Mobile Agents for Robot Control based on PSO

Koji Oda[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, Python, Multi-robot, Particle Swarm Optimization.

Abstract: It is a fundamental concern for the robot research community to explore unknown environments. This paper presents an approach for controlling multi-cooperative robot exploration in an unknown environment. In order to control multiple robots, we take advantage of multiple mobile agents. In this paper, we report our experience in implementing a mobile multi-agent system written in Python 2. We chose Python because it is easy to read and write, and has a rich library. To construct an efficient search algorithm for multiple mobile robots, we employ an extended particle swarm optimization (PSO) for the robot control algorithm. This algorithm allows robots to avoid obstacles while utilizing the PSO as a basic search algorithm. To demonstrate the feasibility of this research, we implemented an agent platform in Python 2, along with a team of mobile robots controlled by a team of mobile agents on this platform. The imaginary application is a search and rescue operation after disaster in an urban area. We believe that the combination of mobile agents and mobile robots contributes lifesaving applications.

1 INTRODUCTION

During the last two decades, robotic systems have made rapid progress, not only in their behaviors but also in the way they in which they are controlled. In particular, a control system based on multiple software agents can be used in controlling robots in an efficient manner (Mondada et al., 2004); (Kambayashi and Takimoto, 2005); (Parker, 2008).

We have conducted research projects that have taken advantage of multiple mobile agents. These projects include evacuation route generation at the time of a disaster using multiple mobile agents, and searching for victims after a disaster (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 resource search is achieved through communication among multiple mobile agents.

We have continued applying mobile agents to other researches. Studies using particle swarm optimization (PSO) as a search algorithm are also being conducted using multiple robots (Zhou et al., 2011). Such researches have applied multiple robots as particles of PSO. However, there are problems in using PSO for searching in disaster fields. For example, because the PSO algorithm does not recognize the existence of an obstacle within the search range, we extended the PSO for our research in order to solve this problem. This algorithm cooperates with other robots conducting a search, and performs actions to avoid various obstacles. A simulation was also carried out.

Certainly, robots that perform tasks such as exploration and photography have contributed to disaster relief in recent years. However, because many of those robots are manipulated by human beings using wired connections, if problems such as a disconnection of the connection lines occur, the robot will go missing. We are especially concerning a case where noxious or radioactive circumstance. Therefore, we are developing an autonomous search robot that does not require human control. Therefore, it is possible for a robot team to operate in a field too dangerous for human beings. In addition, we do not have to worry about a loss of communication. However, because a command is issued before the search is started, a flexible operation based on human judgment is not reflected. The mobile agent system enables us to send any algorithms or intelligence required by intelligent search robots by sending mobile agents after the robots are dispatched.

DOI: 10.5220/0006727303090317
In *Proceedings of the 10th International Conference on Agents and Artificial Intelligence (ICAART 2018) - Volume 1*, pages 309-317
ISBN: 978-989-758-275-2

Such a control algorithm of an autonomous search robot is difficult without using a mobile software agent.

We have accumulated experience on various simulators. In an environment such as a disaster area, however, we cannot expect stable communication in a real situation. In addition, it is unknown how long it will actually take to communicate between the search robots, or the time required to reach the target search object, unless the control algorithm is developed to work on actual machines.

A new question arises when considering the operation on an actual machine. This is a question of what type of method should be used to share data.

In this study, we have constructed a mobile software agent system written in Python 2 that is dedicated to operation on a small computer applied in a robot. We report the effectiveness of the system in human search and rescue under difficult circumstances using PSO and extended PSO.

The structure of the balance of this paper is as follows. In the second section, we describe the background. In the third section, we describe our mobile sofware agents implemented in Python 2 and how the agents control multiple robots in a search field. The fourth section discusses the problems we have to overcome in order to construct a practical search robot team that can operate in a real disaster field, and finally, in the fifth section, we provide some concluding remarks.

