Multi-Agent Approach for Controlling Robots Marching in a File
A Simulation

Yasushi Kambayashi[1], Ryosuke Shibuya[2] and Munehiro Takimoto[2]
[1]*Department of Computer and Information Engineering, Nippon Institute of Technology, 4-1 Gakuendai, Miyashiro-machi, Minamisaitama-gun, 345-8501 Japan*
[2]*Department of Information Sciences, Tokyo University of Science 2641 Yamazaki, Noda 278-8510 Japan*

Keywords: Mobile Software Agent, Multi-agent System, Multi-robot System, Self-organizing System, Simulation.

Abstract: It is a fundamental concern for multi-robot system research community how to explore unknown environments. This paper presents an approach for controlling cooperative multiple robots exploration in an unknown environment. The approach we are proposing aims to minimizing the overall exploration cost for multiple robots that march in procession. In order to achieve the goal, the file of multiple robots must be able to effectively come out of dead-ends while exploring a maze-like environment. The proposed approach employs multiple mobile software agents that can migrate from a robot to another robot freely to bring certain role and ability to a robot. In particular, the mobile software agent brings the role of leader to an arbitrary robot in a file, so that the migrated robot becomes the leader of a subgroup of the robots that can march in a file into a part of environment. In order to demonstrate the effectiveness of our approach, we have built a simulator, and partially constructed a real multi-robot system.

1 INTRODUCTION

In the last two decades, robot systems have made rapid progress not only in their behaviours but also in the way they are controlled. In particular, a control system based on multiple software agents can control robots efficiently (Kambayashi and Takimoto, 2005).

On the other hand, we have witnessed the advent of multi-robot systems. A multi-robot system consists of a large number of homogeneous robots that have limited capability but, when combined into a group, they can generate more complex behaviours (Parker, 2008). In multi-robot systems, robots communicate with each other to achieve cooperative behaviours. There are three major advantages of multi-robot systems over single robot systems (Stone and Veloso, 2000) (Yasuda and Ohkura, 2005). The first is parallelism; a task can be achieved by autonomous and asynchronous robots in a system. The second is robustness; it is realized through redundancy. The system can have more robots than required for a certain task. The third is scalability; a robot can be added to or removed from the system easily.

For the multi-robot system, a control system based on the 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, we have developed mobile agent methodologies 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 (Binder et al., 2001).

We have designed and implemented multiple mobile software agent systems that control several multi-robot systems that 1) playing a tag by dynamically exchanging roles (Kambayashi and Takimoto, 2005), 2) cooperatively assemble themselves at energy-wise optimal locations

Kambayashi Y., Shibuya R. and Takimoto M..
Multi-Agent Approach for Controlling Robots Marching in a File - A Simulation.
DOI: 10.5220/0005279002220229
In *Proceedings of the International Conference on Agents and Artificial Intelligence* (ICAART-2015), pages 222-229
ISBN: 978-989-758-073-4

(Kambayashi et al., 2012), 3) serialize themselves (Shintani et al., 2011a) (Shintani et al., 2011b), and 4) search and recollect arbitrary targets without redundant movements (Ishiwatari et al., 2014), 5) cooperatively transport unknown objects (Shibuya et al., 2013) (Takahashi et al., 2014), and 6) constructing ad hoc network (Kambayashi et al., 2014).

In the multi-robot systems, the multiple mobile software agents migrate in a herd of scattered mobile robots to collect information about them as well as drive minimum number of them based on the collected information (Kambayashi et al., 2012). Moving software agents instead of physical robots greatly save not only energy consumption but also total cost for several tasks through effective utilization of idle resources (Nagata et al., 2013).

In this paper, we propose a multi-robot system that explores maze-like environments in a file. When a file stacked in a dead-end corridor, they must come back. In general, when a file of multiple robots face to a dead-end, all the members have to go backword until the leading robot can turn into a branch of the corridor (Figure 1). The longer the file is, the more futile movements occurs. If arbitrary robot can be a leader, we can let the robot at the most convenient position be a commander of the partial file so that only partial queue of the robots have to turn and follow the leading robot (Figure 2). In order to achieve such a coordination, we employ two types of mobile software agents. One drives the robot and makes it follow the preceding robot, and the other also drives the robot and makes it lead the queue of robots and explore the corridor.

