DESIGN OF THE PEER AGENT FOR MULTI-ROBOT COMMUNICATION IN AN AGENT-BASED ROBOT CONTROL ARCHITECTURE

By

Anak Bijayendrayodhin

Thesis

Submitted to the Faculty of the

Graduate School of Vanderbilt University

in partial fulfillment of the requirements

of the degree of

MASTER OF SCIENCE

in

Electrical Engineering

May, 2002

Nashville, Tennessee

Approved By

Kazuhiko Kawamura, Professor

D. Mitch Wilkes, Professor

ACKNOWLEDGEMENTS

I would like to thank Dr. Kazuhiko Kawamura for the privilege of working in the Intelligent Robotics Laboratory. This opportunity has allowed me to learn a great deal as well as participate in many exciting research projects. I thank Dr. Mitch Wilkes for his guidance and support throughout this research.

I thank everybody in the Intelligent Robotics Laboratory especially in the Mobile Robots Group, Kanok, Chai, Siripun, Surachai, Palis, Jian, Bugra, Carlotta, Imtiaz, and Kelly for their valuable supports and knowledge. Without them, I would not have been here today.

Thank you Flo, without you my thesis would not have looked this good.

I thank my very special friend, Mitsara, whose encouragements mean more than anything in the world to me.

Finally I thank my family for always be there to support me through many tough times.

TABLE OF CONTENTS

Page

ACKNOWLEDGEMENTS ii

LIST OF FIGURES v

LIST OF TABLES viii

Chapter

I. INTRODUCTION 1

II. RELATED WORK 3

Multi-Agent Robotics Systems 3
Evolution of Communication 3
Cooperative Communication and Knowledge Sharing 4

III. BACKGROUND 6

The Intelligent Machine Architecture 6
Self Agent 9
The Commander Interface Agent 10
Sensory EgoSphere 10
Mobile Robots 12

IV. CONCEPTS OF PEER AGENT 23

Overview 23
Peer Agent Architecture 25
Peer Agent Manager 28
Peer Agent Client 30
Peer Agent Manager and Peer Agent Client Interactions 32
Applications of the Peer Agent 33

V. KNOWLEDGE SHARING USING PEER AGENTS 35

Objective of Experiment 35
Experiment Setup 35

VI. CONCLUSIONS 61

Appendices

A. AGENTS USED IN THIS EXPERIMENT 64

IMA Agents for Skeeter 64
IMA Agents for Scooter 65

IMA Agent for Human Commander 65
B. DETAIL OF AGENTS 66
The Peer Agent Manager 66
The Peer Agent Client 68
The SAN Manager 70
SAN Nodes on Skeeter 72
SAN Nodes for Scooter 76
Commander Interface Agent 79
C. TEST RUN DATA 81
REFFERENCES 83

LIST OF FIGURES

Figure Page

1. IMA Agents Overall Architecture 8
2. Interaction between Self Agent and other agents 9
3. Sensory EgoSphere of the robot 12
4. Pioneer 2-AT robot from ActivMedia Robot, Inc. 13
5. Basic Body Structure of the robot 14
6. Front Sonar array of robot 15
7. (a) Front view of the head, (b) Back view of the head 15
8. Modified version of the robot 16
9: Sonar and LASER ranging sensors on Scooter 18
10. (a) Scooter with factory setup, (b) With our modifications 20
11. IMA Agent-based HRI Approach 21
12. Communication within a mobile robot group 22
13. Robot group consisting of robots, Commander Interface Agent and Users 24
14. Peer Agent Architecture 26
15. Multi-Agents Peer connection 27
16. Special communication channel between Peer Managers and Commander Interface Agent 28
17. Communication channels within Peer Agents 29
18. Peer Manager User Interface 30
19. Peer Agent Client's User Interface (Unconnected Version) 31
20. Peer Agent Client's User Interface (Connected Version) 32
21. Applications of the Peer Agent 33
22. Many ways to setup the environment for knowledge sharing demonstration, which results in the same SES data 36
23. Scenario for Knowledge Sharing Demonstration 38
24. A SES agent on Skeeter 39
25. SES data flow between Skeeter and Scooter 40
26. Heading calculation for Scooter 41

27. Evaluation setup for system performance analysis 43
28. Three different Scooter position, (a) at 45, (b) at 90, (c) at 135 degrees 44
29. Graphs plot from performance index at 45 degrees 47
30. Scooter's trajectory from inside the SES circle..... 48
31. Scooter's trajectory when located around the perimeter of the SES circle 48
32. Scooter's trajectory when located outside the SES circle 49
33. Graphs plot from performance index at 90 degrees 50
34. Scooter's trajectory when located inside the SES circle 51
35. Scooter's trajectory when located around the perimeter 51
36. Scooter's trajectory when located outside the perimeter..... 52
37. Graphs plot from performance index at 45 degrees 53
38. Scooter's trajectories when, (a) Larger distance between Skeeter-target, and (b) Smaller distance between Skeeter-target 54
39. The heading calculation agent..... 55
40. Commander Interface Agent Window 56
41. Data flow within the robot group 57
42. Skeeter's SAN Network 58
43. Scooter's SAN Network..... 59
44. Layout of the communication network 60
45. Source addresses page..... 66
46. Setup page 67
47. Links page 67
48. The Peer Agent Client status page 68
49. Broadcast channel page..... 68
50. Path page of the Peer Agent Client 69
51. Manager page of the Peer Agent Client 69
52. Status page..... 70
53. SAN Manager..... 71
54. SAN Manager for Skeeter 71
55. SAN Manager for Scooter..... 72
56. TimeOut Node's interface..... 73

57. Detect Node interface 73
58. Mission Accomplished Module interface 74
59. Mission Failed Module interface 74
60. Changing between nodes 75
61. Scan Node example on Skeeter 76
62. Agent that communicate with Scooter' processor via TCP/IP 77
63. The Heading Agent graphical interface 78
64. Heading Calculation Agent user interface 79
65. The Commander Interface Agent 80

LIST OF TABLES

Table Page

1. Performance index at angle of 45 degrees....46
2. Performance index at angle of 90 degrees....49
3. Performance index at angle of 135 degrees....53
4. Data at 45 degrees angle....81
5. Data at 90 degrees angle....81
6. Data at 135 degrees angle....82

CHAPTER I

INTRODUCTION

The role of communication among mobile robots remains one of the most important research issues in multi-agent robotics system design. There have been many research groups that studied the cooperation among robots [1,6,9,20,31]. When a task requires cooperation, there is a need for some form of communication between the participating agents. Cooperation work requires communication whenever one agent's actions depend critically on knowledge that is accessible only from other agents.

What kind of information robots should share with each other will depend on the task to be performed. Raw sensory data such as sonar or LIDAR data can be shared, but with a group of heterogeneous robots, raw information will have to be translated and modified into a syntax that other robots can readily understand. Otherwise, this information is not going to be very useful to achieve the task. A more practical way of sharing sensory information is to transform raw sensory data into abstract data that each robot in the group can easily understand. This thesis will focus on sharing the Sensory EgoSphere (SES) or the Landmark EgoSphere (LES) that will be describe in details in Chapter III.

From the software side, states, intentions and behaviors of robots can be shared among the group. This has many advantages, if each member of the group is able to sense other members' states, intentions or behaviors, it can modify its own behavior to adapt with the current situation. This can increase the group's performance in achieving group task.

In the Intelligent Robotics Laboratory (IRL) at Vanderbilt University, we are currently developing a method for mobile robots to communicate and share knowledge. We have developed the concept of the Peer Agent that will enable robots to exchange information, raw sensory data, abstract sensory data such as SES or LES and the robot's state information. between one another. This Peer Agent will manage all the communications in and out of the robot and also monitor the condition of each existing communication channel. We will show in chapter V that knowledge can be shared between robots via the Peer Agent and this will help a robot group to achieve its given task more efficiently.

Related work in the multi-robots field and multi-agent communication will be presented in Chapter II. In Chapter III, the background of the thesis such as the Sensory EgoSphere concept and

the Spreading Activation Network concept will be presented. Chapter IV will introduce the concept of Peer Agent, how it works and what kind of tasks can be used with it. An experiment involving the Peer Agent in a multi-robot knowledge-sharing scenario will be described in detail in Chapter V. Finally, Chapter VI will present the conclusions.

CHAPTER II

RELATED WORK

This chapter describes related work in multi-robotic field and what other researchers are currently experimenting involving the multi-robot communication.

Multi-Agent Robotics Systems

Multi-agent robotics systems constitute a large area of current robotic research. Fukuda [2] was among the first to study multi-agent robotic systems. His work is mainly concerned with a group of heterogeneous robots. Cao et. al. [3] described the communication structure for inter-agent interaction and characterized several major types of interactions that can be supported. Gage [4] discussed communication issues relevant to a system consisting of an arbitrary large number of autonomous robots. Mataric et. al. [5, 6, 7], proposed a behavior-based architecture for synthesis of collective behaviors such as flocking, foraging, and docking, based on the direct and temporal composition of primitive basic behaviors. Also, the method for automatically constructing composite behavior based on reinforcement learning was proposed. She also introduced her Broadcast of Local Eligibility (BLE) approach to multi-robot coordination. The BLE mechanism, involving a comparison of locally determined eligibility with the best eligibility calculated by a peer behavior on another robot, allows heterogeneous robots to efficiently allocate themselves to an appropriate task without the need for any explicit communication or global knowledge of particular abilities.

Evolution of Communication

In the field of Evolution of Communication, MacLennan [8] studied the evolution of communication in synthetic agents and concluded that communication can evolve in a society of simple robotic agents. Yanco [9] investigated the evolution of simple communication protocol among nonverbal agents engaging in cooperative tasks. Her research robots, Ernie and Bert, have limited vocabulary that is self-organized over time to improve the performance of the tasks. Cangelosi [10] described different types of models for the evolution of communication and language and showed how evolutionary computation techniques such as artificial life can be used to study the

emergence of syntax and symbols from simple communication signals. Saunders and Pollack [11] implemented a multi-agent system that evolved a communication scheme to solve a given task without a priori native structure in place by simulating a tracker task (e.g., ants searching for food) and proved that communications between agents can influence the agents' behavior.

Cooperative Communication and Knowledge Sharing

For cooperative communication and knowledge sharing, Grabowski et al. [12] stated that communication is essential in a coordinated team. Without explicit communication, a robot can only interact with team members using sensors such as sonar, vision, or motion sensor, however more sophisticated collaborative tasks require exchanging detailed and abstract information that cannot be easily conveyed implicitly. Dudek et. al. [13] explored a follow-a-leader scenario in which one robot leads and another follows and compared the robustness and efficiency of the convoy with and without communication between each robot. Wang [14] compared a sign-board model with message passing, then introduced the condition of zero signal propagation delay under the sign-board and showed that sometime we should take this into account when transmitting signals through certain communication media. He also showed ways to implement a sign-board based on various techniques such as message passing, broadcasting, etc., while neglecting the effect of non-zero signal propagation delay. Morita et al. [15] proposed the expression format of necessary knowledge and knowledge transmission method for the robot to teach other robots through a form of questions and answers. The teacher robot teaches the worker robots by transmitting string data using the Conceptual Dependency theory expression[16], with the presumption that both teacher and student robots possess the same format of knowledge. When dealing with a large-scale robot group, some researchers stated that the idea of global knowledge or global communication within the group of robots could cause many problems and communication should only be needed between agents that are nearest to each other, therefore they prefer to use the local communication instead. Ohkawa et al. [17] proposed a way to solve the local communication problems by using an operator to conduct a group of robots in task-sharing and tested the effectiveness of the method. Duarte and Werger [18] introduced a Common Control Language (CCL) using the Port-Arbitrated Behavior-Based Control (PAB) technique to establish a standard interface from one agent to another for exchange of information and task delegation. By using the PAB technique, the user can set the robot's behavior and communication channel at run time with easy language syntax. Mataric [19] demonstrated a

cooperative transportation task in a group of simulated mobile robots that communicate by leaving landmarks in shared localization space. The method is shown to be robust with respect to significant localization error. Arai [20] analyzed the information diffusion by local communication among mobile robots by employing a probabilistic model to represent the robot motion. This logistic function model makes it possible to evaluate the time so that information is diffused to the required number of robots. Fujii et al. [21] proposed a new strategy for indirectly sharing knowledge between robots in local communication by using a device called the Intelligent Data Carrier (IDC) and placed it wherever a robot wants to share information with others. Other robots in the group can also rewrite new information into these IDC devices.

CHAPTER III

BACKGROUND

This chapter introduces concept of The Intelligent Machine Architecture, The Sensory EgoSphere, and The Self Agent etc. We will show in a later chapter how everything related to each other.

The Intelligent Machine Architecture

The Intelligent Machine Architecture (IMA) is an agent-based robot control architecture [22]. It was initially designed for a humanoid robot system called ISAC [23, 24, 25]. We are applying IMA to the mobile robot navigation problem in order to test the robustness of the software architecture across different types of robot applications. IMA has sufficient generality to permit the simultaneous deployment of several robot architectures. For example, we use variants of Arkin's motor schema [26]. Other IMA behaviors use variants of the subsumption architecture [27]. Our design process within the IMA is to decompose the system into a set of atomic agents. We use the term "atomic" to mean a fundamental building block and to distinguish IMA atomic agents from the more common use of "agent" or "cognitive agent" as an autonomous, intelligent entity be it machine, software, or biological. Within the context of IMA, an atomic agent is one element of a domain-level system description that tightly encapsulates all aspects of that element, much like the concept in object-oriented systems. The atomic agent serves as a superstructure for everything the software system knows or does relating to an element of the robot, a task, or the environment. IMA runs under Windows NT 4.0 or higher. Communication between atomic agents is handled transparently by the Distributed Component Object Model (DCOM). This is a service of Windows, which allows remote objects to be treated as if they were local. IMA agents communicate through message passing and have flat connectivity – any agent can, in principle, communicate with any other. Implicit hierarchies are formed, however, since all but the lowest-level IMA agents employ other agents to complete their tasks or to achieve or to maintain their goals. It can be used to implement almost any logical control architecture.