2 BACKGROUND

Thus far, a number of studies on the use of agents have been conducted (Stone et al., 2000) (Kambayashi and Takimoto, 2005); (Avilés et al., 2014); (Shibata et al., 2014); (Kambayashi et al., 2016); (Taga et al., 2016, 2017). Among them, we are focusing particularly on mobile agents. We have constructed a multi-agent system and have been conducting research on resource searches and evacuation route generation. The technologies used in this research are introduced below.

2.1 Mobile Agent

A mobile agent is a program that contitues processing while moving between computing sites. A mobile agent can autonomously choose the destination to which it migrate. As an example application of a mobile agent, we can imagine multiple resource-searching robots controlled using mobile software agents.

Autonomous search robots occasionally search unnecessary areas. This means that they consume energy unnecessarily. In order to mitigate this problem, we introduce mobile agents that allow multiple robots to share information between them. We can also use mobile agents to issue instructions to control multiple robots during a particular action. As a result, a more efficient search for resources can be accomplished. Using this feature, Taga et al., (2016; 2017) proposed a method for deriving a return home route after a disaster using a mobile agent.

2.2 Particle Swarm Optimization

PSO is an optimization algorithm using swarm intelligence proposed by Kennedy and Eberhart (1995). It is based on the behavior of insects and schooling fish and is an algorithm imitating the habit of gathering at locations where there is a high possibility of food being found. Figure 1 shows the process of particle gathering.

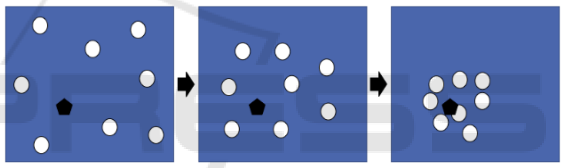
Figure 1: Transition of particles.

Eight or more particles each search for and finally gather near the target. Particle movement is determined using three values. The first one is the localbest. It stores the best value and coordinates that the particle has searched. The second is globalbest. It shares the coordinates with the best value in the search of an entire particle group. The third is inertia. Each particle holds a value indicating how much it should move from its previous coordinates, and is influenced by this value.

PSO is defined using the two equations above, where *i* is the particle number, and *t* is the time. In addition, ω is the inertia coefficient, and is set based on how much inertia is affected from the vector V_i^t from the previous point. Then, V_i^{t+1} is a vector that moves to the next coordinate (1). To calculate this value, we need the constants C_1 and C_2, the random numbers $rand_1$, $rand_2$, the distance from the current location to localbest P_i^t, and the distance to globalbest P_g^t. The coordinate of the next point is generated from the movement vector to the current position and the

$$V_i^{t+1} = \omega\, V_i^t + C_1\, rand_1\, (\, P_i^t - X_i^t\,) + C_2\, rand_2\, (\, P_g^t - X_i^t\,) \qquad (1)$$

next point (2). Figure 2 shows the process for determining the destination coordinates.

$$X_i^{t+1} = X_i^t + V_i^{t+1} \quad (2)$$

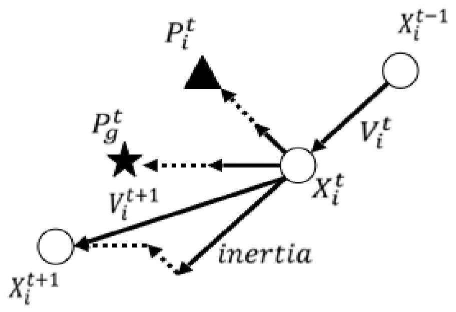

Figure 2: Destination coordinate determination.

A multi-robot resource search based on PSO has proved that more than eight robots can find targets in an open space without a collision (Nighot et al., 2012). In addition, Zhu et al. proposed an algorithm that enables robots to search for targets without accurate position information (Zhu et al., 2011). However, they are intended to search in open spaces.

2.3 Extended PSO

Extended PSO described in this paper is an extension of PSO that includes an obstacle avoidance algorithm. Ishiwatari et al. showed its theoretical effectiveness on a simulator (Ishiwatari et al., 2017). We demonstrated its effectiveness using actual machines.

