A Decentralized Cooperation Protocol for Autonomous Robotic Agents

Fang-Chang Lin and Jane Yung-jen Hsu
Department of Computer Science and Information Engineering
National Taiwan University, Taipei, Taiwan, R.O.C.
{fclin,yjhsu}@robot.csie.ntu.edu.tw

Abstract

This paper proposes a decentralized cooperation protocol for an object-sorting task in a distributed robotic system. The multiple agents are based on a homogeneous agent architecture that consists of search, motion, and communication modules coordinated through a global state. Advantages of the system architecture are simplicity, intra-agent distributed control, no explicit inter-agent control and flexibility. The protocol encourages the agents to help each other in order to facilitate overall task achievement. Simulation results showed that 1)The protocol is stable and reliable under a spectrum of workload. 2)The execution time increases linearly with the number of objects. 3)Increasing the number of agents can significantly decrease the waiting time and improve the performance.

1: Introduction

A multi-agent robotic system uses multiple robots to solve problems by having them work in parallel. Situations found in space exploration, undersea construction, nuclear waste management, explosives detection and many others often require a multi-agent system that can achieve a common task by coordinating their behaviors. Typical application tasks include retrieval, simple construction, routine cleaning and finishing, etc.

Much research on multi-agent robotic system has begun to emerge. Fukuda's CEBOT system[7] demonstrated the self-organizing behavior of a group of heterogeneous robotic agents. Beni and Hackwood's research on swarm robotics demonstrated large scale cooperation in simulation[8]. Brooks et al.[5] developed the lunar base construction robots by using a set of reactive rules. Mataric[9] also studied task performance in a group of mobile robots based on the subsumption architecture[6]. Arkin has demonstrated that cooperation between robotic agents is possible even in the absence of communication[2]. It simplifies the design of an agent because there is no communication between agents. On the other hand, it may be inefficient due to the lack of communication. Arkin et al.[3,4] assessed the impact on performance of a society of robots in a foraging and retrieval task when simple communication was introduced.

This paper is motivated from the work by Arkin et al.[2, 3, 4]. Section 2 outlines their original work and discusses several improvements. The object-sorting task is introduced in Section 3, and the system assumptions are described in Section 4. Section 5 proposes the system architecture and cooperation protocol, followed by the simulation results in Section 6.

2: Motivation

The work by Arkin et al.[4] is based on the foraging and retrieval behavior of the ants. By using the AuRA architecture(Autonomous Robot Architecture)[1], the agent searches randomly for targets and moves them to the base location without communication. If two agents are moving the same target, the moving speed is doubled. Their work also proposed a limited communication scheme by using a shared memory. The current state and coordinates of any agent that is moving a target will be recorded at every time step. Such information enables an foraging agent to locate and help out the nearest busy agent, thus improving the overall performance.

This paper addresses the following problems in their work.

- The termination condition is not well-defined. The agents are not guaranteed to search the entire area by using random search approach.
- An agent is assumed to be able to move any kind of target and the speed is proportional to the number of moving agents. More realistically, a target may need more than one agent to move it.
- Recording the states and coordinates of the agents at every step may cause a large overhead and communication congestion.

It is therefore necessary to have a cooperation protocol that allows multiple agents to help each other in the

0-8186-7087-8/95 $4.00 © 1995 IEEE

problem solving process. The protocol should include the following dimensions:

1) A *communication* architecture that an agent can communicate with the other agents to request for help or to offer help.
2) A *when-help* strategy that determines when to help the other agents.
3) A *select-help* strategy that determines which agent needs help most. When an agent can help the others, maybe many agents need help.
4) A *load-balancing* strategy that balances the system load among all the agents.
5) A *blocked-free* strategy to detect or prevent the system from blocked. When all the agents need help, the system is blocked. The *blocked-free* strategy prevents the system from blocked, or detects and breaks down the blocked state.