Taxonomy of IMA Agents

A brief description of two kinds of IMA agents are given below.

1. Atomic Agents: There are four basic types of atomic agents, plus an additional type of agent that exists as a concession to realistic implementation considerations.
    - *Hardware/Resource Agent:* Interface to sensor or actuator hardware. Those interfacing to sensors can provide other atomic agents with regular updates of sensor data. Those interfacing to actuators accept command and provide updates of current states to other atomic agents
    - *Behavior/Skill Agent:* Encapsulate basic behaviors or skills
    - *Environment Agent:* Provides an abstraction for dealing with objects in the robot's environment, including operations that the robot performs on an object, e.g., "look at"
    - *Sequencer Agent:* Performs a sequence of operations; often interacting with one or more environment atomic agents or activating and deactivating one or more atomic agents. Sequencer agents may call other sequencer agents, but there should be an identifiable highest-level sequencer agent
    - *Multi-Type Agen*t: Combines the functionality of at least two of the first four agent types. For example, in the interest of efficiency of the implementation it may be advantageous to combine the hardware and behavior types into a single multi-type agent
2. Compound Agents: An IMA Compound Agent is an interacting group of atomic agents that are coordinated or sequenced by one or more sequencer agents. The highest-level sequencer agent can be envisioned as the root node of a tree with connections and dependencies on other agents on branches.

The decomposition of an agent-based system into compound and atomic agents is not unique; it depends on the context of the problem and on the choice of robot architecture for the particular task or behavior.

Inter-Agent Communications

Usually, a single atomic agent is not able to perform useful activity by itself, thus collections of agents must communicate and interact to achieve any task. Although IMA does not place

restrictions on the ways in which atomic agents must communicate, most inter-agent communications are usually either: 1) one way data flow, 2) command-in/position-out, or 3) master/slave. Atomic agents that control sensors commonly use one-way data flow. A single sensory atomic agent acts as a data server by providing periodic data updates to one or more client agents. Actuator atomic agents most commonly use command-in/position-out communication. A single actuator atomic agent acts as a server by accepting commands from one or more clients. This actuation server performs a type of command arbitration, e.g., vector sum as in Arkin [26] or subsumption as in Brooks [27]. The server also provides clients with information about its current state, e.g., actuator position. Sequencer agents and environmental agents usually use master/slave communication to activate the slave agent. When the slave finishes, it informs the atomic agent that activated it.

Figure 1 shows the overall architecture of the Intelligent Machine Architecture (IMA). There is a Commander Interface Agent that works with the Sensory EgoSphere, Landmark EgoSphere, Self Agent, Peer Agent and other Atomic Agents. We are testing the robustness of this architecture in mobile robot applications.

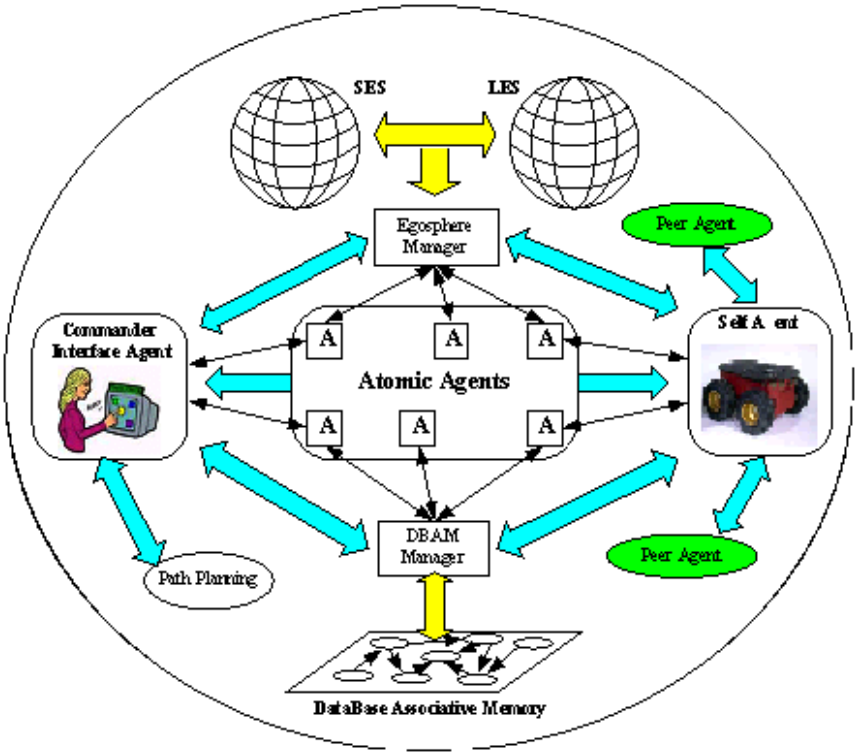

Figure 1 : IMA Agents Overall Architecture

Self Agent

The Self Agent is a compound cognitive agent that activates and maintains the activities of the robot itself. It receives high-level commands from the Commander Interface Agent and decomposes them into a set of executable commands. The Self Agent also monitors the system performance and reports significant errors back to the Commander Interface Agent. The Self Agent consists of:

1. *Command I/O and Status Agents*: Communicates directly with the Commander Interface Agent. It accepts commands from and reports the status back to the Commander Interface Agent
2. *Description Agent*: Provides information about other agents in the system, i.e., agent name & job description, active/inactive and success/error/ not executed
3. *Performance Agent*: Monitors other agents and reports significant errors to the Commander Interface Agent
4. *Activator Agent*: Activates the set of agents necessary for a specific goal or task
5. *Interpreter Agent*: Decomposes high-level commands into a set of executable commands.

Figure 2 illustrates the Self Agent interactions with the Peer Agent, the Commander Interface Agent and other agents.

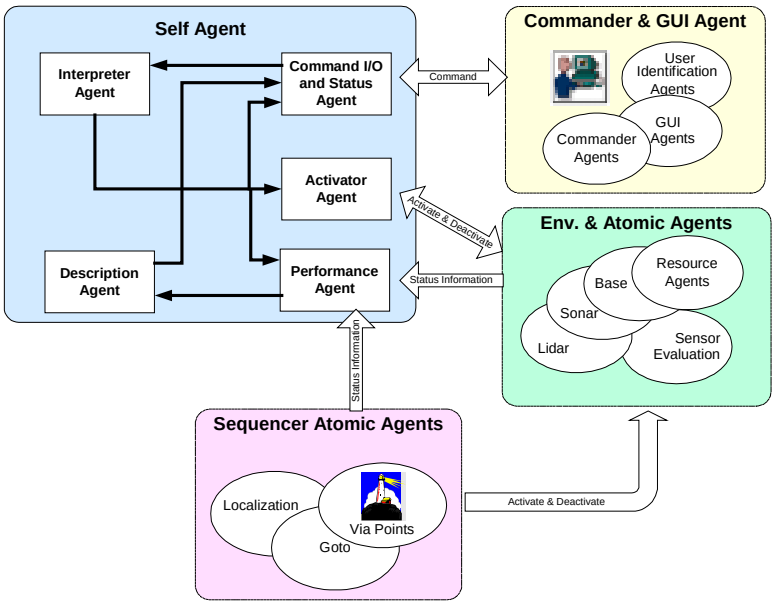

Figure 2 : Interaction between Self Agent and other agents

The Commander Interface Agent

Because Humans and robots communicate very differently from each other, the Commander Interface Agent concept has been developed to ease the communication difficulty and allow human operator to exploit the robot's abilities as much as possible.

The Commander Interface Agent will accept commands from the human operator and translate them into commands that robots can understand and implement. The Commander Interface Agent will also gather status and states from the robots and notify the human about current task progress.

Sensory EgoSphere

To facilitate remote control of a robot, a supervisory control system should enable a user to view the current sensory information. Efficient and accurate remote control of a robot would be facilitated by an intuitively understandable display of the robot's current multi-modal sensory information in the context of significant events in its recent past. Perhaps the most natural remote control environment is a virtual one that puts the user inside the robot as if she or he were operating it. Within such an environment, if sensory information is displayed in a temporal sequence in the direction from which it comes, the human operator can discern which sensory events belong together in space and time. A directional, egocentric display takes advantage of the person's natural patterns of recognition abilities to combine sensory modalities rather than the typical disconnected numerical or graphical displays of sensory data. To enable such display and memory, we are using a data structure, called the Sensory EgoSphere (SES) [28].

Spherical Map

We define SES as a databass, a 2-D spherical data structure, centered on the coordinate frame of the robot, spatially indexed by azimuth and elevation. Its implicit topological structure is that of a geodesic dome, each vertex of which is a pointer to a distinct data structure. The SES is a sparse map of the world that contains pointers to descriptors of objects or events that have been detected recently by the robot. As the robot operates within its environment, both external and internal events stimulate the robot's sensors. Upon receiving a stimulus, the associated sensory processing module writes its output data to the SES at the node that is closest to the direction from which the stimulus arrived. Since the robot's sensory processing modules are independent and

concurrent, multiple sensors stimulated by the same event will register the event to the SES at about the same time. If the event is directional, the different modules will write their data at the same location on the SES. Hence, sensory data of different modalities coming from similar directions at similar times will register close to each other on the SES.

Geodesic Dome Topology

Given that sensors on a robot are discrete, there is nothing to be gained by defining the SES to be a continuous structure. Moreover, the computational complexity of using the SES increases with its size, which is, in turn, dependent on its density (number of points on its surface). We use a (virtual) geodesic dome structure for the SES since it provides a uniform tessellation of vertices such that each vertex is equidistant (along geodesics) to six neighbors. The tessellation frequency is determined by the angular resolution of the sonar array. The SES is a multiple-linked list of pointers to data structures. There is one pointer for each vertex on the dome. Each pointer record has seven links, one to each of its six nearest neighbors and one to a tagged-format data structure. The latter comprises a terminated list of alphanumeric tags each followed by a time stamp and another pointer. A tag indicates that a specific type of sensory data is stored at the vertex. The corresponding time stamp indicates when the data was stored. The pointer associated with the tag points to the location of a data object that contains the sensory data and any function specifications (such as links to other agents) associated with it. The type and number of tags on any vertex of the dome is completely variable. The SES is not a complete geodesic dome, instead, it is restricted to only those vertices that fall within the directional sensory field of the robot. Since the camera is mounted on a pan-tilt head, imagery or image features can be stored at the vertex closest to the direction of the camera. Sonar and laser work only in the equatorial plane of our robot and so their data is restricted to the vertices near the dome's equator. Figure 3 shows an example of the SES.

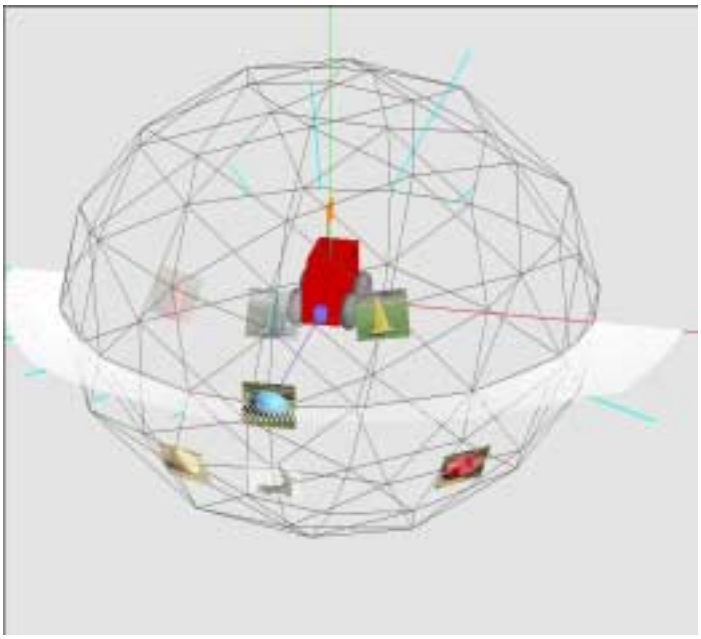

Figure 3 : Sensory EgoSphere of the robot

Mobile Robots

Skeeter

Skeeter is a Pioneer 2-AT robot [29] developed by Activmedia Robotics, Inc. Pioneer robots are, “plug and play” mobile robots, containing basic components for sensing and navigation in a real-world environment, including battery power, drive motors and wheels, position/speed encoders, and integrated sensors and accessories. They are managed via onboard microcontrollers and mobile-robot server software. Figure 4 shows a picture of Pioneer 2-AT robot with the factory default setup.

Figure 4 : Pioneer 2-AT robot from ActivMedia Robot, Inc.

The Pioneer 2-AT also has a variety of expansion power and I/O ports for attachment and close integration of additional sensors and other accessories. These include an addressable I/O bus for up to 16 devices, two RS-232 serial ports, eight digital I/O ports, five A/D ports, PSU controllers (an Integrate circuit chip that regulates output voltage) and more, all accessible through a common application interface to the robot server software, P2OS.

Hardware Specification of Pioneer 2-AT

This specification is the factory default from ActivMedia Robot, Inc.