In general, PSO does not recognize the presence of obstacles within the search range. For this reason, when a robot is conducting a search, it is strongly affected by obstacles. If the robot cannot easily avoid an obstacle, time will be wasted. In order to solve this problem, a method for generating a route to bypass an obstacle using a mobile agent has been developed and studied (Uehara et al., 2016). The effectiveness of this agent was confirmed through simulations too.

An operational outline of extended PSO is as follows. First, when a robot detects an obstacle, it stops its movement and sends the mobile agent to the nearest terminal. The agent holds the movement log of the robot, and compares the movement logs of two robots to judge whether there is an intersection. If so, the agent moves to the other robot. This process is repeated until the agent reaches the target robot. Ultimately, the mobile agent reaches the target robot within the range of L and D in Fig. 3, which are arbitrarily set.

L and D are empirically derived constants. They are used to determine to which robot the mobile agent should migrate. L is the distance and set to be shorter than the wireless range. The moving distance on the straight line between the source and the target robot needs to be not less than L. On the other hand, D is the angle which indicates the rough direction to which the mobile agent migrate. The target robot needs to be within the angle D in front of the source robot. Uehara et al. have intensively studied the optimal values of L and D (Uehara et al., 2016).

The mobile agent then follows the moving robot in reverse order and returns to the first robot. Next, the collected data are handed over to the robot, and the agent stops functioning. The robot receives the search results of the alternative route. The robot moves to a position based on such data and restarts the search according to the PSO algorithm. The calculation of the next point continues even when a detour is being searched. If the next point becomes an obstacle-free direction owing to the influence of globalbest, the search for an alternative route is stopped, and the search of the PSO is resumed. In this way, a detour is created to avoid wasting time if obstacles are present. Figure 3 shows how the extended PSO generates a detour.

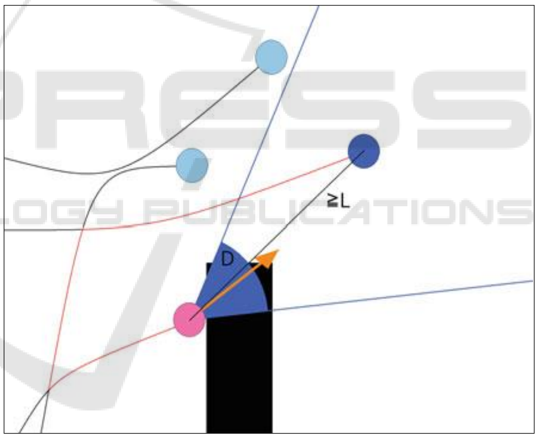

Figure 3: Operation of extended PSO.

Another feature of this algorithm is the replacement of robots. When robots detect a collision, they continue searching after exchanging information such as the search logs. They continue with their actions as if the robots had simply passed each other. This is because it is physically difficult for robots to avoid and pass one another. Uehara et al. (2016) demonstrated the theoretical effectiveness of this algorithm.

2.4 Multi-robot System

Search and rescue can be efficiently conducted using multiple well-coordinated robots. We addressed the collaboration between robots by employing multiple mobile software agents. Because a robot operates using batteries, the driving time is limited. Thus, it is necessary to conduct an efficient search in less time. In this study, we aim to introduce extended PSO using a mobile agent.

3 PROPOSED METHOD

In this study, we use mobile software agents to allow robots to share the globalbest value of PSO, and conduct route generation using extended PSO. To achieve this purpose, we created a platform for sending and receiving mobile agents.

3.1 Python Agent Platform

A control program is used to send and receive agent programs. The control program created in this study is based on a Python agent platform, which is installed in all robots. We have created this Python agent platform for this purpose.

