ad-hoc mode for the agent transition method, 2) developing a new agent system implemented in C++, and 3) realizing a search using multiple mobile robots. We have prepared a specific class to determine the destination of a given agent. As the example experiment, we have implemented a search simulation using the particle swarm optimization.

The rest of this paper is organized as follows. The second section describes the background. The third section describes the agent system in C++ we have designed and implemented. The fourth section demonstrates the applicability of our agent system to an example multi-robot system. We show the feasibility of our mobile agent system by discussing an experiment. Finally, the fifth section concludes our discussion.

Omiya, M., Takimoto, M. and Kambayashi, Y.
Development of Agent System for Multi-robot Search.
DOI: 10.5220/0007702603150320
In *Proceedings of the 11th International Conference on Agents and Artificial Intelligence (ICAART 2019)*, pages 315-320
ISBN: 978-989-758-350-6

2 BACKGROUND

Agent systems have been widely developed in various languages (Stone and Veloso, 2000; Braun and Rossak, 2005). AgentSpace is the prominent agent system (Sato, 1997). AgentSpace was developed in Java language. In the previous study, we developed a mobile agent platform in Python language (Oda et al., 2018). The purpose of selecting C ++ language as a development language in this paper is to improve the agent system's processing speed. We have provided C++ classes that provide basic functions for constructing mobile agents. In this agent system, the users are expected to overwrite functions to construct their own agents. Since our agent platform only sends executable programs as files, agents can be written in arbitrary programming languages such as Java and Python language.

We have conducted projects that have taken advantage of multiple mobile agents. These projects include evacuation route generation (Taga et al., 2016; Taga et al., 2017; Taga et al., 2018), and rescue and search operation (Nagata et al., 2013). These two studies have one commonality, namely, building a multi-agent system in a distributed environment. As a result of these studies, we have shown that an efficient search can be achieved through communication among multiple mobile agents.

3 AGENT SYSTEM

In this agent system, an agent is transmitted as an executable file, and local data are transmitted as separate files. We have designed the system extensively using polymorphism through the C++ inheritance mechanism. Since it is possible to override member functions from derived classes, it is relatively easy to change depending on the purpose. We confirmed the operation of communication in ad-hoc mode environment and the communication in multi-hop between multiple mobile robots are both possible.

3.1 Configuration

As shown in Figure 1, the agent system consists of Agent class and Server class.
The Agent class inherits three classes: Execute class, Transmit class, and Routing class. The users are expected to implement their own class, i.e. the user agent as a subclass of the Agent class, and to take advantages of these classes through multiple inher-

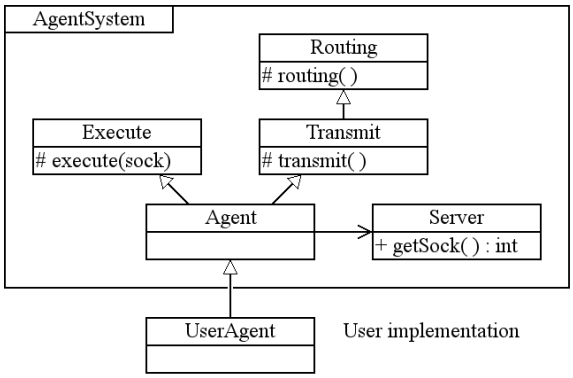

Figure 1: Class relation.

itance. We designed our agent system to implement only one functionality in each class so that it provides as much flexibility and simplicity of the code as possible. The Agent object sends and receives user agents from another device. The Server object establishes the connection for transmission of agents. We explain each class in the following sections.

3.1.1 Server

This class receives the request of agent migration that is transmitted from other devices, and manages a socket connection to other devices. The Server object runs continuously in an independent thread. From now, we call the Server object, the server.

As shown in Figure 2, the server thread receives the request of an agent migration that is transmitted from another device. The Agent object asks the server in order to establish the socket connection through a function call. Likewise, a connected socket can be received through a function as well.