Body

Pioneer 2-AT has a sturdy, but lightweight aluminum body which houses the robot's batteries, drive motors, electronics, and other standard components, including the forward and rear sonar arrays. The body also has sufficient room, with power and signal connectors, to support a variety of robotics accessories inside, including an A/V Wireless surveillance system, radio modems and Ethernet, onboard computer. Figure 5 shows the basic body structure of the Pioneer 2-AT.

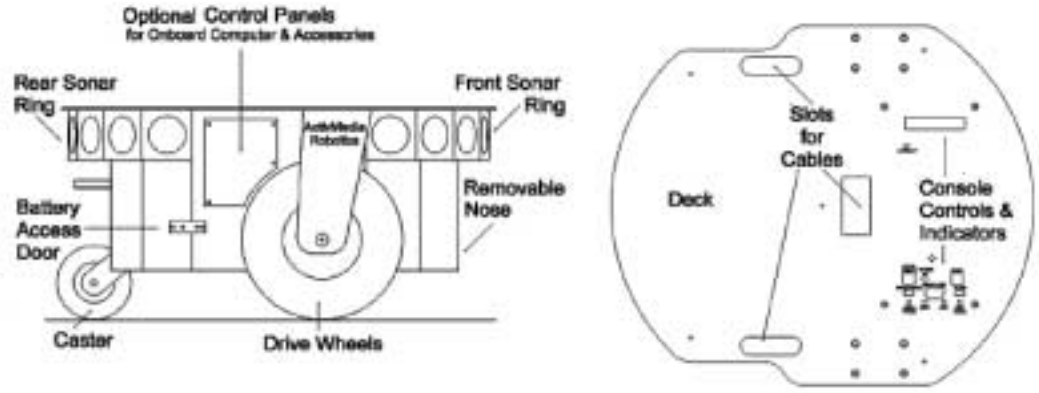

Figure 5 : Basic Body Structure of the robot

Motors and Position Encoders

The Pioneer 2's drive system uses high-speed, high-torque, reversible-DC motors. Each front-drive motor includes a high-resolution optical quadrature shaft encoder that provides 9,850 ticks per wheel revolution (19 ticks per millimeter) for precise position and speed sensing and advanced dead-reckoning.

Sonar Arrays

The Pioneer 2 supports both front and rear sonar arrays, each with eight transducers that provide object detection and range information for features recognition, as well as navigation around obstacles. The positions of the sensors are fixed: one on each side, and six facing outward at 20-degree intervals, together providing 360 degrees of sensing. The sonar-firing rate for each array is 25 Hz (40 milliseconds per sonar per array) and sensitivity ranges from ten cm (six inches) to more than five meters (16 feet). The front Sonar Array is given in Figure 6. The back Sonar Array is identical to the front.

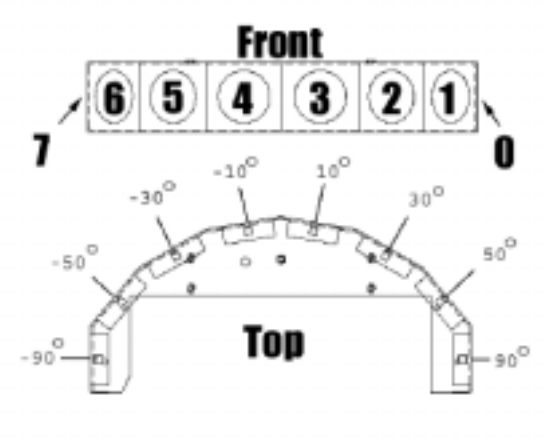

Figure 6 : Front Sonar array of robot

Further modification

Since the factory default configuration was not sufficient for our research task, we modified the robot to meet our specific needs.

Tower and Omni-view Camera Head

Since our work is based heavily on vision, we built a front tower (20" tall) on top of the robot's panel, and placed an omni-directional camera head, as shown in Figure 7. The camera head consists of 7 NTSC CCD cameras and a video multiplexer, on top of the tower. This gave the robot the ability to scan the surrounding environment in less than 4 seconds.

(a) (b)

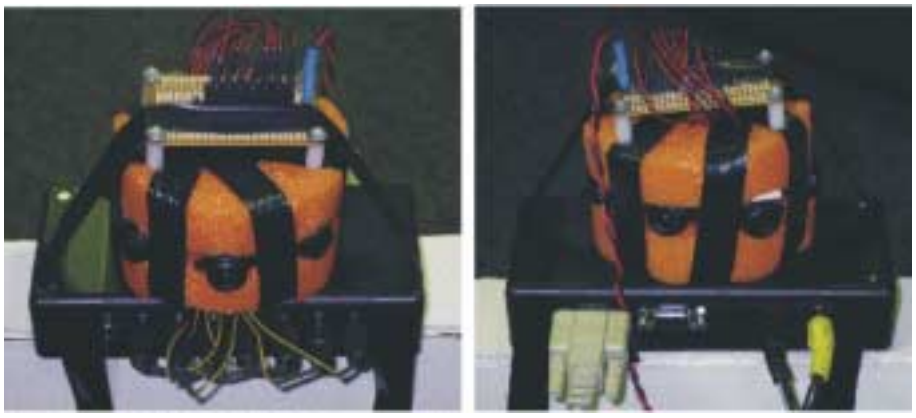

Figure 7 : (a) Front view of the head, (b) Back view of the head

Processing Unit

The process of images from the omni-directional camera head involves intensive calculations. Therefore, we installed a laptop computer on the panel beside the tower for additional computing power.

Software

We wrote software that can communicate directly with the robot without using the factory default software. This gave us total control of the robot.

Wireless Network

We installed a wireless local area network connection on the robot to enable connection to connect to an outside computer network.

The modified version of the robot is shown in Figure 8. We covered the tower that holds the camera head with a T-shirt for purely cosmetic reasons. The laptop is located behind the tower.

Figure 8 : Modified version of the robot

Scooter

Scooter, an All-Terrain Mobile Robot, model ATRV-Jr, was developed by IS Robotics, Inc. and includes four-wheel drive, differential steering, pneumatic knobby tires, a weather-resistant enclosure and many sensors.

Hardware specification of the robot

Below is the default setup from the Factory. [30]

Onboard CPU

Intel® Pentium III™ computer systems in a custom enclosure with specially shock-mounted hard-drives, integrated inside the robot.

- Single Pentium III™ CPU with 128MB SDRAM
- ATX motherboard mounted in a special aluminum enclosure (motherboard dual-processor ready)
- 18 Gigabytes/IDE hard drive, specially shock-mounted
- 1.44 MB, 3.5" double-density floppy-disk drive
- Video accelerator with 1 MB RAM
- Intelligent high speed multi-port serial card (8 ports)
- PCI Ethernet
- Linux operating system
- Mobility Robot Integration Software

Laser and Sonar ranging sensors

Scooter has a SICK Laser Measurement System, which comes with the following description,

- Pulsed IR
- 492'(150m) range

- 180-degrees coverage
- 0.5% angle resolution
- +/- 50 mm dist. measurement resolution

For sonar sensors, Scooter has a front sonar ring unit, two units on each side and two units on the back of the robot for obstacle avoidance, as shown in Figure 9.

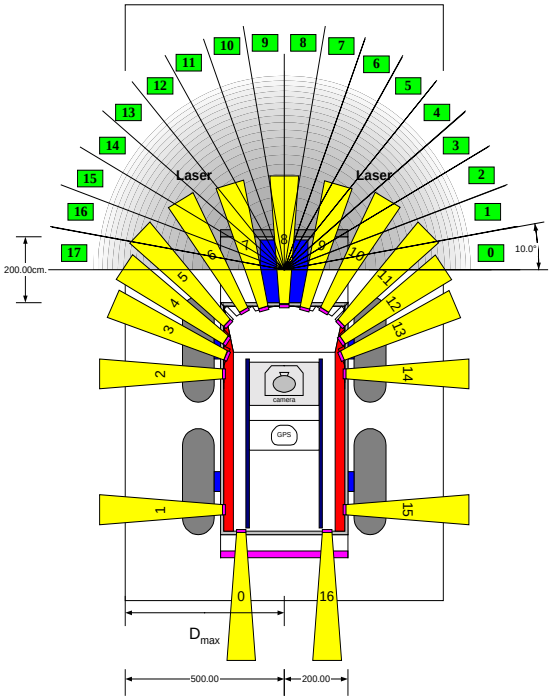

Figure 9: Sonar and LASER ranging sensors on Scooter

Compass

- Single-axis heading sensor with 0.5 degree accuracy
- RS-232 communication port
- Auto-calibration compensates for field effects of other on-board equipment

Drive Motors Unit (DMU)

In the Scooter Drive Unit, gyros are used for increased accuracy

- 3-axis pitch/roll/yaw gyros
- 3-axis pitch/roll/yaw accelerometers
- 150 degrees per second gyro rate
- 10Hz gyro bandwidth
- 100Hz accelerometer bandwidth
- RS-232 communications interface
- Low power consumption

Camera and Accessories

There is a camera mounted on top of Scooter that consists of,

- 2 PCI Frame Grabbers
- Pan-Tilt Head
- 2 XC999 Color CCD cameras with 6mm lenses (NTSC or PAL)
- Custom adjustable stereo camera mounting bar
- Synchronization electronics
- Power, signal and control cabling

Communications

Scooter is equipped with BreezeNet Wireless Ethernet having a maximum bandwidth of 10 Mbps.

Figure 10 (a) shows the original ATRV-Jr robot produced by IS Robotics. The modified one is shown in Figure 10 (b).

(a) (b)

Figure 10 : (a) Scooter with factory setup, (b) With our modifications

Communication within mobile robots group

The Commander Interface Agent [31] is responsible for presenting the system status and sensory data to the human commander (via the GUI). Each robot will have its own Self Agent representing the robot's individual information, and will have Peer Agents representing the other robots' information. For example, if the team consists of Scooter (ATRV-Jr) and two Pioneer 2-AT robots (Skeeter1 and Skeeter2), then Scooter will have its own Self Agent and two Peer Agents representing each Pioneer Self Agent. Since the robots in the squad are connected to each other by an Ethernet network, the commander will be able to easily observe the information the robots are sharing. Additionally, by using the IMA Agent-based Human-Robot Interaction (HRI) approach, we should be able to integrate communications and operations of tasks for the desired goal in an efficient and productive manner, as shown in Figure 11.

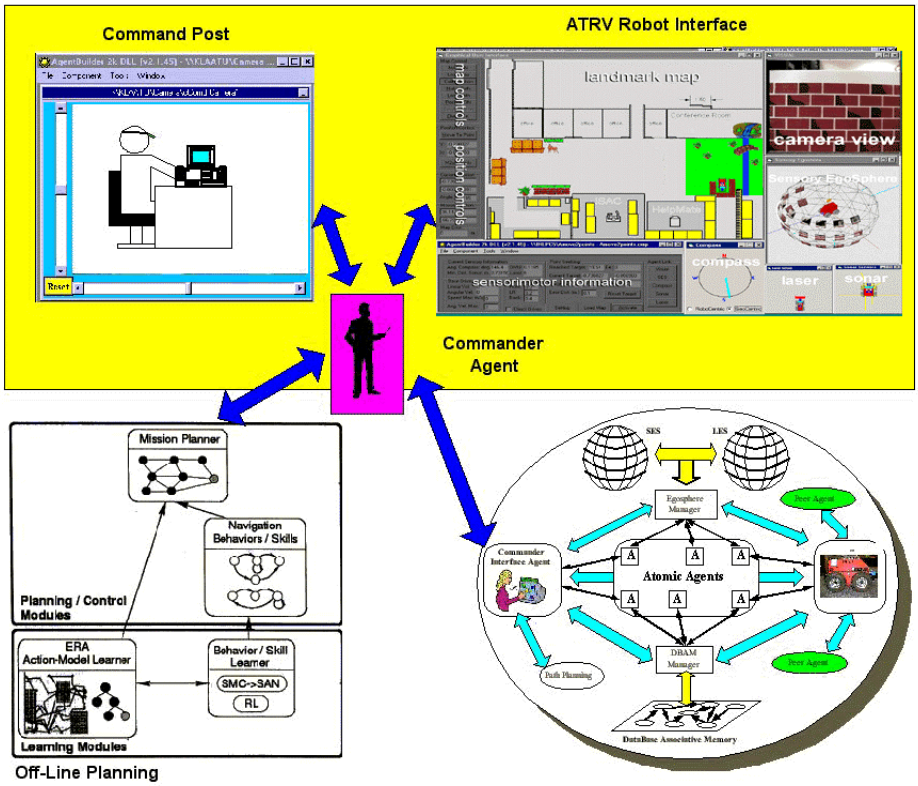

Figure 11 : IMA Agent-based HRI Approach

Jung and Zelinsky [32] offered an interesting approach to communication between robots using a layering solution. Their protocol is robust when dealing with an uncertain environment, but it becomes more complex with increasing layers. Our approach is based on a flat, one-layered communication protocol to avoid some of the complexity in a layered protocol. When a robot senses an object and needs to share the information, it will generate its own SES and send back the symbolic description of the object (i.e., a red ball, a green cone, etc.) and the direction with respect to itself to the other robots through its Peer Agent. The other robots will then be able to process the new SES data and update their map, as illustrated in Figure 12.