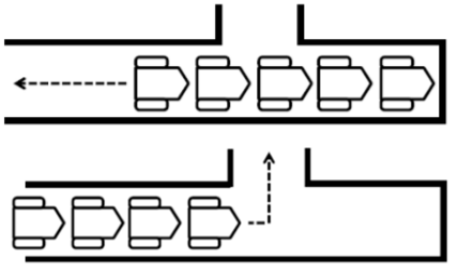


Figure 1: The file has to go back all the way until the leading robot can enter the branch.

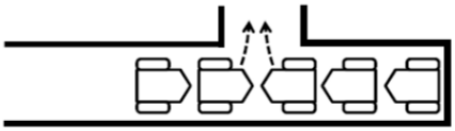


Figure 2: An arbitrary robot can be the leading robot; the order of robots in the file is changed.

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 how to control multiple robots in a file by the mobile agents. The fourth section demonstrates the feasibility of our approch through simulatoin, and we conclude our discussionsin in the fifth section.

2 BACKGROUNDS

Even though many research scientists have carried out in the area of exploration for single autonomous robot, not much of this type of works has been applied to multi-robot systems. Most of these research efforts have been taken existing algorithms that are developed for single robot exploration and have extended them to multiple robots. Just recently, research scientists have developed new algorithms that are fundamentally distributed.

Fox et al. took advantage of multiple robots to obtain more accurate positions than using single robot (Fox et al., 2000). Roumeliotis and Bekey demonstrated the ability of Kalman-filter based approach that provides a team of mobile robots to simultaneously localize by sensing their teammates and combining location information from all the team members (Roumeliotis and Bekey, 2002). Vaughan et al. proposed an algorithm that makes a team of robots be able to navigate in an unknown environment by using a trail of waypoint markers (Vaughan et al, 2002). They demonstrated that their approach can handle the problem caused by accumulating small errors of odometers, thus it is robust against the failure of individual robots. They claimed that their multi-robot system almost always converges to the best route discovered by any robot of the team.

In order to take advantage of multiple robots to explore unknown environments, formation control has been one of the most important topics of multi-robot systems. Fredslund and Mataric address the problem of achieving formation control using only local sensing and interaction (Fredslund and Mataric, 2002). Their idea is to make each robot keeps track of one particular robot within its view. They demonstrated that their approach can provide a variety of formations. Even though these researches have produced marvelous achievements, none of them takes advantage of multiple mobile agents.

Gonzalez et al. proposed an application of multi-agent system to multi-robot system (Gonzalez et al., 2011). They proposed control architecture for intentional cooperation that distributes

responsibilities by applying a hierarchical decomposition of the multi-agent cooperative control. By assigning well-defined agents to multiple robots, they achieved to make the multi-robot system pursue to solve multi-resolution problem. Even though their system is based on multi-agent model, the agents are static, not movable among robots.

Shintani et al. proposed a multi-agent system based multi-robot system (Shintani et al., 2011a) (Shintani et al., 2011b). They employed two types of mobile software agents that make scattered multiple mobile robots form a line. The notable point of their idea is that they make pheromone as mobile agents to attract and to guide mobile robots into a queue while minimizing energy consumption. Our approach is dynamically assigning the leading role to a robot in a file by migration of a mobile software agent with the leading ability.

3 CONTROLLING ROBOTS IN A FILE

In the traditional applications of multi-agent system to a multi-robot system, each agent has fixed role, is fastened on a particular robot and drives the robot to achieve its own specific role. In our approach, however, the characteristic of a robot depends on the mobile agent that currently resides on the robot. Therefore any robot in a file can be not only a follower robot but also a leader robot depending on the circumstance around it.