The platform consists of two parts. One is the server part, which receives and processes agents. The other is the client part, which transmits agents. The server part waits for communication from the client to receive an agent. On the other hand, the client specifies the IP address of the server and sends the agent to the remote server of the address. At that time, the program code of the mobile agent is transmitted after being converted into a character string. Then, the server side of the remote site receives the character string, and converts it into a program file. The server then imports the content of the program file as a module, and executes it. This is similar to the serialization and deserialization in the Java system. Figure 4 shows the schematic diagram of the agent platform.

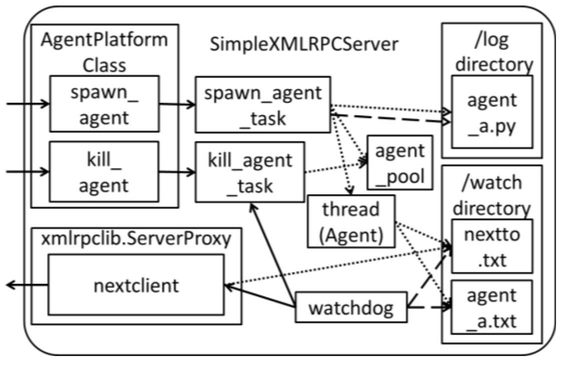

Figure 4: Agent platform overview.

The solid lines represent the function invocations. The large dotted lines indicate references, and the small dotted lines indicate the relation between the generation and deletion of files. Another agent platform on the left is assumed, and communication is carried out. Details of this are provided in the following subsections.

3.1.1 Initialization

All the robots are controlled using mobile agents. Each robot waits for a starting signal that is issued by the mobile agent. This initial transmission of the mobile agent is achieved using the main controller on the server PC. The main controller issues one mobile agent for each robot as the starting signal. The content of this signal is a file called start.txt. The initial mobile agent puts this text file in the robot's directory. When the robot recognizes this file, it starts searching.

3.1.2 Creation of Mobile Agents

The mobile agent program file is distributed to each robot as start.txt. The mobile agent program can be easily created, similar to a normal Python program. The following program fragment shows the skeleton of a typical agent program.

```
def main(name, arg):
   f=open('./watch/'+name+'.txt','w')
   f.write(anything)
   f.close()

   −here the body of the agent activities.

   os.remove('./watch/'+name+.txt')
```

The name of the agent created is given as an argument to the constructor. This name is used as an accompanying text file name that indicates the existence of the agent when it arrives at a new site. When the agent leaves the site, or commits suicide, its destructor deletes this text file.

When the mobile agent wants to migrate to another site, it notifies the server regarding where it will move next. In order to do so, the mobile agent creates a text file called nextto.txt, and writes the IP address of the target server.

```
f=open('./watch/nextto.txt', 'w')
f.write(IPaddress)
f.close()
```

3.1.3 Sending Mobile Agent

Upon receiving the agent's request, the agent server calls the client function to send the agent. The following program fragment shows the client function.

```
def nextclient(task,name,arg):
   server="http://"+nextip+":7070"
   c=xmlrpclib.ServerProxy(server)
   c.spawn_agent(task,name,arg)
```

The client function uses xmlrpclib of the Python library. This library follows XML-RPC. First, the destination IP address is set as the URL. Then, the xmlrpclib.ServerProxy function is instantiated. This function calls the function of the target server. By activating the spawn_agent function existing on the target server, the mobile agent starts up and begins moving.

The arguments include the task (mobile agent program code) and name (mobile agent name), among others. The program code uses an open function to refer to the program file. Alternatively, it is also possible to transmit the received code as is or after modification.

3.1.4 Receiving Mobile Agent

The server that receives the mobile agents uses the SimpleXMLRPCServer library as follows:

```
s=SimpleXMLRPCServer((myip, 7070))
s.register_instance(AgentPlatform())
s.serve_forever()
```

In the initialization step, each server obtains its IP address and assigns it to the variable myip. Next, it specifies the AgentPlatform class using the register_instance function in order to invoke the methods defined in AgentPlatform. When all the servers in this agent system perform this action, all the functions in AgentPlatform are available for all the servers in the system. It is necessary for the server that receives the mobile agent to be active at all times. In order to achieve this, the serve_forever function is used.