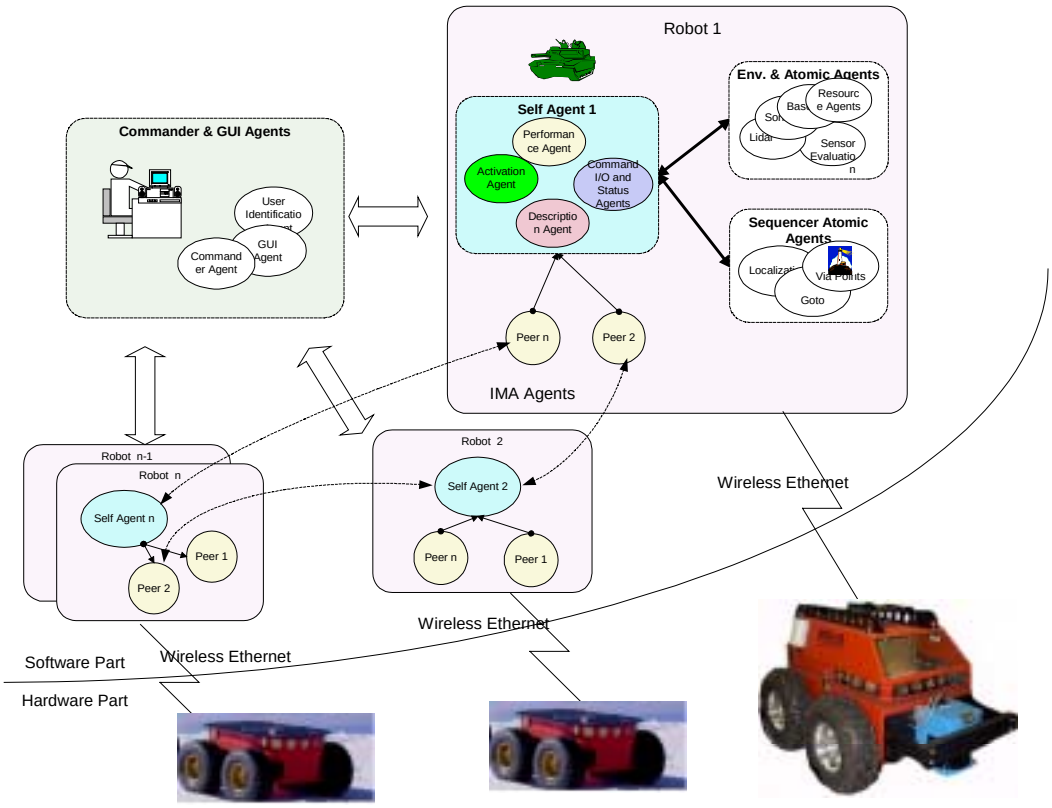

Figure 12 : Communication within a mobile robot group

CHAPTER IV

CONCEPTS OF PEER AGENT

In this chapter, we introduce the concepts of Peer Agent and how Peer Agents can be used for multi-robot communication.

Overview

Within the robot group, each robot has its own group of agents that represent the individual robot such as the Self Agent which represents the robot's thinking/planning mechanism, the Peer Agent which represents the robot's means of communication, and many other agents, which have their specific applications, (e.g., a SES agent that represents the robot's perceived environment).

In order for each robot in the multi-robot group to communicate with each other, the concept of the Peer Agent has been developed. The Peer Agent is an agent that handles all the communication coming in and going out of a robot and also represents one robot's Self Agent to the other robots in the group. The Peer Agent dependently couples with the robot's Self Agent. Since Self Agents cannot communicate with one another directly, all the communications (information and commands) between the robots will be sent through each robot's Peer Agent.

A Peer Agents consists of two parts: 1) *The Peer Client*, which will be attached to an agent within a robot that requires data from another robot, and 2) *The Peer Manager*, which is attached to the Self Agent of the robot that acts as an information server for the other robots in the group. There may be as many Peer Clients as possible within one robot, but there may be only one Peer Manager per robot.

The Peer Client specifically is used to obtain data from one agent and then send that data to the agent that the client is attached to. We can state that the Peer Client encapsulates the communication channel between two remotely separated agents within a robot group. Thus, a Peer Client that is connected to an agent will monitor the availability of the exiting communication channel and will close the communication channel immediately if the channel is lost, damaged or closed by the information source. Because of this, the components/agents that are connected to each other will not be damaged or malfunction when something happens to the communication channel.

Imagine what would happen if you have many Peer Clients connecting all kinds of components/agents between a group of robots; you would not be able to manage all the communications that are going on within the robot group. That is why the Peer Manager was developed. The Peer Manager, which is attached to the Self Agent, will establish a communication channel for any Peer Client that needs some information from a particular robot. Each Peer Client will have to register itself and request the data (or the address of a source data container) through the Peer Manager of the robot. Therefore, the Peer Manager, which has the knowledge of the whereabouts of all the data/information within the robot, will send back the address of the source agent to the requested Peer Client.

Figure 13 illustrates how components, the Peer Agent, users, and the Commander Interface Agent interact with each other. The Commander Interface Agent will also have an ability to retrieve information from robot by creating the Peer Agent Client and register it to the Peer Agent Manager on the robot.

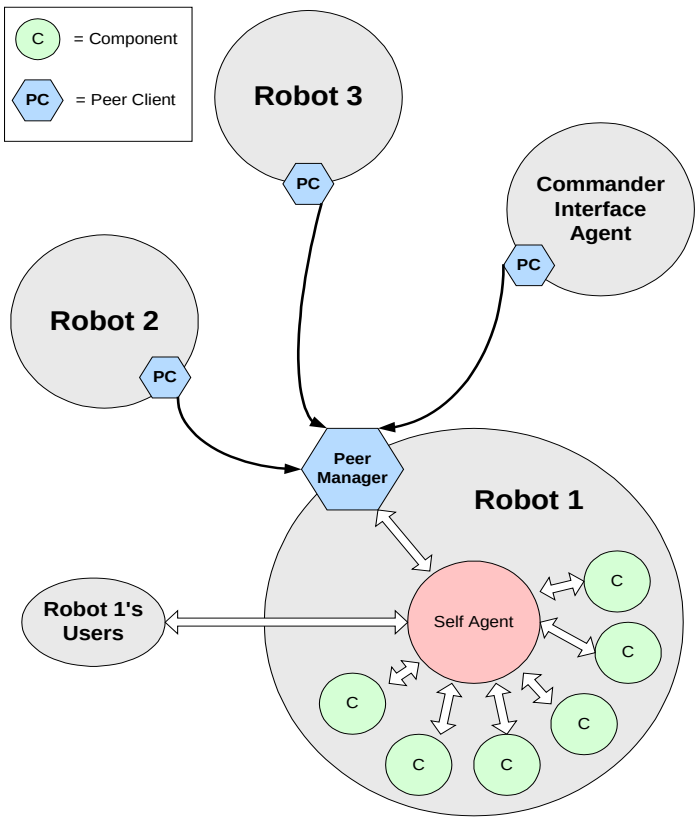

Figure 13 : Robot group consisting of robots, Commander Interface Agent and Users

Furthermore, the Peer Manager will be able to determine the priority of each request, thus it will be able to sort all requests and respond to the more important task-related requests in priority order. There are three kinds of priority settings in the Peer Manager as follows:

1. Priority of the agent/robot making a request: Some robots, e.g., the leader robot, will have a higher priority than other robots
2. Priority of the request itself: Each request may have a different priority depending on the task specifications
3. Priority of the Source robot's task: If the task on the source robot has a higher priority than the request's priority, the Peer Manager will not grant the request until the request priority is higher than the source robot's task priority

Each Peer Manager will have a special broadcast channel to communicate with other Peer Managers. This channel will be used to broadcast emergency information and to provide the status of the robot's Self Agent. The Commander Interface Agent will be able to monitor this channel and use it to determine suitable task-related decisions/commands.

Peer Agent Architecture

Inside the Peer Agent as shown in Figure 14, there will be two communication modules; one will be used to communicate with the attached Self Agent and another will communicate between a Peer Manager/Peer Client Agent. The Interface module will be used for displaying useful information about its own status or states to the user. Finally, Index Module will be used as a database that contains the known information sources within the robot.

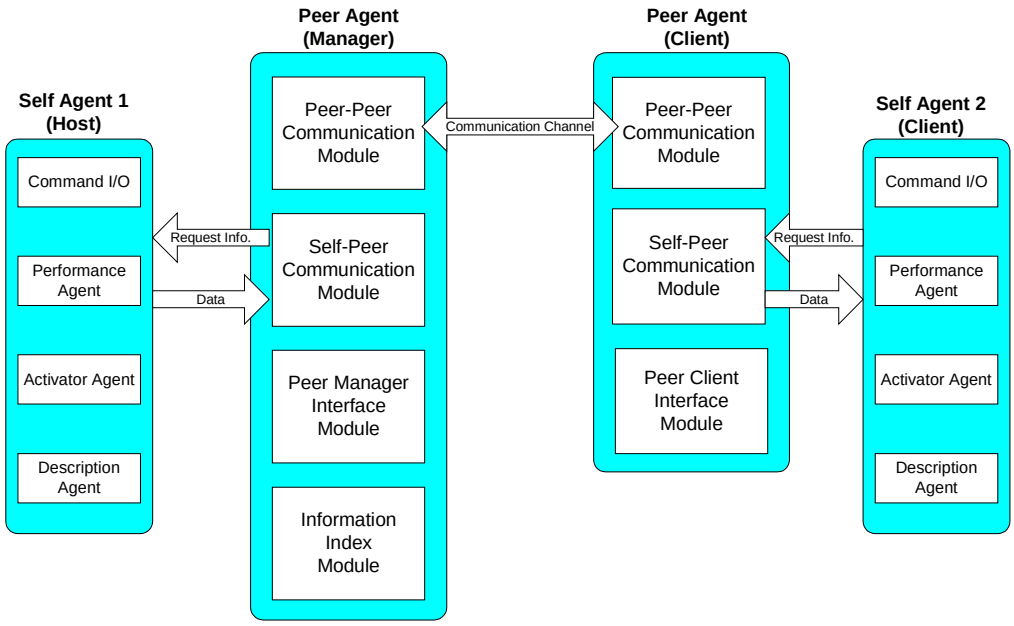

Figure 14 : Peer Agent Architecture

Figure 15 shows a network of robots communicating with each other using Peer Agents. In this situation, Robot 1 is the source of information to the other robots in the group. Each robot, which needs to retrieve information from Robot 1, will create its own Peer Agent Client and connect it to Robot 1's Peer Agent Manager. The information flow will begin with Robot 1's Self Agent into the Peer Agent Manager and is sent across the robot network into the Peer Agent Client where it will finally flow to another robot's Self Agent.

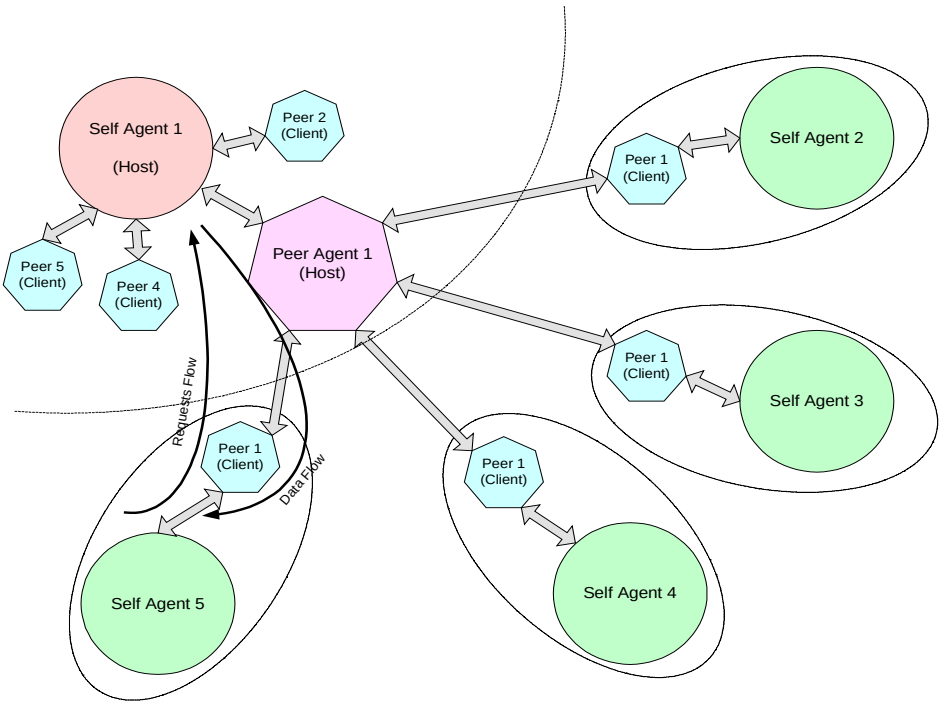

Figure 15 : Multi-Agents Peer connection

Between each Peer Agent Manager, there is a special communication channel created for broadcasting some very important status information or the task-related emergency information of each robot. This way, every Peer Agent Manager and The Commander Interface Agent will be able to notify or check any robot's status in the robot group, as depicted in Figure 16.