In order to realise such requirement, we made each robot have the following capabilities:

1. Each robot has a visible identifier, and each robot has a visual sensor to follow the preceding robot.
2. Each robot has a communication mechanism either wireless LAN or ad hoc communication mechanism so that mobile software agents migrate from a robot to another robot freely.
3. Each robot has an ability to sense the geographic characteristics of the immediate surroundings. Particularly the robots must sense the existence of branches of a corridor.

With these capabilities, a file of robots follows the following algorithms in order to explore maze-like corridors and get out of dead-end.

3.1 Movement of a File of Robots

Figure 3 shows a corridor splitting into two branches. Both of the branches are dead-ends. A file of robots comes from the top, and enters in the left branch and checks the right branch. Upon realising both of them are dead-end, it retreats. In the figure, the dotted lines depict the corridors, and the solid line depicts the file of robots where the diamond shape depicts the leading robot and the circle depicts the rear of the file.

Figure 3b shows that the leading robot just comes to the dead-end of the left branch. Then the most conveniently located robot in the file, i.e. the robot at the junction, disengages from the line and becomes the new leading robot. The immediately preceding robot of the new leader becomes the selector (triangle shape), and the former leader becomes the end robot. Consequently, the file splits into two, and the new leader that was the latter half of the file enters the right branch while the stacked half lines of robots are left in the left branch and waiting as shown in Figure 3c. When the last robot of the file just passes the junction, i.e. in front of the selector, the selector follows the rear robot. In order to do so, the rear and the selector has the same label; ‘A’ in this case.

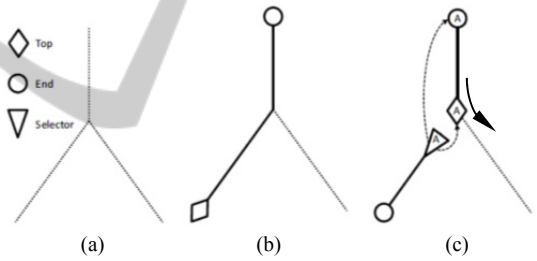


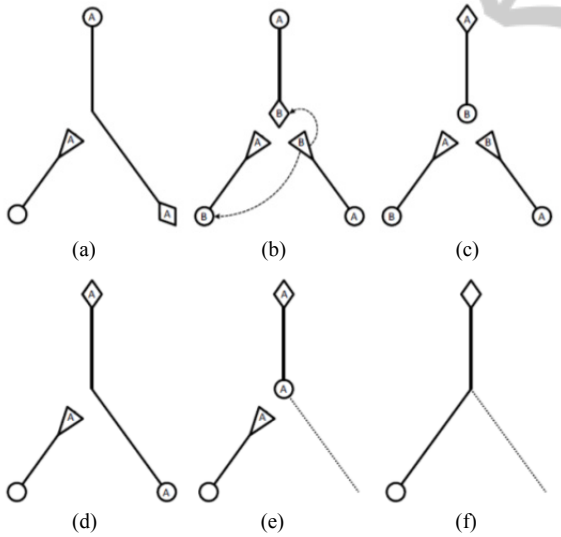
Figure 3: A queue of robots comes to a dead-end and explore the other branch.

If the right branch corridor is long enough to contain entire robots, the robots reform a file and continue the exploration. But if the corridor is also dead-end, the entire robots have to come back. In such a situation we have two cases. One is the case where the file is longer than the corridor, and the other is the case where the file is shorter than the corridor. We examine each case in turn.

First we consider the case where the file is longer than the corridor. Figure 4 shows this case. The new leading robot comes into the dead-end of the branch before the rear robot comes at the junction (Figure 4a). The file splits itself into two. One of the two robots at the junction becomes the new leader and the other becomes the selector (Figure 4b). They are labelled ‘B’. Since there is no third corridor, the new leader exchanges its role with the rear robot (Figure 4c). Note that the new leader’s label is A while the

exchanged end robot's label is B (Figure 4c). The selector B follows the end robot B to form single line (Figure 4d), then as the file recedes and when the rear robot comes to the junction, the selector A follows the end robot A (Figure 4e), so that they complete to form the entire file to go out of the corridors as shown in Figure 4f.