3: The object-sorting task

This section formally defines the object-sorting task. Let $O=\{o_1,...,o_M\}$ be a set of objects that is randomly distributed in a bounded area A. Every object, $o_i=(l_i,d_i,n_i)$, is associated with an initial location l_i, a destination location d_i, and the number n_i of agents for moving it. The destination location d_i specifies the location to which the object should be moved. An object can only be moved if there are at least n_i agents available to move it. Therefore, an object-sorting task cab be completed only if the total number of agents N is larger than or equal to the maximal number n_{max} of agents to move any object. The agents must search for the objects and move them to their destinations as specified. When all the objects have been moved to their destinations, the task is finished.

For load-balancing consideration, this paper uses a uniform distribution model with a cooperation protocol to do the task. The agents, $R=\{r_1,...r_N\}$, are uniformly distributed into the area $A=\{a_1,...,a_N\}$, and each agent r_i has its duty subarea a_i. The agents search for the objects in their subareas and move them to the destinations. When an agent finds an object requiring more than one agent to move it, the agent must call for help from the other agents in order to move the object. In our system, the agent will broadcast a *help* message and follow a protocol to communicate with the other agents for the help. When all the agents have finished their subtasks, the object-sorting task is finished.

4: Assumptions

The system assumptions are the following:

1. The agents in our system are mobile robots and they are homogeneous.
2. The agents have no prior knowledge about the environment. They know the environment around them only after they have detected it.
3. The agents can communicate with the others by a broadcast channel or a point-to-point channel.
4. Every agent is autonomous and has the basic capabilities:
   - object identification. It can identify the other agents, the obstacles, and the target objects.
   - navigation and obstacle avoidance. It can move from one location to a specified location.
   - object movement. It can move the object alone or with the other agents.
5. The objects are stationary.

5: Overview of cooperation architecture

In our system, the architecture of each agent is as showed in Fig. 1. The search module, communication module and motion module are all finite state automata. They share the global state information and change the state information according to their state functions. Its state transition diagram is as showed in Fig. 2, and the state diagrams of each module are showed in Fig. 3.

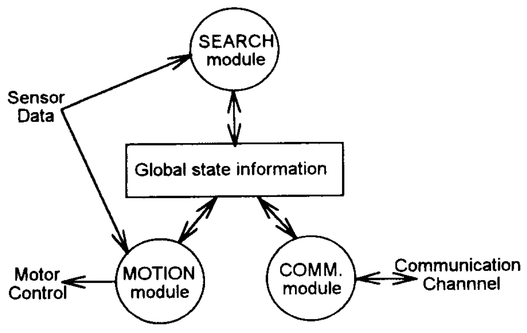

Fig. 1: Agent architecture

5.1: Search module

The search module searches for the objets and identifies their destinations and required number of agents. An object is a small object if it can be moved by an agent alone. An object is a large object if it requires more than one agents to move it.

An agent broadcasts a *help* message when it finds a large object and we call it the helped agent. The other agents coming to help the helped agent are called the helping agent. The search module searches for the objects and identifies their destinations in the *SEARCHING* state. It changes the state to *MOVING* if the object is small, or to *WAITING* and broadcasts a *help* message if the object is large.

Because the agents are independent, they don't know in which situation the others are. When an agent has completely searched its subarea, it does not mean the system task is done. It is necessary to have a protocol to let the agents know when the task is done. This will be described in Section 5.3.