3.1.5 Starting Mobile Agent

When the client starts the spawn_agent function in the AgentPlatform class on the target server, the agent starts up and begins moving. This is possible because all the functions in all the AgentPlatform classes are visible.

The received server creates a file called name.txt using “name” in the “/log” directory. Next, it stores the file as a module in the mloader library, and then deletes the file. The mobile agent program is then executed by assigning a thread.

3.1.6 Mobile Agent Movement

In this section, we describe how the code in Section 3.1.2 works. The server is always monitoring the “/watch” directory in the watchdog library. This makes it possible to conduct operations triggered through a file creation, modification, or deletion.

When an agent creates a file called name.txt in the directory, the server creates a name item in a dictionary called agentpool. In it, the task, arg, and tid are described. Here, tid is the return value at thread startup. Then, when the agent deletes name.txt, the server deletes the name item from within agentpool using the kill_agent_task function. With this flow, we manage the list of running agents.

On the other hand, when the agent generates a file called nextto.txt, the server refers to the contents of the document. Then, when the mobile agent stops its operation, it executes the client function for that IP address and transmits the mobile agent.

3.2 PSO

In this study, robots conduct a search using extended PSO. For this reason, we first create a moving algorithm using ordinary PSO and install it in the robot. Next, the operation of extended PSO is added. In PSO, each particle generates coordinates of the next point from three values: localbest, globalbest, and inertia. Among them, globalbest should always be shared with other particles. A mobile agent is used as a sharing method.

In this study, because a robot is regarded as a particle, it is necessary to delicately adjust the motor to move to the next point. The motor control at this time is described in Section 3.4.

The mobile agents are responsible to allow all the robots to share the globalbest. When a robot obtains a better value than the currently held globalbest, it shares its value and the current coordinates with the other robots. Sharing will transmit only one hop mobile agent to each robot. This agent compares its value with the text file holding the globalbest of the receiving side, overwrites the higher score, and disappears. In this way, the values are shared.

At this time, we use the radio field intensity indicator as the evaluation value. This value is called the received signal strength indicator (RSSI). All values are negative, and the closer a value is to zero, the stronger the intensity, whereas the further away from zero the value is, the weaker the intensity. Therefore, as the value approaches zero, it can be judged as approaching the coordinates of the search target.

In order to obtain the RSSI, the command iwconfig is common. However, this command must be connected to the target access point, which is not suitable for searching for an unspecified terminal called a victim. Therefore, we created a new command called "getsi" that can be executed on the terminal. The contents of the getsi command are as follows:

```
#!/bin/sh
sudo iwlist wlan0 scan
    | grep -e ESSID -e Quality
```

By executing this getsi command, it is possible to obtain the SSID and RSSI values within the range that can be currently recognized. The link quality refers to the connection quality, and discrimination of noise and other factors is carried out. However, because it is difficult to obtain fine variations in values, utilization at the time of medium distance use is considered. At this time, we use RSSI as the main evaluation value.

The RSSI can be obtained using the above command. By applying it, we can execute this terminal command on Python using a subprocess library. By adjusting the result, we can obtain the RSSI value. Thus, an evaluation value at this coordinate is obtained.

3.3 Extended PSO

We extend PSO such that the algorithm includes a feature for avoiding obstacles. Uehara et al. have developed this avoidance algorithm (Uehara et al., 2016). A mobile agent is used to generate a route to avoid obstacles. At that time, other search robots cooperate in route generation while conducting a search as usual.

When a robot detects an obstacle, it stops and generates a mobile agent. It then transmits to another wirelessly connected robot. This agent moves between the robots while holding the moved coordinate log stored in each robot. The moved agent determines whether there is an intersection of the movement log of the current robot and the movement log of the transmission source robot; it then holds the log and moves to the next robot if it exists. On the other hand, if there is no intersection point, it goes back one step and moves to another connectable robot and compares the logs. Finally, if a robot within L meters and D degrees of the front of the source robot is reached, it takes the previous movement log back to the source robot. After that, interrupt handling is applied at the server, and a detour route is generated to avoid any obstacles.