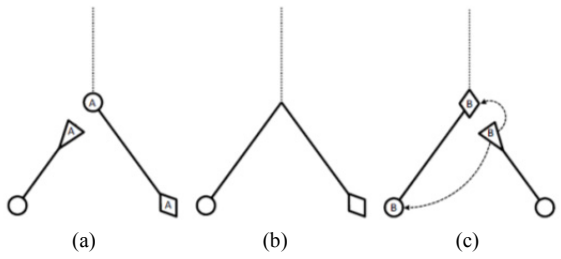
Next, we consider the case where the file is shorter than the corridor. Figure 5 shows this case. As in the first case, when the file comes into the dead-end of the left branch, it splits into two and the selector A is created at the end of the left line. The new leading robot commands the latter half of the file and enters into the right branch. At some point, the end robot comes to the junction as shown in Figure 5a. Then the selector follows the last robot of the latter half to form a complete file (Figure 5b). When the new leader comes into the dead-end of the right corridor, the file split into two lines again, and a new leader B and selector B are created (Figure 5c). The new leader retracts the file, and then the selector B follows the end robot B to form the entire file again to get out the corridors as shown in Figure 5d and 5e.



(a) (b) (c)

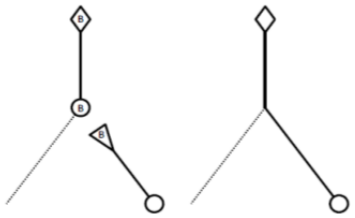
(d) (e) (f)

Figure 4: The case of the file is longer than the corridor.



(a) (b) (c)

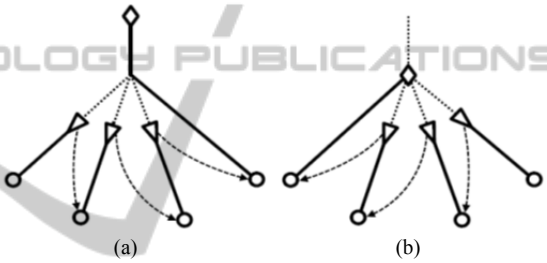
Figure 5: The case of the file is shorter than the corridor.



(d) (e)

Figure 5: The case of the file is shorter than the corridor (cont.).

The difference between these two cases is just the difference of the order of reconnecting the queues to reform the file. In the first case, the queues are connected in the reverse order in their being split, while the second case connects in the order in their being split. This algorithm of dead-end corridor exploration is applicable not only two way branches but also many ways branches. The Figure 6 shows the four ways cases.



(a) (b)

Figure 6: The four way branches. The file is longer than the route (a), and the file is shorter than the route (b).

3.2 Control Algorithm using Mobile Software Agents

In this section we describe how the algorithms explained in the previous section are implemented by the mobile software agents. We have created two types of agents: *Follower Agent* (FA) and *Pioneer Agent* (PA). The former has the role that follows the immediately preceding robot as well as the role of selector. The latter has the role of leader. As mentioned in the algorithms, the leader exchanges its role quite often. Therefore the PA is implemented as a mobile agent and migrates from a robot to another robot. Because PA migrates into a robot on which FA already resides, PA has higher priority and overrides the role of FA. We describe each of them in the following subsections.

3.2.1 Follower Agent

FA has two states; one is just following specified robot (state A), and the other is waiting until it is

notified to connect to the specified robot (state B). The base state is A. Upon receiving command from PA, FA turns to its state to B. When the connection is done, the state turns into A again.

Figure 7 shows the behaviours of FAs in pseudo code when they are in state A and B, respectively. `detect_robot()` is the function that recognizes the robot in front of the robot that FA is riding. `follow_robot` specifies the robot FA is supposed to follow, and `follow_robot_alt` specifies the selector robot.