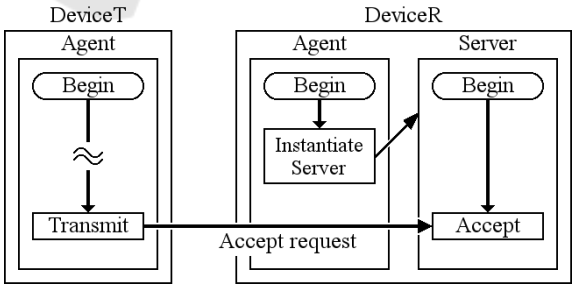

Figure 2: Receive request.

3.1.2 Execute

This class receives agents from other devices using the communication connection established by the server and executes them as an independent thread. When a member function of this class, i.e. execute

function, is executed, as shown in Figure 3, the server communicates through the socket connection, and the received data are saved as files on the local disk.

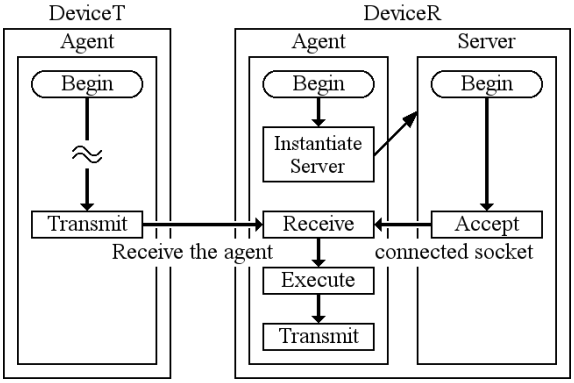

Figure 3: Receive agent.

3.1.3 Routing

This class determines the destination device of the agent. By default, as shown in Figure 4, the routing function attempt to connect in the order of the address list prepared in advance, and the succeeding address is taken as the next destination. The user can override the routing member function in a derived class.

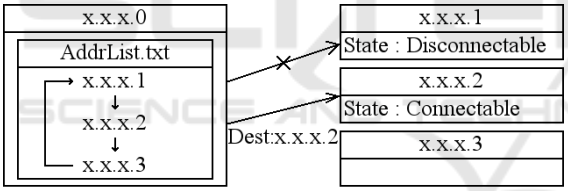

Figure 4: Default routing algorithm.

3.1.4 Transmit

This class transmit agents to other devices. This class inherits the Routing class. When the transmit function, which is a member function of this class, is executed, the inherited routing function is executed to determine the destination device. And then, the transmit function communicates with the determined destination device and transmits the designated agent. The user can override the transmit function in a derived class.

3.1.5 Agent

When the Agent class is instantiated and an Agent thread starts, the Agent thread operates as an independent thread different from other threads. Figure 5 shows the flow of an Agent thread. The Agent thread first instantiates the Server class.

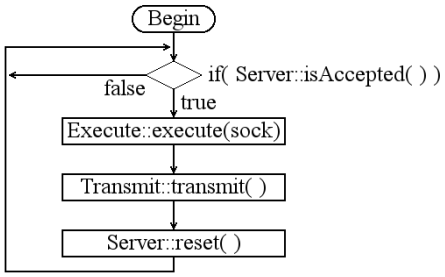

Figure 5: Agent thread flowchart.

Since the server also starts the server thread, as shown in Figure 6, multiple threads are executed at the same time. After instantiating the Server class, the agent thread waits until the server thread receives a request of an agent migration. When a request of an agent migration is received, the Agent thread executes the execute function. The execute function receives an agent and local data files, if the agent uses them through a socket connection established by the server, and then makes the agent running. When the agent's processing completes, the Agent thread executes the transmit function and transmits the agent to the next destination. After transmitting the agent, the Agent thread resets the server and waits again until it receives another request of agent migration.

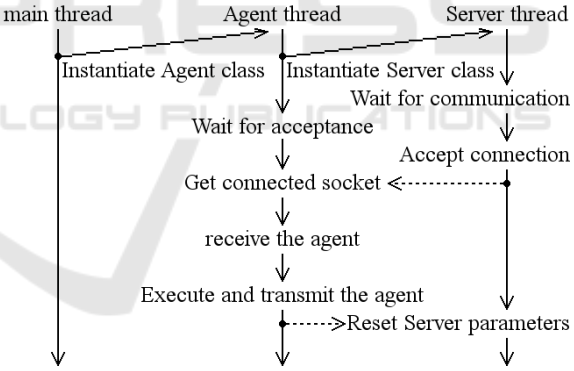

Figure 6: All running threads.