The source robot shares its globalbest even before the agent returns. Therefore, as the coordinates of the globalbest are changed, the search may be possible without bypassing the obstacle. At this time, the search is immediately resumed.

When robots collide, they exchange all information with each other. Then, the two robots behave as if they had passed one another. For this purpose, the robot stores all the values such as the movement log and localbest in an array and transmits the array using a mobile agent. When the exchange of the values is completed for both robots, the replacement ends. Next, both robots continue searching for the next point. Ishiwatari et al., (2017) developed this exchange algorithm.

3.4 Search Robot

Search and rescue is achieved using a team of mobile robots. Because each search robot needs to communicate with other robots, it is necessary to construct a wireless network. Therefore, we created robots equipped with Raspberry Pi. Our agent system has been built to perform on top of the Raspbian operating system in Raspberry Pi. Raspberry Pi uses the GPIO pin to acquire data from the sensor and control the motor. The robot has a caterpillar, and was designed for comfortable rotation, making a change in direction to the next point through PSO easy. With PSO, because the sharing of coordinates is important, controlled movement to an accurate position is necessary. Therefore, directional data are acquired using a compass module, which is used for angle control. In addition, the coordinate data are obtained through a GPS sensor. Because it is difficult to adjust the position using a motor, a highly accurate operation by the compass module is important. The power supply supplies power from a mobile battery for smartphone use. Figure 5 shows the search robot.

Because Raspbian is used as the operating system of Raspberry Pi, and the agent platform is developed using Python 2, the robot's motor control system and PSO were also written in Python. The reason for adopting Python is that, because the agent platform is on Python, it was judged that a more efficient arrangement can be obtained if the same language is used, and the behavior, including the robot's motion toward the mobile agent, can be directly described. As a result, it is possible for the moving agent to control the robot through interrupt handling.

To control the motor, we use the wiringpi library and send instructions from the GPIO pin to the motor driver. Four AA batteries are used for motor control, and are installed separately from mobile batteries.

Figure 6 shows a schematic diagram of the robot system. Descriptions of each part are described below.

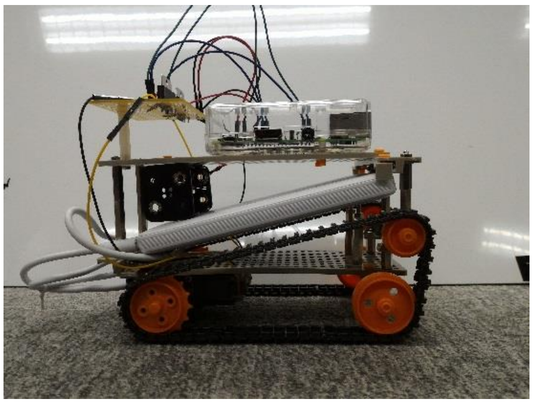
Figure 5: The search robot.

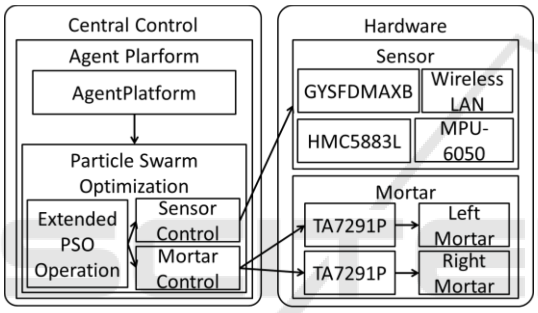

Figure 6: System configuration overview.