The driving behaviours of a FA are determined by the state. FAs cannot change their own states. PA has the privilege to write variables in FAs so that PA can determine the state of a certain FA. PA also specifies to which robot (label) the FA drives to follow. As shown in Figure 7, FA cannot even turn the driving robot to the reverse direction.

The important point is that only PA can turn the driving robot other way round. Therefore in order to turn around and to retract the robot, PA has to migrate to the robot. The behaviours of PA are described in the next subsection.

```
procA () { // behaviour of state A
  r = detect_robot();
  if (r == follow_robot)
    follow ( follow_robot)
  else
    stop ()
   }
procB () { // behaviour of state B
  stop ()
  while ( state == 'B') {
    r = detect_robot();
    if (r == follow_robot
     || r == follow_robot_alt) {
      if(r is End ){
        follow_robot = r
        state = 'A'
      }
    }
  }
}
```

Figure 7: The behaviours of Follower Agents.

3.2.2 Pioneer Agent

PA makes the driving robot explore corridors. Therefore PA is always resides on the leading robot. When the robot comes to a dead-end, PA has to change the directions of the entire file as well as to exchange the roles of leading robot with rear robot.

Figure 8 shows the behaviours of PA in pseudo code. `current_robot` specifies the robot the PA currently resides. `arrive_branch()` is the function to detect a junction, `arrive_deadend()` is the function to detect a dead-end, `turn_around()` makes the current robot turn the other way round, and `migrate_to` makes the PA migrate to the specified robot designated by an IP address.

The most important behaviour is what PA has to do when `arrive_deadend()` returns `true`. First, it saves the `current_robot` to `tmp`, and moves back to the robot at the last junction by performing the function `back_to_branch()`. Note that turning the robot to the reverse direction is done in this migration function calls. Also note that when the PA comes to the junction, the two robots are facing each other. The PA needs to change the state of the FA on the robot it is facing. Thus the PA makes the FA selector (state B) by making `end_robot` and `current_robot` be `follow_robot` and `follow_robot_alt` (labelling) respectively as well as `state`. Finally it makes the `tmp` robot be the rear robot.

```
explore() {
 while (true) {
   if (arrive_deadend()) {
     tmp = current_robot
     back_to_branch()
     prev_r = detect_robot()
     prev_r->FA->follow_robot
              = current_robot
     prev_r->FA->follow_robot_alt
              = end_robot
     prev_r->FA->state = 'B'
     end_robot = tmp;
  }
  go_foward()
  }
}
back_to_branch() {
  while (!arrive_branch()){
    turn_around()
    r = detect_robot()
    FA->follow_robot = r
    migrate_to (r->address )
  }
}
```

Figure 8: The behaviours of Pioneer Agents.

4 IMPLEMENTATION BY A SIMULATOR

In order to demonstrate the feasibility of our algorithms, we have implemented a simulator for the multi-robot system to explore corridors with dead-ends. In this section, we first describe the mobile agent system we employ and then explain the simulation of robot movements.

4.1 Agent System as a Thread

In order to implement mobile agents, we employed Java threads and made them migrate. We have created `Agent` class that inherits `Serializable` interface so that the object code can be serialised. In order to implement the migration, we translate the compiled Java bytecodes so that the migrating code can save local variables, stack variables, and the program counter. Figure 9 shows a Java pseudo code example.

```
class MovableAgent extends Agent {
  run (){
    switch (loadPC()){
      case 1:
        loadLocals();
        loadStacks();
        goto L1;
        break ;
      default :
        break ;
    }
    /* do something */
    storeLocals();
    storeStacks();
    storePC(1);
    migrate(IP address);
    throw new MigrateException();
  L1:
   /* do something */
  }
}
```

Figure 9: Pseudo Java code for mobile agents.