3.2 User Agents

This agent system executes and moves an agent prepared in advance in the local directory. The user prepares a file used as an agent and data files necessary for the agent's processing in the local directory.

When the user constructs a user agent class, he or she is expected to prepare a class that inherits the Agent class, such as the MyAgent class, as a program that uses our agent system. The execute function, the transmit function, and the routing function can be overridden from the derived class, if necessary.

The agent starts processing for the first time after it is received. The entire process is initiated by transmitting an agent to the first device using a program that only transmits the agent.

4 EXAMPLE OF MULTIROBOT APPLICABILITY

In order to investigate the feasibility of our simple agent system, we have implemented a set of search robots using our mobile agent system. The set of our robots implements the particle swarm optimization (PSO) for its search for a target. Potential usefulness of PSO for search robots is demonstrated (Ishiwatari et al., 2017).

4.1 Particle Swarm Optimization

The particle swarm optimization (PSO) is an optimization algorithm using swarm intelligence proposed by Kennedy and Eberhart (1995). PSO is a method of performing a search using modelled particle swarm. Each particle has position and velocity, and adjusts its position and speed from surrounding particles. Particles converge to a position considered as a solution. If x is the position and v is the velocity, each particle repeats the update of its own position and speed using the following equation:

$$x \leftarrow x + v \tag{1}$$

$$v \leftarrow wv + c_1 r_1(\hat{x} - x) + c_2 r_2(\hat{x}_g - x) \tag{2}$$

w is the inertial constant. c_1, c_2 are the proportion of particles going to a better position. r_1, r_2 are random numbers. $\hat{x}$ is the local best. $\hat{x}_g$ is the global best.

These parameters are stored in data files. We prepared three files for storing the particle data: Particle.txt, LocalBest.txt, GlobalBest.txt.

Since robots represent particles of PSO data present in each particle's attributes, they are stored in files bound to each robot. They are Particle.txt and LocalBest.txt. The data shared among all the particles, i.e. robots, are stored in a file namely GlobalBest.txt. These files are bound to each mobile agent. Mobile agents are trying to make the global data consistent among the robot by migrating among the robots.

Particle.txt is a file that records the current coordinates and speed of the robot as particle data. Each robot holds its own file.

LocalBest.txt is a file that records parameters of a local best particle. Each robot holds its own file. The agent records the nearest neighbour coordinates on the trajectory of the robot in LocalBest.txt as shown in Figure 7.

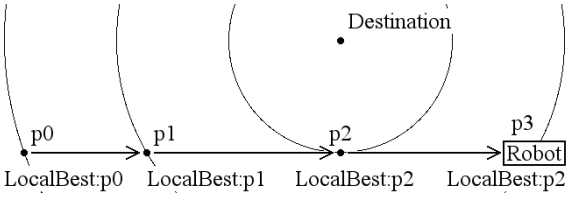

Figure 7: LocalBest at each point.

GlobalBest.txt is a file that records parameters of a global best particle. This file moves with the agent and shares among the robots as the global best for the robots toward the destination. The agent records in GlobalBest.txt the coordinates of the robot that was the closest to the destination among the robots that it got through as shown in Figure 8.

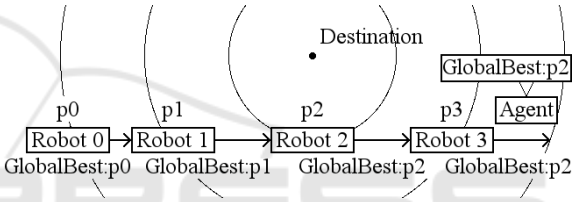

Figure 8: GlobalBest at each robot.

Figure 9 shows a robot's local directory. The robot's control computer possesses Particle.txt and LocalBest.txt in its local directory. The visiting agent possesses GlobalBest.txt, but it is a separate file so that the control computer can access this file as well.

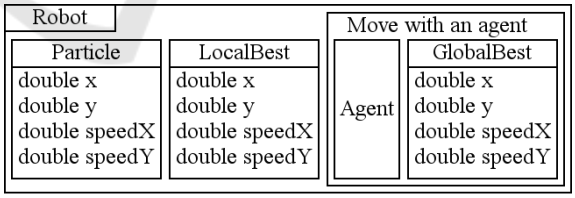

Figure 9: Particle files in the robot's local directory.