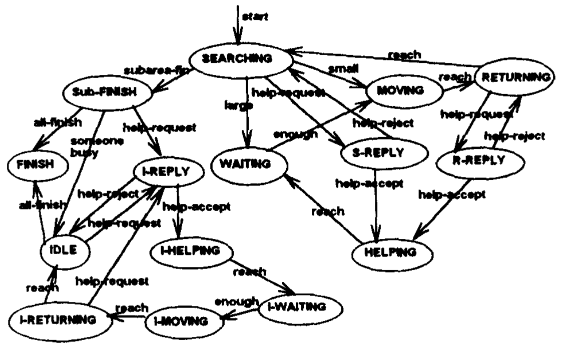

Fig. 2: State transition diagram

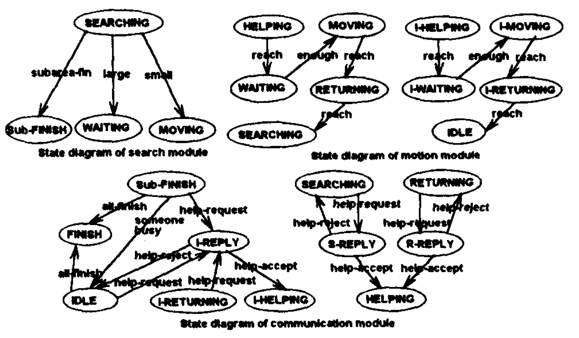

Fig. 3: State diagrams of the modules

5.2: Motion module

The motion module consists of the motion function and the object movement function. The object movement function performs the action that moves an object alone or with other agents. The motion function performs the capability of moving to goal.

The motion module moves an object to its destination in the *MOVING* state, and changes the state to *RETURNING* after having moved the object to its destination. When in *RETURNING* state, the agent returns to its subarea and the motion module changes the state to *SEARCHING* when reaches the goal location. A helping agent in *HELPING* state is moving to the helped agent and will change the state to *WAITING* when it reaches the helped agent. If there are enough agents to move the objects, the agents in *WAITING* state will change the state to *MOVING*.

After an agent has finished its subtask, it enters *IDLE* state and returns to its base location. The agent will be in *I-HELPING* when it is moving to the helped agent, and will change the state to *I-WAITING* when it reaches the helped agent. If there are enough agents to move the objects, the agent in *I-WAITING* state will change the state to *I-MOVING*. When it has moved the object to the destination, the agent comes into *I-RETURNING*, then the state is changed to *IDLE* when it has returned to its base location.

5.3: Strategies of cooperation protocol

Most of the cooperation protocol is embedded in the communication module. Before describing the module, we focus on the strategies of the protocol.

The *when-help* strategy is based on the principle that an agent will help the others when it is not busy. That is, an agent may accept a *help* message only when it is in *SEARCHING*, *(I-)RETURNING* or *IDLE* states, otherwise the message is queued for later processing.

The *select-help* strategy is designed by the rule that an agent select the nearest to offer help. An agent that responds to a *help* message is called a will-help agent.

The *load-balancing* strategy is considered in two parts. One part is the equal partition of the area and the uniform distribution of the agents. Another part is the negotiation process for establishing cooperation, especially the selection from the will-help agents. When an agent needs help from n agents, there may be m will-help agents such that m is greater than n. The *load-balancing* strategy enables the agent to select the first n nearest agents and send an *accept* message to each of them. The other will-help agents will be rejected by *reject* messages.

The *blocked-free* strategy is designed by the following algorithm. First, the agents in *(I-)WAITING* state detect the blocked situation by checking a timeout event, then they broadcast *blocked* message to exchange their state information and break down the blocked situation by comparing their priorities. The lower priority agents will exit their *(I-)WAITING* state by entering *(I-)HELPING* state and go to help the highest priority agent. After the lower priority agents return to their subarea, they can continue their suspended task. According to Theorem 1, the timeout event occurs when the agent stays in *(I-)WAITING* state for $2(N-1)MTT$ long.

An additional problem is task termination condition. When an agent has completely searched its subarea, it broadcasts a *sub-fin* message and come into *Sub-FINISH* state. Any other agent whose subtask is not finished will reply a *busy* message to the sub-finish agent. If the agent in *Sub-FINISH* receives a *busy* message, it enters into

IDLE state, otherwise, it means all the subtasks are finished and the agent broadcasts an *all-finish* message. All the *IDLE* agents come into *FINISH* state after receiving the *finish* message and the system task is finished.

Theorem 1. Let N be the number of agents, and *MTT* be the *maximum travel time* between any two locations in the bounded area. When an agent stays in *(I-)WAITING* state for $2(N-1)MTT$, the system is blocked.
Proof:

Consider the situation that there is only one large object found at a time. The system will not be blocked according to the *when-help* strategy. If the system would become blocked in the future, there must exist at least two large objects found at the same time.