In order to perform a migration, we make an instance of `MovableAgent` save local and stack variables before calling the function `migrate()`, set a label and save the address of the label. In this case the label is `L1`. Right before the label, the instance throws an exception and let the run-time system know the thread wants to migrate. Upon catching the exception, the run-time system serialises the instance of `MovableAgent` and sends it to the designated address. Another run-time system that receives the serialised object, restores the instance, and makes it run as a thread. When the thread resumes running, it restore the local and stack variables, loads the program counter, and jumps to L1 to complete reproduction of environment and execution of the thread at the sent site.

4.2 The Simulator

We have implemented the *Pioneer Agent* (PA) and *Follower Agent* (FA) by using the agent system described in the previous subsection. This subsection reports the result.

The Figure 10 shows the experiment of the case where the file of robots is shorter than the corridors. The file of seven robots is ready and starts marching (Figure 10a). While each robot has a FA, only the leading robot is driven by PA. When the leader finds that the right corridor is dead-end, the PA starts migrating back the robots one by one and turns them around into reverse direction (Figure 10b).

Once PA comes to the robot at the junction, the PA makes it as the new leader (Figure 10c). The second file, which consists of the latter three robots, enters into the centre corridor, while the other four robots in the right corridor are stack in the dead-end and waiting (Figure 10d). The second file soon comes into the dead-end, the PA moves back to the middle of three, and makes it the leader for exploration to the left corridor. At this time, the former leading robot is left and made waiting, and the four robots in the right corridor join to the file with the very new leader (Figure 10e). When the marching file comes into the dead-end of the left corridor, the PA changes the leading robot again, and makes the file retreat. Then, we can observe that the new file is constructed as the right queue first, then the centre robot joins, and finally the left queue follows the new file (Figure 10f).

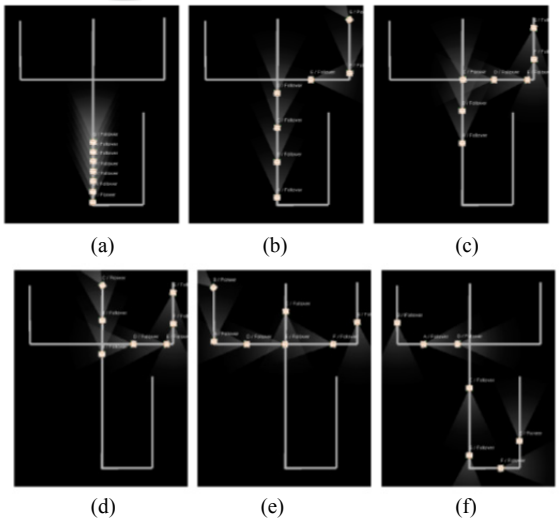
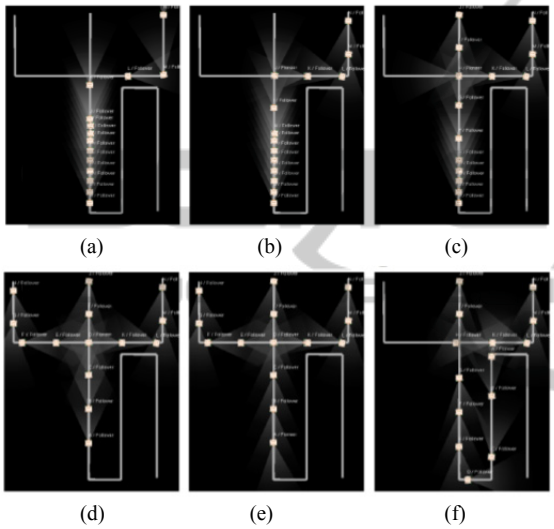


Figure 10: The case of the file is shorter than the corridors.