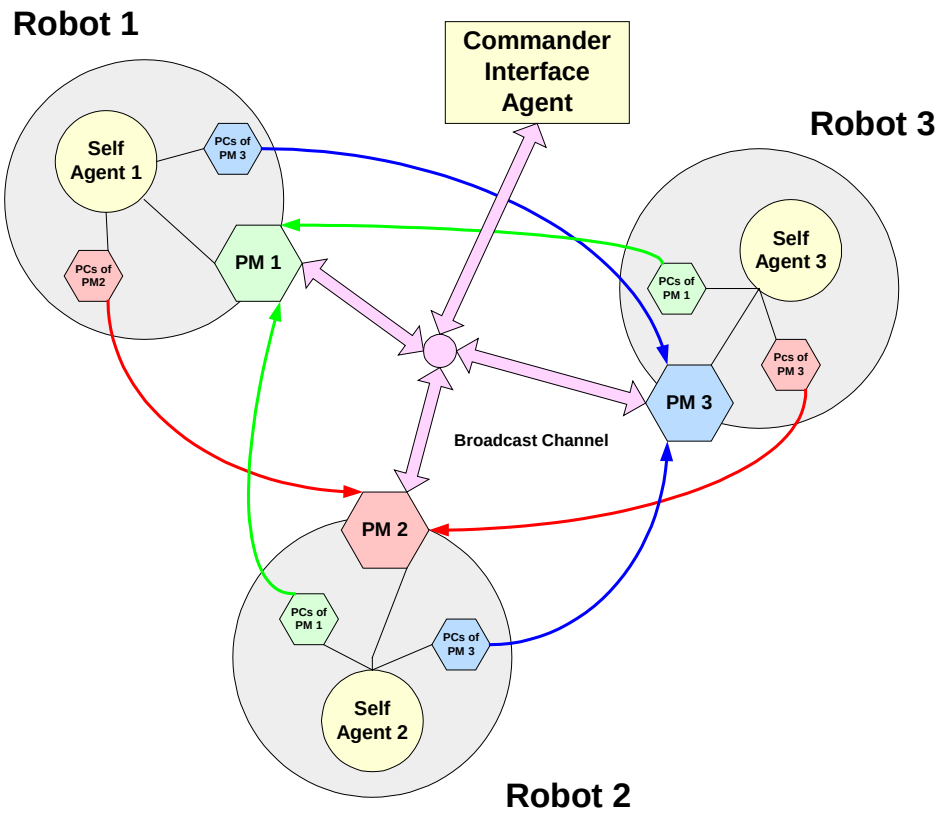

Figure 16 : Special communication channel between Peer Managers and Commander Interface Agent

Peer Agent Manager

The Peer Agent Manager will act as a communication and data server for the robot. There are four communication channels within the Peer Agent Manager as follows:

1. *Normal Input Channel* (Request Channel): Accepts a request from Peer Agent Clients
2. *Normal Output Channel* (Reply Channel): Answers (grants or denies request) back to Peer Agent Clients
3. *Broadcast Channel*: Communicates with other Peer Agent Managers and the Commander Interface Agent
4. *Self Agent Interaction Channel*: Interfaces with the Self Agent.

The operations of the Peer Agent Manager consist of,

1. Scan the robot for all available data sources
2. Update the data source during specified time intervals and inform the Peer Agent Clients of the update sources

3. Monitor all requests from Peer Clients
4. Reply to requests made by Peer Clients
5. Set priorities for all requests made by Peer Agent Clients
6. Inform the Self Agent of incoming requests and priority levels
7. Broadcast emergency signals (current behavior and status) through a special channel, which is monitored by other Peer Managers and the Commander Interface Agent

Figure 17 shows the communication channels between the Peer Agent Client and Peer Agent Manager including the Special Broadcast Channel, which the Peer Agent Managers use to communicate with each other.

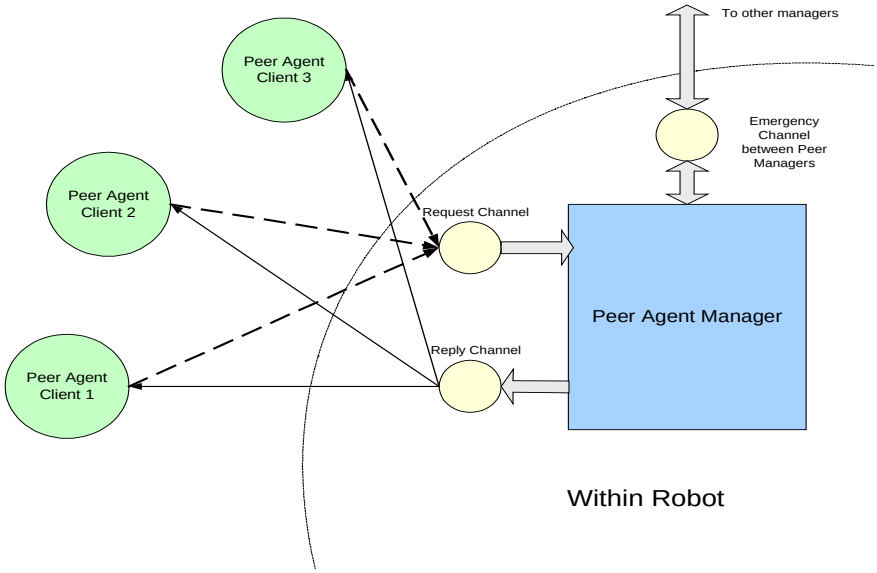

Figure 17 : Communication channels within Peer Agents

Parts of the Peer Agent Manager are shown in Figure 18. Figure 18(a) shows a list of data source's addresses known to the Peer Agent Manager that can be sent to a Peer Agent Client if requested. Figure 18(b) shows a list of Peer Agent Clients that are registered with the Peer Manager.

(a) (b)

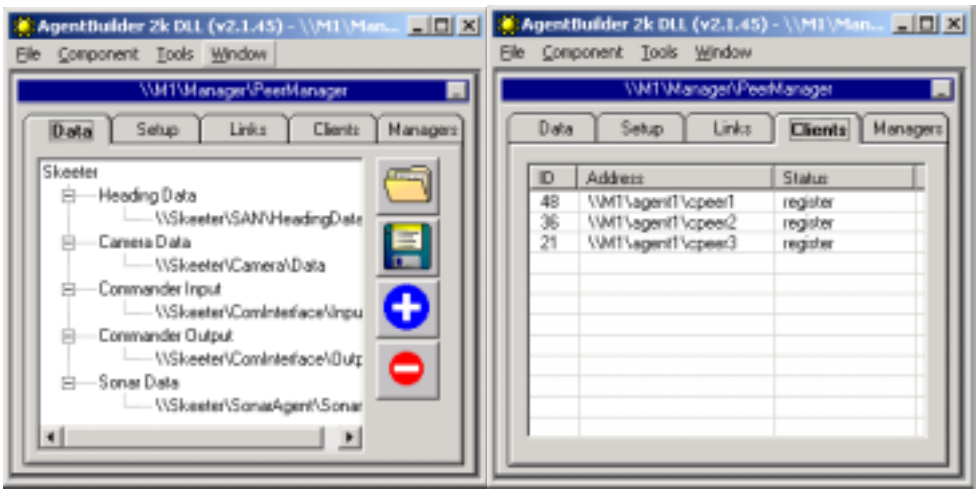

Figure 18 : Peer Manager User Interface

Peer Agent Client

The Peer Agent Clients encapsulates communication channels. Each agent requiring information from another robot must have a Peer Agent Client registered with the Peer Agent Manager. When the Peer Agent Client has been initialized and registered to a specific Manager, that agent will be able to request data from the source robot via the Peer Agent Client.

Peer Agent Client operations will consist of the following:

1. Initialize and register itself to Peer Agent Manager
2. Connect to any source components without knowing which type of components they are and across any machine in the network
3. Monitor the status of the channel; if the connection is lost, the Peer Client will disconnect the channel without damaging other agents
4. Share and keep track of data between the two components/agents.

In a Peer Agent Client, there are two kinds of communication channels:

1. Interaction channels with the Peer Agent Manager: Consists of an input channel and an output channel. The Peer Agent Client will send requests via the input channel and receive the Peer Agent Manager response via output channel

2. Data Transfer channels: Consists of two channels; one will connect to the data source agent and the other will connect to the target agent. The data flow will begin from the source agent and continue into the target agent.

When the Peer Agent Client successfully establishes a link between itself and the Peer Agent Manager, it will generate an ID number and initialize itself with the Peer Agent Manager. If the ID number is not unique, or has been used by another Peer Agent Client, the Peer Agent Manager will decline the initialization and require the particular Peer Agent Client to generate a new ID number.

After the initialization has completed, the Peer Agent Manager will send out a package containing all the known data sources within the robot to the Peer Agent Client and the Peer Agent Client can use this information to request specific information from the Peer Agent Manager.

Figure 19 and Figure 20 show a user interface of a Peer Agent Client in an unconnected state and connected state, respectively.

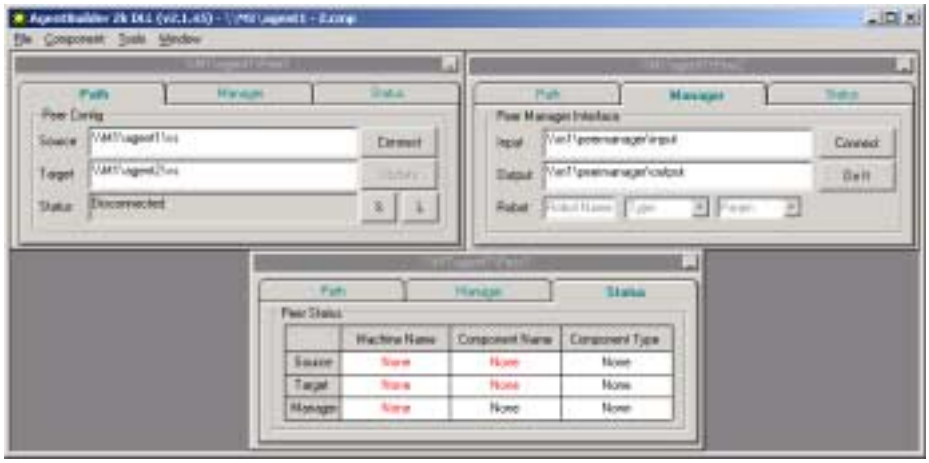

Figure 19 : Peer Agent Client's User Interface (Unconnected Version)

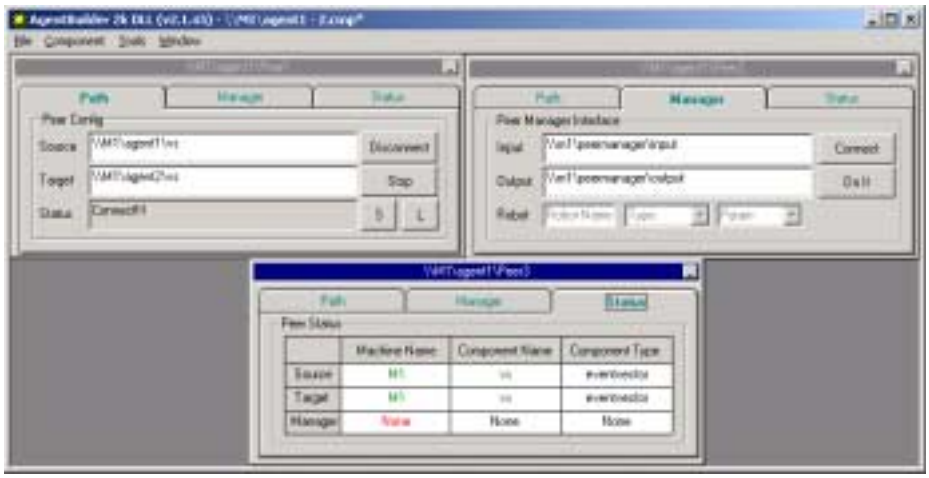

Figure 20 : Peer Agent Client's User Interface (Connected Version)

Peer Agent Manager and Peer Agent Client Interactions

Between the Peer Agent Client and the Peer Agent Manager, there is an interaction channel that used by the Peer Agent Client to register itself or request information to the Peer Agent Manager. The Peer Agent Manager also uses this channel to reply the request from the Peer Agent Client or send some information to the registered Peer Agent Clients.

There are four interaction procedures used by the Peer Agent Client as follows,

1. Register: The Peer Agent Client uses this procedure to register itself to the Peer Agent Manager for information retrieving. The Peer Agent Manager, if accepts the register by the Peer Agent Client, will grant the register back to the Peer Agent Client and send a list of known data source within the robot to the Peer Agent Client.
2. Unregister: The Peer Agent Client uses this procedure to unregister itself from the particular Peer Agent Manager. The Peer Agent Manager then will stop monitoring the activity of this Peer Agent Client.
3. Request: The Peer Agent Client uses this procedure to request data retrieval from the Peer Agent Manager. If the Peer Agent Manager grants this request, it will send an address of the data source requested to the Peer Agent Client.
4. Disconnect: The Peer Agent Client uses this procedure to inform the Peer Agent Manager that it will stop retrieving data from the information source.

Applications of the Peer Agent

The main application of the Peer Agent is to share knowledge between robots within the group. Additionally, the Peer Agent can also broadcast the status of a robot or it's behavior to other robots' Self Agent and uses this method to trigger a collective behavioral change within the group to improve performance in order to achieve the final task/goal.

Figure 21 illustrates various applications of the Peer Agent. The word "raw data" means all the sensory data that the robot can perceive, e.g., Sonar data, Laser ranging data, and visual data from the camera.