Let $L_t=\{o_i | o_i \in O, r_j \in R$, s.t. o_i was found by r_j at time $t\}$be a set of large objects found at time t. And $W_{t,u}=\{o|\ o \in L_t$, some agent is in *WAITING* or *I-WAITING* state at time $t+u$ due to $o\}$, so $W_{t,0}=L_t$. For simplicity, $W_{t,u}$ is referred to as W_u in the following discussion. Let

MOVE=$\{r \in R$, r's state is *MOVING* or *I-MOVING*$\}$
HELP=$\{r \in R$, r's state is *HELPING* or *I-HELPING*$\}$
WAIT=$\{r \in R$, r's state is *WAITING* or *I-WAITING*$\}$
AVAIL=$\{r \in R$, r's state is *SEARCHING* or *RETURNING* or *I-RETURNING* or *IDLE*$\}$

If the system would not become blocked, not all of AVAIL, MOVE, and HELP sets are empty. Given $|W_u|>0$, we consider the agents in these three sets.

1)HELP set:
When an agent in HELP set reaches the object, either it enters MOVE set if there are enough agents to move the object or enters WAIT set. Because an agent stays in HELP state no more than one *MTT*, the maximum time for these agents to reach and move an object in W_u is one *MTT*.

2)AVAIL set:
Every agent in the set is a will-help agent. If the agents in AVAIL are accepted by the agents requiring help, they enter HELP set. So the maximum time for these agents to reach and move an object in W_u is one *MTT*.

3)MOVE set:
Because an agent stays in MOVE and enters AVAIL set no more than one *MTT*. After that, they enter HELP set immediately if there is any agent requiring help. So the maximum time for these agents to reach and move an object in W_u is 2*MTT*.

So the maximum time for the agents not in WAIT to reach and move an object in W_u is 2*MTT*. On the other hand, they all enter WAIT set if they cannot move any object in W_u in 2*MTT*. That is, the system is blocked if $|W_{u+2MTT}| = |W_u|$.

If the system is not blocked, at least one large object will be moved for every 2*MTT*. It takes at most $2(N-1)MTT$ to reduce every object in W_0.

So if an agent stays in *(I-)WAITING* state for $2(N-1)MTT$, all the agents must be in *(I-)WAITING* state and the system is blocked.

5.4: Communication module

The communication module performs the communication function and follows the protocol in order to cooperate with the others. It receives the messages sent by the other agents and replies them if necessary.

When receiving a *help* message in *SEARCHING* or *RETURNING* states, the comm. module replies a *will-help* message and changes its state to *S-REPLY* or *R-REPLY*. If the helped agent accepts the *will-help* message, it sends an *accept* message to the helping agent, otherwise, it sends a *reject* message to reject the help. When receiving an *accept* message, the communication module changes the state to *HELPING*. When receiving a *reject* message, the communication module returns to its previous state. The helping messages mechanism looks like the Contract Net Protocol[10], however, it is designed for simplicity and effectiveness.

An agent in *IDLE* state stays in its base location and waits for *help* message in order to help the others. The help mechanism is the same as described above.

5.5: Advantages

The advantages of the architecture are discussed as the following:

- *Simplicity*. The global state function is very complex. Comparing with a central module controlling all the state transition, the architecture is more simple ,clear and fast to implement it.
- *Intra-agent distributed control*. Functional separation for each agent makes the intra-agent distributed control. All state transitions are distributed into the modules and a single transition is done only by one module. The modules can run concurrently or simultaneously based on multiprocessor or uniprocessor. The communication between modules is through the shared global state information.
- *No explicit inter-agent control*. An agent communicates with the other agents through the communication system. The inter-agent control can be designed for different purpose. It may be central control if the agents are organized into a hierarchical organization. On the other hand, it is distributed control if every agents is independent.

- *Flexibility*. It is ease to add a new functional module to the agent architecture. The architecture can be extended to handle other function if the corresponding module is included.

6: Simulation

The object-sorting task was simulated in a multi-strategy simulator developed on Sun workstation and with graphic user interface showing the task execution process. The simulator is a testbed for testing different strategies of the cooperation protocol on the object-sorting task.