The Figure 11 shows the experiment of the case where the file of robots is longer than the corridors. At this time, the file consists of fourteen robots. Figures 11a through 11d show that the robots display the same behaviours as shown in Figures 10a through 10d. Upon finding the last corridor (left one)

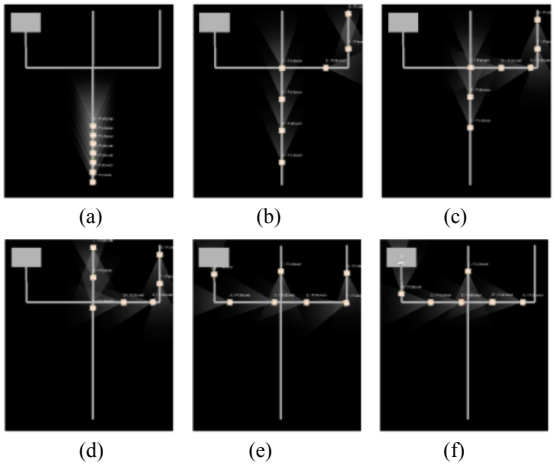
is dead-end, the PA come back to the robot at the junction (Figure 11d). Then the PA also finds there is no alternative corridors to explore and realises it has to make all the robots retreat. The PA further migrates back to the last robot and makes it the leading robot (the very bottom one) (Figure 11e). The PA drives the new leader and makes all the robots go downward. At this time, we can observe that the new file is constructed as the queue that does not enter the dead-end corridors first, then the left queue joins next, the centre queue, and finally the right queue joins to complete the file.



(a) (b) (c)

(d) (e) (f)

Figure 11: The case of the file is longer than the corridors.



(a) (b) (c)

(d) (e) (f)

Figure 12: The case of the file is longer than the corridors.

As the final example, we show the case of existence of a goal place instead of dead-end. Figure 12 shows the case. The upper left rectangle is the goal place. Robots are supposed to stop and stay when they reach there. We can observe that the file of mobile robots behaves just the same as the case where the file is shorter than the corridors until the leading robot reaches the goal place. Then the PA stops working and wait for the rest of the file the FAs drive.

We have confirmed that the algorithms we proposed for mobile software agents successfully drive a file of robots to explore maze-like corridors in both of the cases where the file is shorter than the corridors and where the file is longer than the corridors. The changes of leading robot by migrations of a mobile agent PA successfully guide the file of robots.

5 CONCLUSIONS AND FUTURE WORKS

We have presented a multi-robot system that explores maze-like environments in a file. When a file stacked in a dead-end corridor, they must come back. In order to accomplish smooth retraction, we propose mobile software agent approach that changes the leader of the file dynamically by exchaging the roles by the agents.

Dynamic exchange of roles is executed by the migration of software agents. The software agents are implemented as Java threads. In order to physically move a Java thread to another computer, we not only use Java Serializable interface but also translation of compiled byte codes so that the transmitted serialised code can be restored with local and stack variables as well as program counter.

In order to demonstrate the feasibility of our approach and algorithms, we have implemented a simulator. The simulator shows a file of mobile robots can explore maze-like corridors in both of the cases where the file is shorter than the corridors and where the file is longer than the corridors. The corridors can branch out in arbitrary numbers.

Because we could confirm our algorithm work well in the simulator, it would the natural step to implement the mobile robot system by using real robots. Since the results of simulator are simply theoretical ones and the setting is too ideal, we thought it has no point to execute numerical experiments with the simulator. Instead, we are constructing a herd of mobile robots to perform the real world experiments. The corridors are implemented as lines of silver tape commonly used for line-trace robots. The setting is just a line-trace robot exercise commonly done in any technical colleges. We are solving the problems we encounter

during the implementation one by one, and are going to show a complete multi-robot system that explores maze-like environment. The knowledge and experience we obtain through this project will contribute to the rescue robotics research community.

ACKNOWLEDGEMENTS

This work is supported in part by Japan Society for Promotion of Science (JSPS), with the basic research program (C) (No. 26350456), Grant-in-Aid for Scientific Research.

REFERENCES

Binder, W., Hulaas, J. G. and Villaz, A., 2001. Portable resource control in the j-seal2 mobile agent system. In *AGENTS '01*, *Fifth International Conference on Autonomous Agents*, pages 222–223.