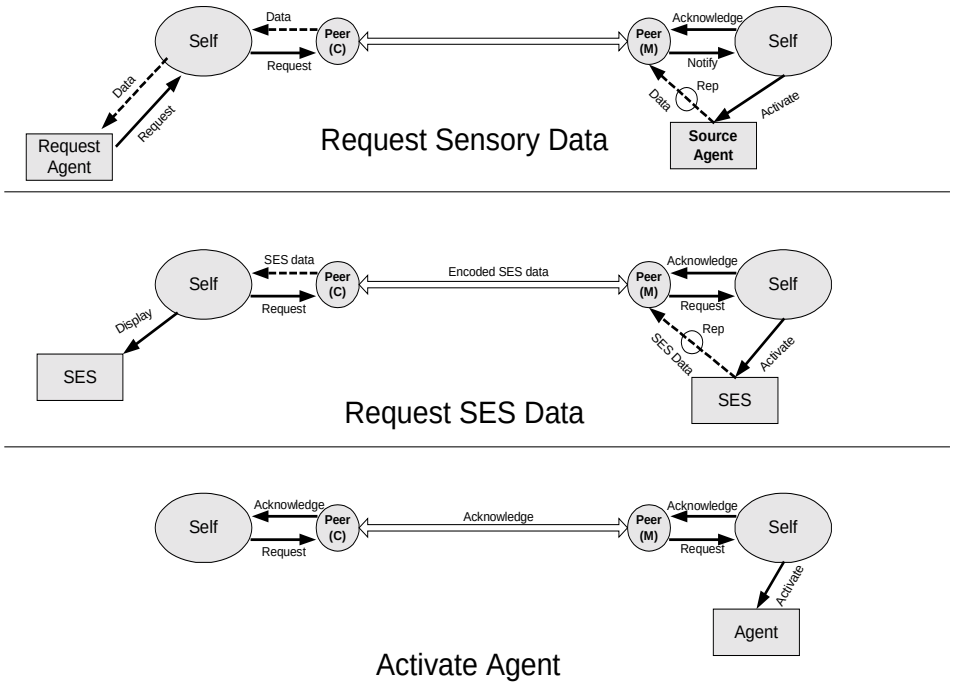

Figure 21 : Applications of the Peer Agent

Sharing information between robots involves two primary methods or means, as described in the following:

1. *Sharing raw sensory data between mobile robots:*

   Sharing sensory data between robots is nothing neither new nor very interesting in mobile robot research, but it is still very important for a robot within the same group to be

able to sense what other robots sense and/or see what other robots see, in order to be able to complete tasks.

2. *Sharing Sensory EgoSphere (SES) between mobile robots:*

Usually raw sensory data does not make a lot of sense to other robots with different configurations. It is easier and more practical to combine the sensory data into an abstract data structure that every robot will understand and interpret correctly.

The SES is one such practical and easy method to combine data into abstract data. SES is the combination of ranging data and visual data. When a robot makes a SES, the final abstract data will consist of a series of the types of objects and angles of that object relative to the center of the robot. This data is a string of text and can be shared easily with other robots.

3. *Broadcast behaviors within the robot collective:*

The Peer Agents allow one robot to share its' states or current behavior within the group through a broadcasting channel. Each Self Agent will monitor this channel and change the robot behavior to adapt to the rest of the group to improve the group's performance and robustness of the collective. The Commander Interface Agent can also monitor this channel and notify the Human Commander of the current situation within the robot group.

CHAPTER V

KNOWLEDGE SHARING USING PEER AGENTS

In this chapter, we showed how SES data are created and how robot shared them with others. We also propose an algorithm that used the shared SES data to calculate new heading for robot. The agents that used in this experiment are also described later in the chapter. At the end of the chapter, we propose a method to test the performance of the algorithm and we analyze the algorithm's performance at different scenarios.

Objective of Experiment

The objective of this experiment is to show how to share knowledge among robots using the Peer Agent. Two mobile robots, Skeeter (Pioneer-2 AT robot) and Scooter (ATRV-Jr robot) were used. They will have their own Self Agent, Peer Agent Manager, and Peer Agent Clients. The Self Agent will act as the decision-making part of the robot. The Peer Agent Manager will manage all communication channels between robots and also, when the robot wants to retrieve data from other robots, it will create Peer Agent Clients to connect to other robot's Peer Agent Manager to request the data.

We will show that knowledge sharing between robots can lead to a more stable, robust and efficient system within a group of robots. Additionally, when a robot broadcasts its' current behavior to the group, other robots in that group will adapt their own behaviors (when necessary) to a more suitable one, and allow the group to achieve its given task more efficiently.

Aside from sharing knowledge between robots within the group, the human commander can also tap into the robots' communication channels and receive important task-related information for better understanding of the current situation.

Experiment Setup

There are two mobile robots involved in this experiment. Skeeter, which has an omni-directional camera head, stands in the middle of the scene and is surrounded by six to eight objects with different colors (e.g., red, green, blue, pink). Scooter waits around the perimeter for further instructions from the human commander.

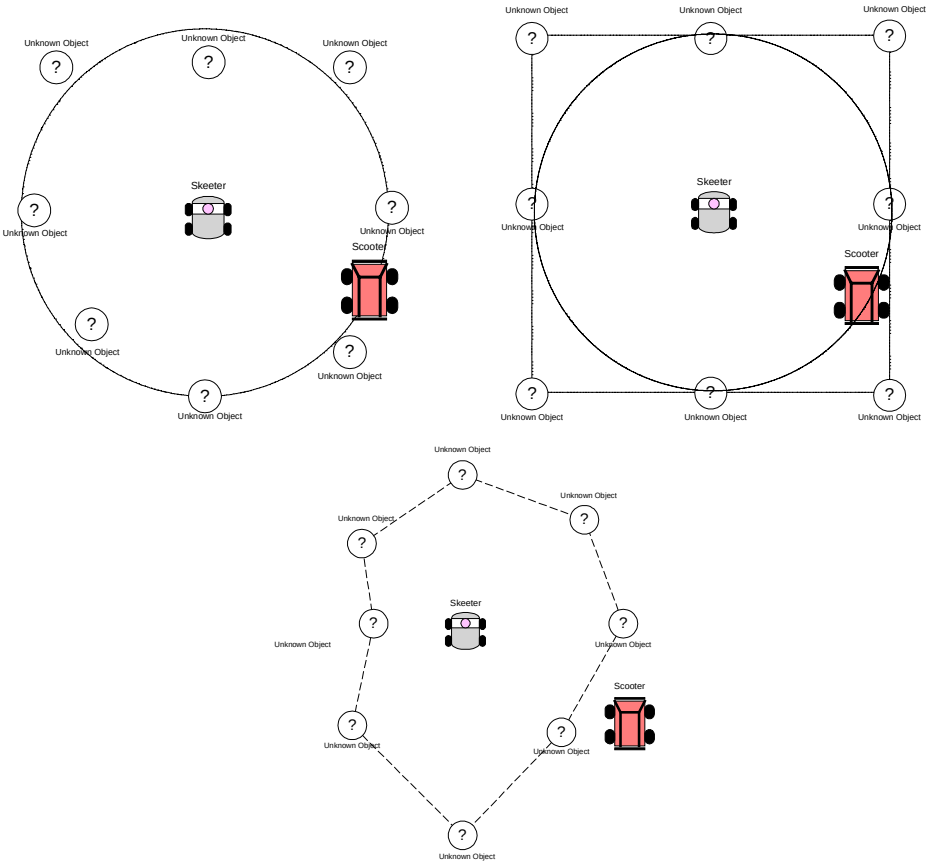

Figure 22 : Many ways to setup the environment for knowledge sharing demonstration, which results in the same SES data

As shown in Figure 22, since the Sensory EgoSphere data may or may not contain distance information, and is independent from the shape of the environment, we can setup the environment into any configuration we desired, such as contour or rectangular or any irregular shape, and still get the same SES data from each configuration (as long as they contain the same objects at the same angle). So, for the simplicity of this experiment, we projected each object onto the circular shape (called SES Circle) and let the algorithm treats each object as having equal distance.

The human commander, located outside of the perimeter, will be connected to the robots via the Commander Interface Agent. The Commander Interface Agent will be responsible for displaying the current status of the task to the human commander and receiving instructions from the commander to send to the robots.

In this experiment, the Self Agent is a compound agent that receives from the human commander instructions which it interprets then sends to the Spreading Activation Network (SAN) [33, 34], which provides the Action-Selection scheme to select an appropriate action for the robots

to achieve the goal implied by the commander. This SAN is a set of connected nodes and consists of Condition Nodes and Competency Module Nodes, which act as a decision-making mechanism that will select the action that is most suitable for the robots current situation.

In this setup, Skeeter acts as the observing post and uses its camera head to scan the surrounding environment for interested objects. In the meantime, Scooter acts as a scout robot that moves towards the interested objects provided by the human commander and observes them. The specification of the interested object will be given by the human commander via the Commander Interface Agent. Skeeter then scans the surrounding area to create a Sensory EgoSphere (SES). If it finds the object that matches the commander's specification, it sends this knowledge (i.e. the SES data that contains the object's angle and Scooter's angle) to Scooter. At that point, Scooter calculates its new heading from the SES data Skeeter provided and heads in the direction of the target object.

Test Bed Demonstration

First, the human commander sends instructions to the robots through the Commander Interface Agent, such as "Find red object". When the instruction arrives at Skeeter, the Spreading Activation Network (SAN) initiates the creation of the Sensory EgoSphere (SES) of the surrounding area and search for the particular object. If the object is present within the vision range, then Skeeter will send this SES data to Scooter via the Peer Agent. When Scooter receives the SES data, it calculates its new heading and navigates towards target direction.

During Scooter navigation towards the target, some conditions may occur and alter the state of the robot, such as Scooter encounters obstacles along the way, Scooter's SAN network will switch from Move-to-target behavior to Avoid-obstacle behavior until the robot has a clear path then it will switch back to Move-to-target.

This process will go on until Scooter detects the desired target and moves to a close range of the target. Scooter's SAN will then send a signal to Skeeter that the target has been reached and request the next target.

Figure 23(a) shows a scenario for searching and knowledge sharing for a target. When the human commander sends the instruction to the robots, e.g., "Find red object", Skeeter scans the environment for that target. Figure 23(b) shows where Skeeter has found the target and sends SES data to Scooter, thus Scooter calculates a new heading and moves toward the target.

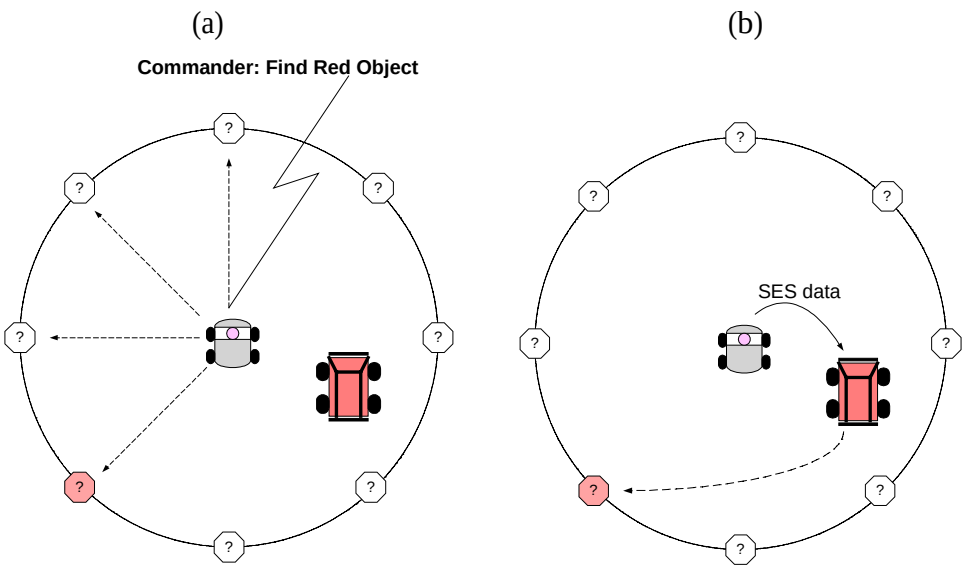

Figure 23 : Scenario for Knowledge Sharing Demonstration

Creation of the Sensory EgoSphere

Creating SES data is very important for this demonstration to be able to succeed efficiently and effectively. When Skeeter receives commands from the Commander Interface Agent, it will start scanning the surrounding environment using its omni-directional camera head. In each camera, Skeeter will capture the image and analyze it for the particular colors of interest, which stored in Skeeter's database, then store the data in a syntax that can be understood by other robots. After each scan session, Skeeter will send this data out for other robots and the Commander Interface Agent to retrieve.

A SES data is a short-term database structure, which is indexed by an azimuth, elevation and time. Directional sensory processing modules write information on the SES at the location corresponding to the source object's direction [35]. Each module calls SES agent with location, a tag, a time, and a pointer to its data. Other agent that used to display data or to perform data analysis can read from or write to any section on the SES.

Figure 24 shows an SES agent, which is used to create SES data for Skeeter in this experiment. A more detail descriptions of this agent are provided in Appendix B section.