4.2 Search Agent

Our agents uses PSO to search. The agents perform three tasks to search as follows:

1. acquisition of parameters necessary for PSO,
2. update of parameters with acquired coordinates, and
3. recording of updated parameters.

The robots approach the destination by the agents' repetitions of these three tasks, i.e. acquiring, updating and recording of parameters among the colleague search robots.

The agent reads the parameters from the files in the robot's local directory. We use an orthogonal coordinate system, and the *x* velocity and the *y* velocity are provided as parameters for each robot. Every robot first needs to set its own parameters and local best. Therefore, the agent needs to determine whether the robot of the destination has already been initialized. We decided to use an empty file named Initialized.txt to determine if the parameters of the robot were initialized. The agent causes the robot that executed the agent to create Initialized.txt. The agent determines whether data has been initialized, based on whether Initialized.txt exists in the robot.

When the agent is executed, it first checks whether the robot that is running the agent has Initialized.txt. If Initialized.txt does not exist in the robot, the agent initializes the parameters, creates Initialized.txt, and then acquires the particle data. If Initialized.txt exists, i.e. it is already initialized, the agent acquires particle data immediately.

In parameters acquisition, the agent reads each data file and obtains parameters as local variables. If the distance from the current robot to the destination is closer than the distance from the local best coordinates or the global best coordinates, the local best or global best is updated with the current coordinates. After obtaining the parameters, it updates the parameters. When updating, the agent substitutes the acquired coordinates into the calculation formula of PSO, i.e. equation (1) and (2), and modifies the coordinates and speed. After updating the particle data, the agent records the updated data in each file and moves with GlobalBest.txt to another robot. The agent repeats these steps of processing with the next destination.

4.3 Experiment

We have constructed a simulator of multi-robot system based on our multi-agent system, and conducted an experiment in order to demonstrate that our agent system is feasible for searching with PSO. We have set the search field as a two-dimensional field of 1024 × 1024 and set the destination as (512, 512). We set parameters for each of the three robots as follows:

- Robot 1 (*x*=0, *y*=0, *speedX*=1, *speedY*=1).
- Robot 2 (*x*=768, *y*=256, *speedX*=-1, *speedY*=0).
- Robot 3 (*x*=256 *y*=1024, *speedX*=1, *speedY*=0).

In this experiment, assuming that there is no local solution, we updated the velocity of each particle with a simplified formula with an inertial constant w set to 0.9 and a random number r_1, r_2 associated with local best and global best fixed at 0.5.

Figure 10 shows the result. Figure 10 (a) shows the initial step, Figure 10 (b) shows the situation after ten steps, Figure 10 (c) shows the situation after thirty steps, and Figure 10 (d) shows the situation after fifty steps. In comparison with the initial values of robot 1, *x* approached 462.23 and *y* approached 510.387. In comparison with the initial values of robot 2, *x* approached 243.33 and *y* approached 242.832. In comparison with the initial values of robot 3, *x* approached 224.881 and *y* approached 479.691.

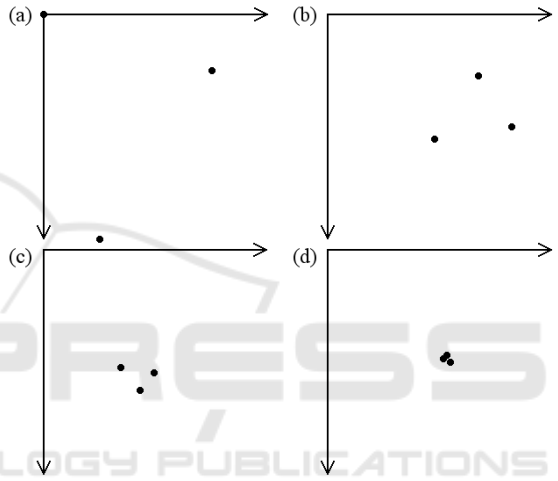

Figure 10: The position of each particle.