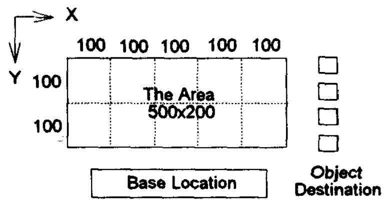

Fig. 4: The simulation map for 5x2 partition

6.1: Simulation Environment

The task is performed on a bounded area which is equally partitioned into the subareas, and each subarea is assigned to an agent by the *load-balancing* strategy. The area is represented in a two-dimensional coordinate system, e.g. 500×200. The partition of the area is represented by a $m \times n$ notation, m in x-axis direction and n in y-axis direction. For example, 5×2 partition means that the area is equally partitioned into 5 subparts in x-axis direction and 2 subparts in y-axis direction, as shown in Fig. 4. In the experiment, the area is 500×200 and partitioned into different size under different number of agents. Fig. 4 shows the map used in the simulation for N=10 agents. The partitions for different number of agents N used in the experiment are shown in Table 1.

N	1	2	4	8	10	20	30	40	50
Partition	1x1	1x2	2x2	4x2	5x2	5x4	6x5	8x5	10x5

Table 1: Partition and the number of agents

The agent searches for the objects in its subarea and move the objects to their destinations. The following code fragment describes the actions performed by the agent at each time step.

```
for each agent $i$ ,
        do the search module of agent $i$;
        do the motion module of agent $i$;
        do the comm. module of agent $i$.
```

When all the agents has completely searched their subareas, the task is finished.

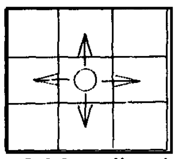

Fig. 5: Move directions

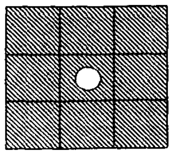

Fig. 6: Sensor ranges

The area is a grid area, and the agents can only move to one of the four locations from a location in a time unit as the figure 5. The agents can sense the objects located in the grids adjacent to the agents as the figure 6. The agents may use any exhaustive search method to search their subarea. In this experiment, the row-major order method was used. They start search from the initial location, the (0,1) location of their subarea, to the right side boundary, and search the first three rows which are under the sensor range. When they have reached the right side boundary, they search the second three rows and change the search direction from right to left. When they has reached the left side boundary, they change the search direction from left to right and start searching the next three rows. The search repeats until the subarea has been searched.

The object is randomly generated for its destination, initial location, and the number of required agents. This experiment generates 10 sets of the object for each number of the objects M using n_{max}=10. All of the object sets are run for N=10,20,30,40, and 50. When the object sets are applied to N=1,2,4, and 8, then n_{max} is set to N and all the n_i are proportionally adjusted in order to accomplish the task.

6.2: Simulation Results

The experiment was run by varying the number of agents N, and the number of objects M. For a given number M of objects, the execution time is the average of the execution time run from the 10 generated object sets of M. The performance of the proposed cooperation protocol is evaluated by comparing the execution time under different number of agents and different number of objects. The actual execution time performed by the cooperation protocol is represented by C. The reference execution time are the estimated upper bound, U, and the estimated lower bound, L.

The estimated upper bound is the execution time based on the central-controlled model that all the agents work together to do the task, that is, all the agents go together to search the area, when find an object, they move the object to its destination, then return to their previous location and continue searching. The task is executed as if there was a very powerful agent to do the task so that it could move any found object. However, the objects are moved sequentially and centrally. The estimated upper bound is the search time plus all the object-moving time.

$U = TS + \Sigma_j (TM_j + TR_j)$

where

U: estimated upper bound.

TS: time required to search the entire area.

TM_j: time required to move object j to its destination.

TR_j: time required to return to the initial location of object j from the destination of j.

The estimated lower time is the execution time under the operation model that the other agents are always available for helping when an agent finds a large object. The other agents is assumed to be available in the central point of their subareas. An agent chooses the helping agents by the *load-balancing* strategy which chooses the agents closer to it. The execution time of each agent is its subarea search time and all the processing time for the objects located in its subarea. The processing time of the object j located in the agent i is the wait time for all the helping agent coming in and the object-moving time for object j, and the return time from the destination of the object j to the initial location of j. So, the estimated lower bound is the maximum time among the execution time of all the agents.