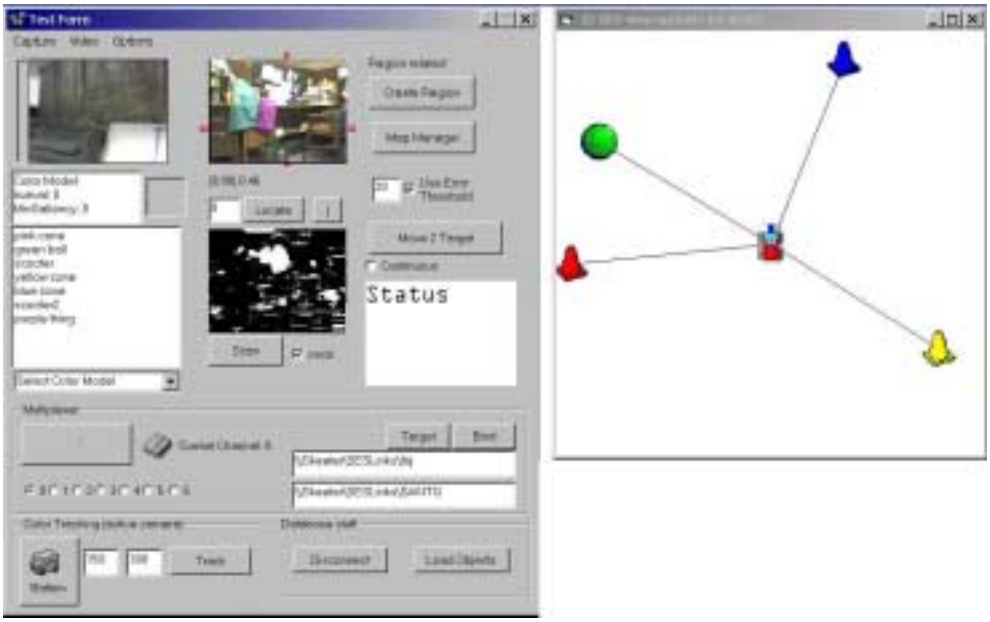

Figure 24 : A SES agent on Skeeter

Sharing the SES data

As we stated earlier in the chapter, Skeeter is responsible for scanning the environment to create the SES data and share it with Scooter. When Scooter receives a new SES data from Skeeter, it will use that SES data to calculate a new heading and try to navigate its way towards the target. For Scooter to be able to retrieve SES data from Skeeter, it will need to create a Peer Agent Client and request SES data from Skeeter, as shown in Figure 25.

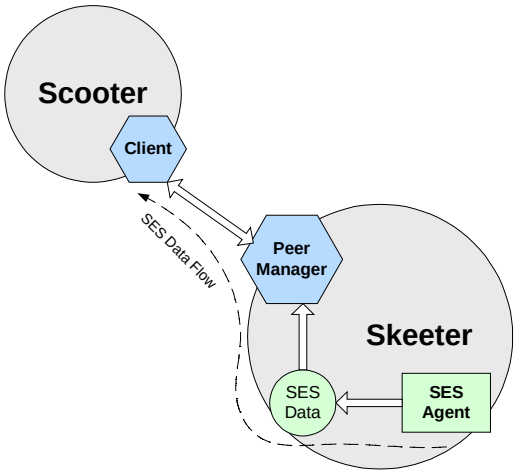

Figure 25 : SES data flow between Skeeter and Scooter

Heading Calculations

For Scooter to be able to move towards the target, specified by the human commander, the heading calculation algorithm is very important. The heading calculation (Figure 26) has two main assumptions as follows:

1. Since SES data may or may not provides distance information, each object will be projected onto the perimeter of the "SES Circle" and assumed to have equal distance,
2. The initial heading of Skeeter and Scooter should be the same.
3. Skeeter has to be able to perceive both Scooter and target.

Definition of parameters

The parameters in this algorithm are defined as follows:

θ_0 : The initial heading of the robots

θ_1 : The angle from Skeeter to Scooter relative to initial heading

θ_2 : The angle from Skeeter to target relative to initial heading

θ_3 : The angle between Scooter to Skeeter and Scooter to target

θ_d : The desired heading for Scooter

θ_h : The current heading for Scooter

a, b : The distance from Skeeter to Scooter and the distance from Skeeter to the target, respectively.

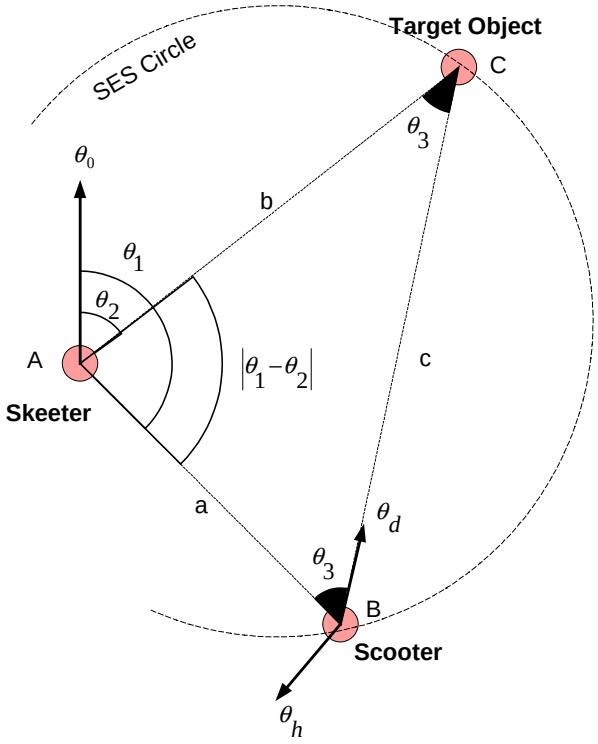

Figure 26 : Heading calculation for Scooter

Algorithm

From θ_1 and θ_2, we can say that the angle between Scooter and the target with respect to Skeeter, is equal $|\theta_1 - \theta_2|$ degree. Because we assumed that each object always locate at the perimeter of the circle, the distance from Skeeter to Scooter (a) and distance from Skeeter to the target (b) will have the same length, and by that the three objects (Scooter, Skeeter and target) form an Isoceles Triangle (ΔABC), which has two equal sides and two equal angle (θ_3). We then can calculate the desired heading for Scooter to move towards the target using θ_3.

Mathematical Prove of the Algorithm

If: side a = side b,

Then: ΔABC is an Isoceles Triangle, which has angle $A\hat{B}C$ and $B\hat{C}A$ equal to each other,

And: $\theta_3 = \frac{180 - |\theta_1 - \theta_2|}{2} = A\hat{B}C = B\hat{C}A$

From θ_3, we can calculate the desired heading for Scooter as follows:

$$\theta_d = 180 + \theta_1 + \theta_3$$

or: Turn angle for Scooter = $\theta_d = 180 + \theta_1 + \theta_3 - \theta_h$

System Performance Evaluation

In this section, we will evaluate the performance of the algorithm by setting up different object configuration scenarios to test run Scooter and calculate the performance of the algorithm and evaluate how the algorithm reacts to different scenario configurations.

Evaluation Setup

Since the algorithm only concerns with one target at a time, we setup the control environment that consisted of only one target, Scooter and Skeeter. There are three variables that can be varied, which are the angle between the Scooter-target (θ), a distance between the Skeeter-target (d_{Target}) and a distance between the Skeeter-Scooter ($d_{Scooter}$). By varying the Skeeter-Scooter's distance and Skeeter-target's distance, will result in changing distance between Scooter-target. And by increasing the angle between Scooter-target will cause Scooter to navigate its way deeper into the SES circle and result in more error in the calculation. If we draw a straight line between Scooter and the target, we will get the minimum distance between Scooter and the target and that line will reflect the best potential path form Scooter to the target ($d_{\min}$). If there is an obstacle on the minimum path, the minimum path will be a curve line that shows the best possible path for the robot to navigate around the obstacle. But since this is a control environment, we assume that there is no obstacle between Scooter and the target and Scooter will has a clear path towards the target as shown in Figure 27. We let Scooter to navigate its way towards the target with constant speed and measured the time that Scooter took to reach the target.

We setup the environment in such a way that three cases will occur in each angle scenario:

1. Set Skeeter-Scooter's distance to be smaller than Skeeter-target's distance, then measure the performance when Scooter started within the SES circle.
2. Set Skeeter-Scooter's distance to be equal to Skeeter-target's distance, then measures the system performance.
3. Set Skeeter-Scooter's distance to be larger than Skeeter-target's distance, then measure the performance when Scooter started outside the SES circle.

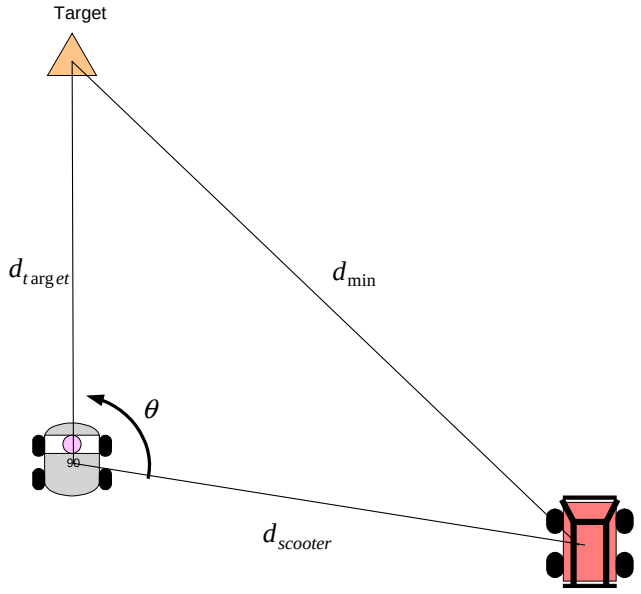

Figure 27 : Evaluation setup for system performance analysis

In this experiment, we varied Skeeter-Scooter's distance and Skeeter-target's distance from two to five meters. The reason we chose these two numbers is because the Skeeter's maximum visual range is five meters. The minimum of two meters is the minimum distance between Skeeter and the target for Scooter to navigate between them. We selected three values of angle between Scooter-target (θ) to be 45, 90, and 135 degrees respectively, and at each angle, we varied the Skeeter-Scooter distance to force Scooter to start at inside, around, and outside the SES circle perimeter as illustrates in Figure 28.

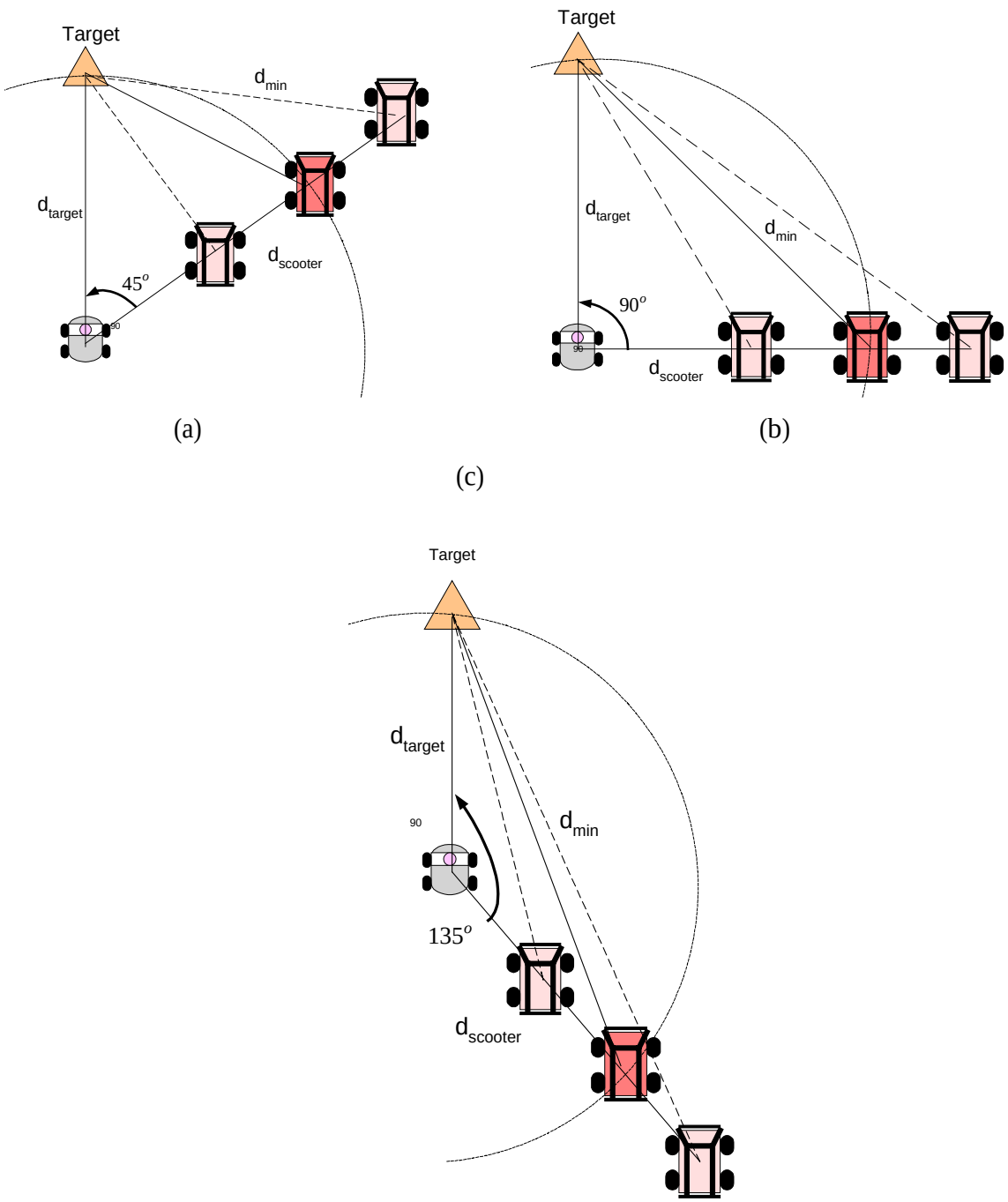

Figure 28 : Three different Scooter position, (a) at 45, (b) at 90, (c) at 135 degrees

Performance Index

We defined a value (f), that reflects how well the algorithm performed, as follows,

$$f = \left[\frac{d_{\min}}{d_{actual}} * \frac{t_{\min}}{t_{actual}} \right]$$

and

$$t_{\min} = \frac{d_{\min}}{v}$$

where

f is the performance index

$d_{\min}$ is the minimum distance from Scooter to the target

d_{actual} is the actual distance that Scooter took to reach the target

t_{actual} is the actual time that Scooter took to reach the target

$t_{\min}$ is a time constant that reflects the minimum travel time if Scooter follows the minimum path.

v is a constant that reflects Scooter's speed.