Central Control uses Raspberry Pi. Coordinate calculations of PSO are performed based on data from each sensor. At this time, the globalbest is shared using the mobile agent. The next coordinates are calculated, and the motor is controlled to guide the robot to the next coordinates. At this stage, errors from the motor may occur, and it is necessary to continuously monitor the robot using a compass module, GPS sensor, or similar device, correct any errant values, and move to the accurate coordinates.

Agent Platform receives and executes an agent. The agent starts operation from the client by activating the function in it. It manages what agents reside on the robot. In addition, when the agent is a mobile agent, as described above, a new agent program is generated, and a function used as a client is activated by specifying the next movement destination. This is a control program that mainly manages and operates a list of agents.

Sensors collect the surrounding data are numerized and imported into a robot. Based on these values, PSO is executed. Because the sensors are connected to GPIO pins, addition and reduction are easy to achieve. Data essential for PSO, such as the coordinates and angle, are obtained from each sensor. The data are analyzed programmatically and reflected in the robot operation. Table 1 shows the mounted sensors.

Table 1: Mounted sensors.

GPS sensor	GYSFDMAXB
Compass module	HMC5883L
Acceleration sensor	MPU-6050

Mortar is TA7291P that is used for the motor driver applied to the motor control. Control is carried out using a program with the aforementioned carpi library. The speed of the motor can be adjusted. It is important to rotate at a low speed so as not to avoid overlooking the value of the compass module when turning toward the next movement point. Four AA batteries are used as the motor power supply.

Figure 7 shows a team of mobile robots search a target under control of mobile agents. The Wi-Fi access point in the photo is the target.

4 DISCUSSION

For the experiment applied during this project, we used an environment in which communication through wireless LAN can be established. The agent uses the IP address to specify the transmission destination. Therefore, this agent platform cannot operate unless the terminal is given an IP address.

However, we may need to cope with an environment in which wireless LAN is not available. In such a case, it may be necessary to achieve communication using a mobile ad hoc network. This results in a dynamic network configuration and very unstable communication. However, if information sharing using mobile agents can be achieved, the search can be conducted more efficiently.

If a mobile ad hoc network is to be implemented, routing that takes into account network congestion and terminal power consumption becomes necessary. Furthermore, it is necessary to deeply consider a method for bypassing the communication when a terminal is disconnected. Such methods have been actively debated (Kambayashi et al., 2016).