$L = \mathrm{Max}_i \; TA_i$

$TA_i = TS_i + \Sigma_j TO_{ij}$

$TO_{ij} = TW_{ij} + TM_{ij} + TR_{ij}$

where

L: estimated lower bound.

TA_i: the total execution time of agent i.

TS_i: time for agent i to search its subarea.

TO_{ij}: the processing time of object j located in the subarea of agent i.

TW_{ij}: the maximum wait time of object j for all the helping agents of agent i.

TM_{ij}: the time that agent i moves object j to its destination

TR_{ij}: the time that agent i returns from the destination of object j to the previous location.

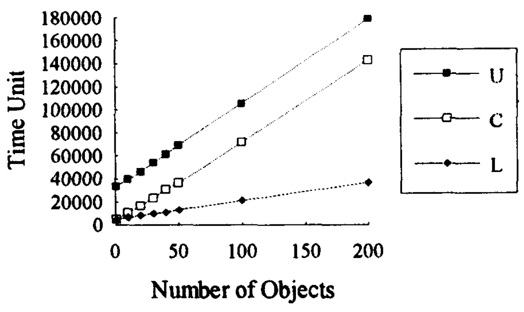

Fig. 7: The execution time of N=10

At first, the actual execution time, C, is compared with two types of estimated execution time, an estimated lower bound and an upper bound. Fig. 7 shows the relationship between the execution time and the number of objects for N=10 agents. Intuitively, as the number of objects to be moved increases, the execution time, either estimated or actual, will increase. The upper bound model always needs the most time to do the task because it moves the objects sequentially. The cooperation protocol is stable, even under heavy workload situation, e.g. 200 objects.

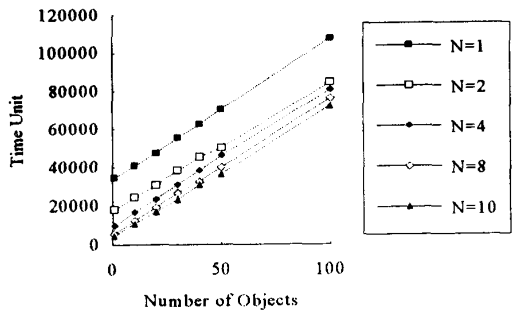

Fig. 8: The execution time of $N \leq 10$

When the workload is light, the performance of the cooperation protocol approaches the lower bound. The agents search their subareas simultaneously and move objects simultaneously if they don't need to wait for the others. It takes less time to wait for help under light workload. Under heavy workload, there are more objects in the area and the agent must take more time to wait for help. Because every agent is mostly busy, the degree of concurrence is decreased. This effect will cause the agent gradually to group together to move the objects. If most of the objects require a large number of agents to move, the system behavior will approach the upper bound in which objects are moved one by one.

Second, the execution time for different number of agents under the same object sets are compared. Fig. 8 shows the execution time for N=1,2,4,8 and 10. The workload is proportionally adjusted for any M by adjusting the number of required agents for each object. When the number of objects increases, the execution time increases linearly with it for all N. It shows the cooperation protocol is very stable under different workloads. When the estimated upper bound and lower bound are considered again, we find that U, C and L overlap when N=1. It is very clear that both the upper bound model and the lower bound model work the same as the proposed model if there is only one agent to do the task and n_{max}=1.

Finally, Fig. 9 shows the speedup effect of increasing the number of agents to do the task. When the generated object sets are applied to $N \geq 10$, the execution time decreases with the increasing number of agents. It shows that the protocol can effectively utilize the benefit from increased agent-power. The speedup is high when N is changed from 10 to 20. Meanwhile, it is low when N is changed from 40 to 50, which can be explained from Fig. 10.