Fox, D., Burgard, W., Kruppa, H. and Thrun. S., 2000. Collaborative multi-robot exploration. *Autonomous Robots*, 8(3), pages 325-344.

Fredslund, J. and Mataric, M. J., 2002. A general algorithm for robot formations using local sensing and minimal communication, *IEEE Transactions on Robotics and Automation*, 18(5), pages 837-846.

Gonzalez, E., De la Rosa, F., Miranda, A. S., Angel, J. and Figueredo, J. S., 2011. A control agent architecture for cooperative robotic tasks, In Yasuda, T and Ohkura, K. eds. *Multi-Robot Systems, Trends and Development*, InTech, Rijeka, Croatia, pages.

Ishiwatari, N., Sumikawa, Y., Takimoto, M. and Kambayashi, Y., 2014. Multi-Robot Hunting Using Mobile Agents. In *Proceedings of the Eighth KES Conference on Agent and Multi-Agent Systems (KES-AMSTA 2014)*, volume 296 of LNAISC, pages 223-232. Springer.

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

Kambayashi, Y., Yamachi, H., Takimoto, M., 2012. Feasibility Studies of the Intelligent Cart System. *Communications in Information Science and Management Engineering*, 2(6), 1-8.

Kambayashi, Y., Shinohara, T., Takimoto, M., 2014. Self-Optimizing Algorithms for Mobile Ad Hoc Networks Based on Multiple Mobile. In *Proceedings of the 6th International Conference on Agents and Artificial Intelligence (ICAART 2014)*, pages 156–163. SciTePress.

Nagata, T., Takimoto, M. and Kambayashi, Y., 2013. Cooperatively Searching Objects Based on Mobile Agents, *Transaction on Computational Collective Intelligence XI*, volume 8065 of LNCS, pages119-136. Springer.

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.

Roumeliotis, S. I. and Bekey, G. A., 2002. Distributed multirobot localization, *IEEE Transactions on Robotics and Automation*, 18(5), pages 781-795.

Shibuya, R., Takimoto, M., and Kambayashi, Y., 2013. Suppressing energy consumption of transportation robots using mobile agents. In *Proceedings of the 5th International Conference on Agents and Artificial Intelligence (ICAART 2013)*, pages 219–224. SciTePress.

Shintani, M., Lee, S., Takimoto, M., Kambayashi, Y., 2011. A Serialization Algorithm for Mobile Robots Using Mobile Agents with Distributed Ant Colony Clustering. In *Knowledge-based and Intelligent Information and Engineering Systems*, *(KES 2011)*. volume 6881 of LNAI, pages 260-270. Springer.

Shintani, M., Lee, S., Takimoto, M., and Kambayashi, Y. 2011b. Synthesizing pheromone agents for serialization in the distributed ant colony clustering. In *Proceedings of the International Conference on Evolutionary Computation Theory and Applications and the Proceedings of the International Conference on Fuzzy Computation Theory and Applications*, pages 220–226. SciTePress.

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

Takahashi, R., Takimoto, M., and Kambayashi, Y. 2014. Cooperatively Transporting Unknown Objects Using Mobile Agents, In *Proceeding of the 6th International Conference on Agents and Artificial Intelligence (ICAART 2014)*, vol. 2, pp.60-68. SciTePress.

Takimoto, M., Mizuno, M., Kurio, M., and Kambayashi, Y. (2007). Saving energy consumption of multi-robots using higher-order mobile agents. In *Proceedings of the First KES Symposium on Agent and Multi-Agent Systems (KES-AMSTA 2007)*, volume 4496 of LNAI, pages 549–558. Springer.

Vaughan, R. T., Stoy, K., Sukhatme, G. and Mataric, M.J., 2002. LOST: Localization-Space Trails for Robot Teams, *IEEE Transactions on Robotics and Automation*, 18(5), pages 796-812.

Yasuda, T. and Ohkura, K., 2005. Autonomous role assignment in a homogeneous multi-robot systems, *Journal of Robotics and Mechatronics*, 17(5), pages 596–604.