We have observed that the whole particles approached to the optimal solution. From this observation, we can conclude that our agent system works well for search robot using PSO. Although the simulation uses only three robots in this simple experiment, the behaviors of the agents are as expected, and we can expect that the many robots can cooperatively search for a target. We believe that it is possible to search for multiple robots in an actual machine by acquiring the coordinates with GPS.

5 CONCLUSIONS

We have developed an agent system in C ++ language. The system is designed and constructed to be easily extensible so that we can add and modify functions by extending the base classes. Since our agent system can perform multi-hop communication in ad-hoc mode, we believe that it can be used for less

restrictions and wide range of applications. For example, we can record the address of devices that passed while the agent was moving, and use it to return to the transmitter of the agent.

We have developed a simulator of a multi-robot system to demonstrate the feasibility of our agent system. When the user executes the agent system, the agents move to robots that mimic PSO in the multi-robot system. We could demonstrate the feasibility of our agent system on a set of search robots.

In the current system, each agent is passed as a program file, and when it arrives at the destination, the agent does not have any local data. Local data are also sent with the agent so that the agent acquires the necessary data from the file. This is a disadvantage. We will re-design the agent system so that the agent can move with its own environment and maintain its own state even during execution.

The current system does not allow interruption during the execution and the migration of the agent. At the next stage, we will extend the agent system so that execution can be interrupted and moved to a destination. And then, we will build an actual multi-robot system for search as a future work.

ACKNOWLEDGEMENTS

This work is partially supported by Japan Society for Promotion of Science (JSPS), with the basic research program (C) (No. 17K01304), Grant-in-Aid for Scientific Research (KAKENHI) and Suzuki Foundation.

REFERENCES

Braun, P. and Rossak, W.R., 2005. Mobile Agents: Basic Concepts, Mobility Models, and the Tracy Toolkit, Morgan Kaufmann.

Ishiwatari, H., Sumikawa, Y., Takimoto, M. and Kambayashi, Y., 2017. Cooperative Control of Multi-Robot System Using Mobile Agent for Multiple Source Localization, In *Eighth International Conference on Swarm Intelligence*, pages 210-221.

Kambayashi, Y. and Takimoto, M., 2005. Higher-Order Mobile Agent for Controlling Intelligent Robots, *International Journal of Intelligent Information Technologies*, 1(2), 28-42.

Kambayashi, Y., Nishiyama, T., Matsuzawa, T., and Takimoto, M., 2016. An Implementation of an Ad hoc Mobile Multi-Agent System for a Safety Information, In *Thirty-sixth International Conference on Information Systems Architecture and Technology*, AICS vol. 430, pages 201-213.

Kennedy, J. and Eberhart, R., 1995. Particle swarm optimization, In *IEEE International Conference on Neural Networks 4*, pages 1942-1948.

Nagata, T., Takimoto, M. and Kambayashi, Y., 2013. Cooperatively Searching Objects Based on Mobile Agents, *Transaction on Computational Collective Intelligence XI*, pages 119-136.

Nishiyama, H., Ito, M., Kato, N., 2014. Relay-by-Smartphone: Realizing Multi-Hop Device-to-Device Communications. *IEEE Communications Magazine*, vol. 52, no. 4, Apr, pp. 56-65.

Oda, K., Takimoto, M. and Kambayashi, Y., 2018. Mobile Agents for Robot Control Based on PSO, In *Tenth International Conference on Agents and Artificial Intelligence*, pages 220-227.

Stone, P. and Veloso, M., 2000. Multiagent systems: A survey from a machine learning perspective, *Autonomous Robots*, 8(3), 345-383.

Taga, S., Matsuzawa, T., Takimoto, M. and Kambayashi, Y., 2017. Multi-Agent Approach for Evacuation Support System, In *Ninth International Conference on Agents and Artificial Intelligence*, pages 220-227.

Taga, S., Matsuzawa, T., Takimoto, M. and Kambayashi, Y., 2016. Multi-Agent Approach for Return Route Support System Simulation, In *Eighth International Conference on Agents and Artificial Intelligence*, pages 269-274.

Taga, S., Matsuzawa, T., Takimoto, M., Kambayashi, Y., 2018 Multi-agent Base Evacuation Support System Using MANET, In *Tenth International Conference on Computational Collective Intelligence*, LNAI 11055, pages 445-454.