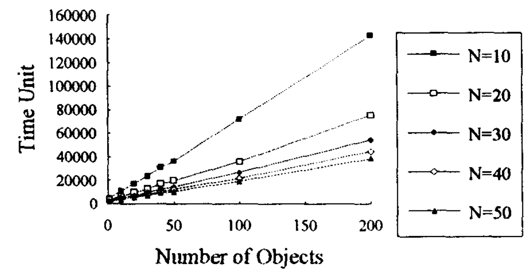

Fig. 9: The execution time for N=10,20,30,40 and 50

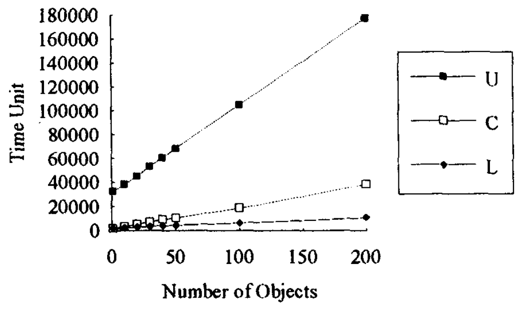

Fig. 10: The execution time of N=50

Fig. 10 shows the U, C and L for N=50 agents under different number of objects. When comparing Fig. 10 with Fig. 7, it shows that C approaches L for N=50 even on heavy workload, M=200 objects. It means that the more agents to do a task, the faster it is finished. Initially the waiting time is significantly decreased when using more agents. That is, C approaches to L. If there are usually enough agents available to help out whenever a large object is found, the benefit from increasing N will be a smaller subarea for each agent.

7: Conclusions

In this paper, we focus on the problem of an object-sorting task in multi-agent robotic system and present the system architecture of the agent, the global finite automata and the finite automata of each module, a decentralized cooperation protocol and the strategies for doing the object-sorting task. A multi-strategy simulator is developed as a testbed for testing the different environments and strategies on the cooperation protocol. The experimental results show that the cooperation protocol has stable and reliable behavior under different workloads. Increasing the number of agents will decrease the waiting time of agents and speedup the execution time.

Using the simulator, further experiments will be conducted in order to analyze the effects of different protocols, strategies and object distribution.

8: References

[1] R. C. Arkin, "Motor Schema-Based Mobile Robot Navigation", *International Journal of Robotics Research*, Vol. 8, No. 4, August 1989, pp. 92-112

[2] R. C. Arkin, "Cooperation without Communication: Multi-agent Schema Based Robot Navigation", *Journal of Robotic Systems*, Vol. 9(3), April 1992, pp. 351-364.

[3] R. C. Arkin and J. D. Hobbs, "Dimensions of Communication and Social Organization in Multi-Agent Robotic Systems", *Proc. Simulation of Adaptive Behavior 92*, Honolulu, HI, Dec. 1992.

[4] R. C. Arkin, T. Balch and E. Nitz, "Communication of Behavioral State in Multi-agent Retrieval tasks", *Proc. of 1993 IEEE International Conference on Robotics and Automation*, GA, May 1993.

[5] R. A. Brooks, P. Maes, M. Mataric and G. More, " Lunar Base Construction Robots", *IEEE International Workshop on Intelligent Robots and Systems (IROS '90)*, pp. 389-392, Tsuchiura, Japan, 1990.

[6] R. A. Brooks, "A Robust Layered Control System For A Mobile Robot", *IEEE Journal of Robotics and Automation*, Vol. RA-2, No. 1, March 1986, pp. 14-23.

[7] T. Fukuda, S. Nakagawa, Y. Kawauchi, and M. Buss, "Structure Decision for Self Organizing Robots Based on Cell Structure - CEBOT", *Proc. of IEEE International Conference on Robotics and Automation*, Scottsdale Arizona, pp. 695-700, 1989.

[8] S. Hackwood and S. Beni, "Self-organization of Sensors for Swarm Intelligence", *Proc. of 1992 IEEE International Conference on Robotics and Automation*, Nice, pp. 819-829, 1992.

[9] M. Mataric, "Minimizing Complexity in Controlling a Mobile Robot Population", *Proc. of 1992 IEEE International Conference on Robotics and Automation*, Nice, pp. 830-835, 1992.

[10] R. G. Smith, "The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver", *IEEE Transaction on Computers*, vol. C-29, No. 12, Dec. 1980.