5 CONCLUSIONS

In this paper, we proposed and developed a mobile

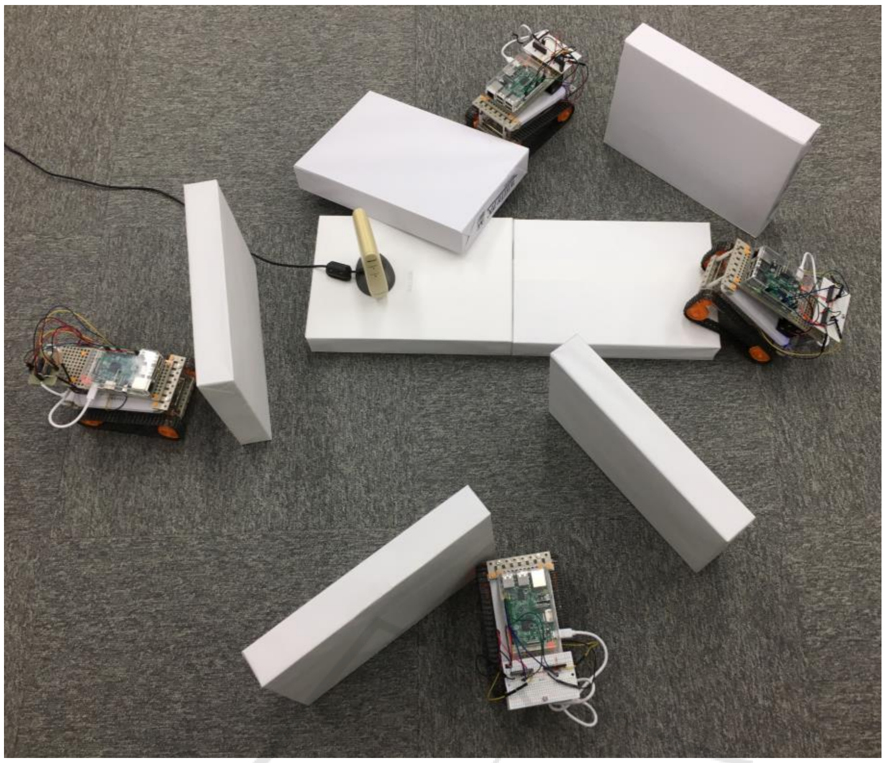

Figure 7: Team of robots cooperatively search.

agent system using Python in order to create a system that carries out expanded PSO using real robots, which conduct a search as PSO particles. The imaginary application is a search and rescue operation after disaster in an urban area. Even though the current system is a team of prototype robots, revised robots should be able to locate the victims buried in rubble in the disaster area.

In addition, we believe that flexibility between mobile agents and mobile ad hoc networks is compatible within a limited environment such as a disaster site. However, in constructing MANET, the problem raised in Section 4 occurs. In order to operate a search system using an agent system at a real disaster site, it is necessary to address these issues.

ACKNOWLEDGEMENTS

The authors appreciate Yuta Arai and Taiki Miyakawa for their contribution in the implementation of the robot systems. 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

Avilés, A., Takimoto, M. and Kambayashi, Y., 2014. Distributed evacuation route planning using mobile agents, In *Transactions on Computational Collective Intelligence XVII*, pages 128-144.

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

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 *Proceeding of the Thirty-sixth International Conference on Information Systems Architecture and Technology*, pages 201-213.

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

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

Mondada, F., Pettinaro, G. C., Guignard, A., Kwee, I. W., Floreano, D., Deneubourg, J., Nolfi, S., Gambardella, L. M., and Dorigo, M., 2004. Swarm-Bot: a New

Distributed Robotic Concept, *Autonomous Robots*, pages 193-221.

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

Nighot, M. K., Patil, V. H. and Mani, G. S., 2012. Multi-robot hunting based on swarm intelligence, In *Hybrid Intelligent Systems (HIS), 2012 12th International Conference*, pages 203-206.

Parker, L. E., 2008. Distributed intelligence: overview of the field and its application in multi-robot systems, *Journal of Physical Agents*, 2(1), pages 5-14.

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

Shibata, K., Takimoto, M. and Kambayashi, Y., 2014. Expanding the Control Scope of Cooperative Multiple Robots, Proceedings of the Eighth KES International Conference on Agent and Multi-Agent Systems: Technologies and Applications, In *Advances in Intelligent Systems and Computing Volume 296*, pages 17-26.

Taga, S., Matsuzawa, T., Takimoto, M. and Kambayashi, Y., 2017. Multi-Agent Approach for Evacuation Support System, In *Proceeding of the 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 *Proceeding of the Eighth International Conference on Agents and Artificial Intelligence*, pages 269-274.

Takahashi, R., Takimoto, M. and Kambayashi, Y., 2015. Cooperatively Transporting Unknown Objects Using Mobile Agents, In *Agents and Artificial Intelligence: 6th International Conference*, pages 46-62.

Uehara, S., Takimoto, M. and Kambayashi, Y., 2016. Mobile Agent Based Obstacle Avoidance in Multi-Robot Hunting, In *Proceedings of the 20th Asia Pacific Symposium on Intelligent and Evolutionary Systems*, pages 443-452.

Zhou, Y., Xiao, K., Wang, Y., Liang, A. and Hassanien, A. E., 2013. A pso-inspired multi-robot map exploration algorithm using frontier-based strategy. *International Journal of System Dynamics Applications*, 2(2), 1-13.

Zhu, Q., Liang, A. and Guan, H., 2011. A pso-inspired multi-robot search algorithm independent of global information, *Proceedings of IEEE Symposium on Swarm Intelligence*, pages 1-7.