Value of f reflects the overall performance of the algorithm. Higher f value means better performance and vise versa. The f value will range from 0 to 1, if f equal to one, it means Scooter has taken the minimum path towards the target and this is the best performance, if f is almost zero, it tells that Scooter may have lost its way during the experiment or there are a lot of errors produced by the algorithm resulting in Scooter taken a longer than minimum path towards the target.

Test Runs

As stated earlier, there are three scenarios. First scenario assumed Scooter and the target are located close to each other and have the angle between Scooter-target equal 45 degree. In second scenario, we increased the angle between Scooter-target to be 90 degree to test how the algorithm performed for mid-range distance. Finally, in the third scenario, we increased the angle to be 135 degree. In each scenario, we varied the distance between Skeeter-Scooter and Skeeter-target from two to five meters and test run each configuration and measure required parameters.
The results of each test run are provided in tables in Appendix C.

Performance Analysis

In this section, performance analysis is performed, Note that we have to keep in mind the factors that can affect the performance index, such as errors due to Scooter's base drive unit, or a delay due to Skeeter's complete scan, these factors can cause variations in the trajectory and affect the performance index of each run. Also note that, when Scooter moves closer towards the center of the circle, the algorithm will produce an error in the heading calculation, and this degeneration of the algorithm can cause variations in the performance index as well. We divided this section into three parts as follows,

1). At angle 45 degrees different

We calculated the performance index and show in Table 1. The first column provides a distance between Skeeter-target, the first row provides the distance between Skeeter-Scooter and each cell provides the performance index at each distance configuration.

Dt \ Ds	2	3	4	5
2	0.729483	0.733106	0.788253	0.404853
3	0.297896	0.689179	0.681902	0.479115
4	0.289033	0.39475	0.755004	0.921081
5	0.20513	0.367257	0.518931	0.84914

Table 1 : Performance index at angle of 45 degrees

An easier method to illustrate these numbers is to plot them into graph as shown in Figure 29.

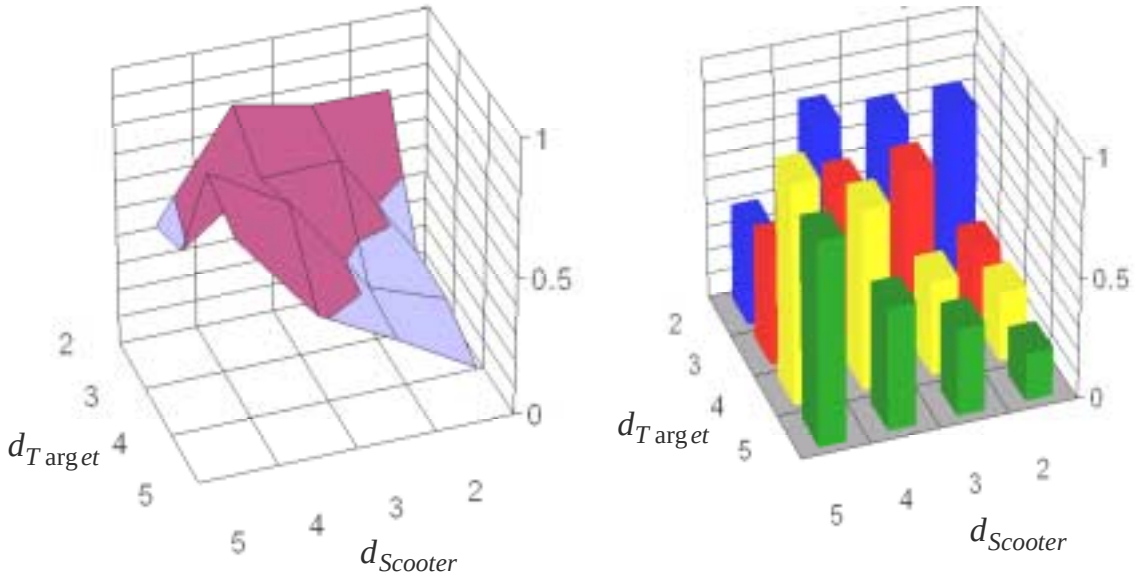

Figure 29 : Graphs plot from performance index at 45 degrees

From the data and graphs, we can see that the algorithm performed better at the configurations where distance between Skeeter-Scooter ($d_{Scooter}$) and distance between Skeeter-target (d_{Target}) are nearly equal to each other than when the Skeeter-Scooter's distance and the Skeeter-target's distance have large different. That means this algorithm performed well at the angle of 45 degrees when Scooter started around the perimeter of the circle and the algorithm degenerated when Scooter started further from the circle's perimeter.

From observing the trajectories of Scooter, we can summarized them into three cases:

- When Scooter started inside the circle, it moved pass the target position then turn back towards it as shown in Figure 30. This due to the degeneration of the algorithm when Scooter is located inside the circle.

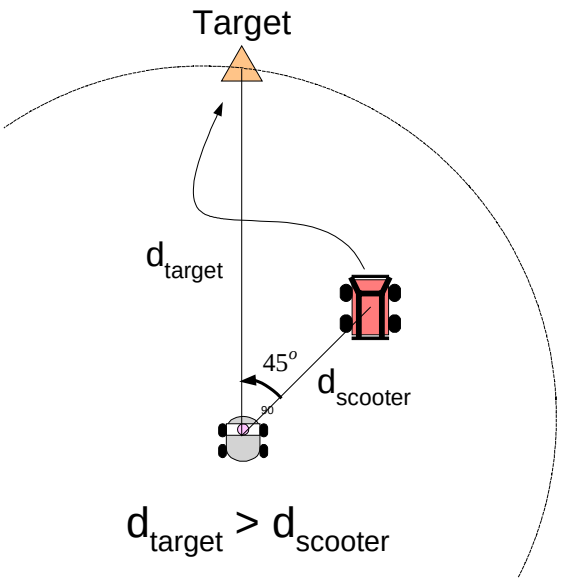

Figure 30 : Scooter's trajectory from inside the SES circle

- When Scooter stared from around the perimeter of the circle, it headed directory towards the target position as illustrates in Figure 31.

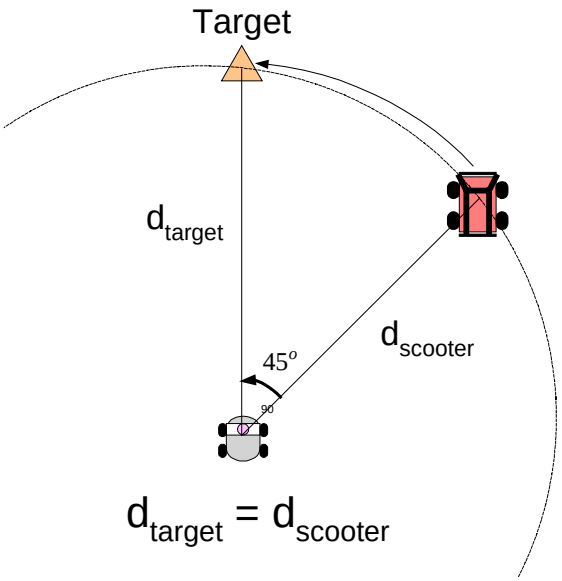

Figure 31 : Scooter's trajectory when located around the perimeter of the SES circle

- When Scooter started outside the circle, it moved pass the target position then turn back towards it as shown in Figure 32. This also due to the degeneration of the algorithm when Scooter is not located on the perimeter of the circle.

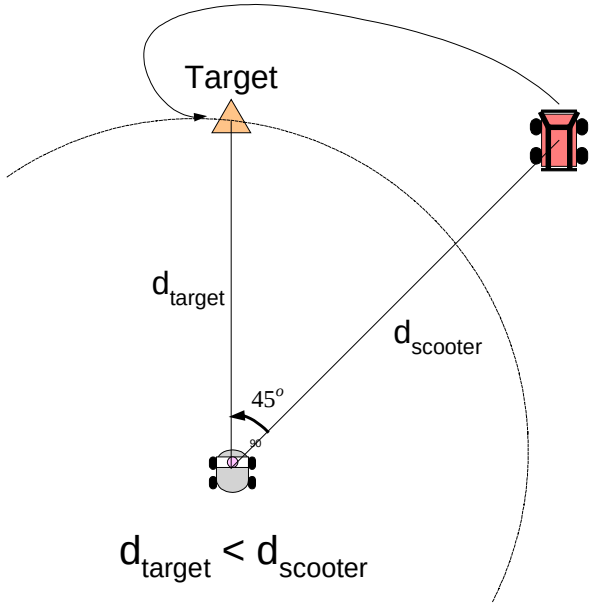

Figure 32 : Scooter's trajectory when located outside the SES circle

From the trajectories, we concluded that the algorithm will give the best performance when Scooter is locate around the circle's perimeter, which also supported by the graphs.

2). At angle 90 degrees different

The performance index is shown in Table 2and the graphs are illustrates in Figure 33.

Dt \ Ds	2	3	4	5
2	0.282636	0.223505	0.822602	0.898502
3	0.27606	0.279666	0.376614	0.805208
4	0.356431	0.344196	0.343091	0.740687
5	0.215914	0.33804	0.377414	0.387746

Table 2 : Performance index at angle of 90 degrees

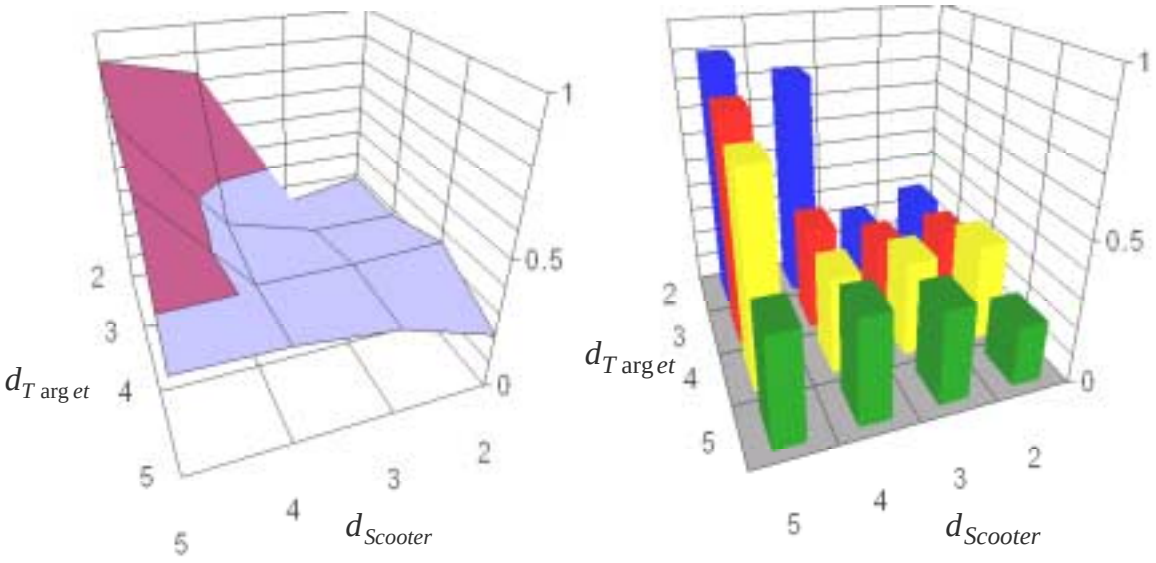

Figure 33 : Graphs plot from performance index at 90 degrees

From the graphs, we see that the performance peaked when the distance between Skeeter-Scooter is large and the distance between Skeeter-target is small and the performance is the lower when the distance between Skeeter-Scooter is small and the distance from Skeeter-target is large.

From observing the Scooter's trajectories, we can also summarized the trajectories into three categories as follows,

- When Scooter started inside the circle, the degeneration of the algorithm caused ripples in Scooter's trajectory as illustrates in Figure 34.

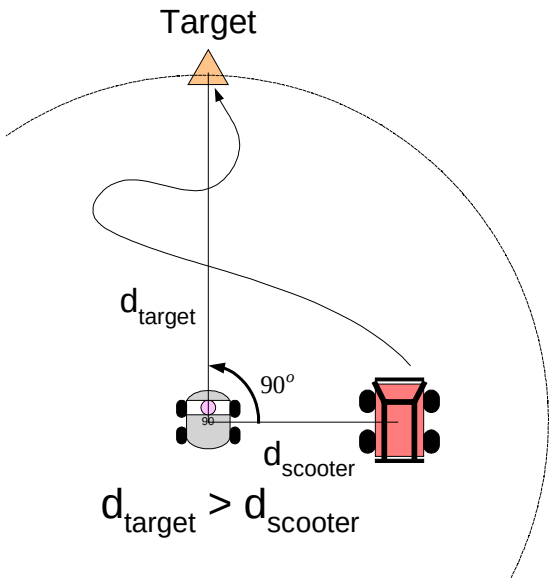

Figure 34 : Scooter's trajectory when located inside the SES circle

- When Scooter started around the perimeter of the circle, the degeneration of the algorithm still caused little ripple in the trajectory as shown in Figure 35.

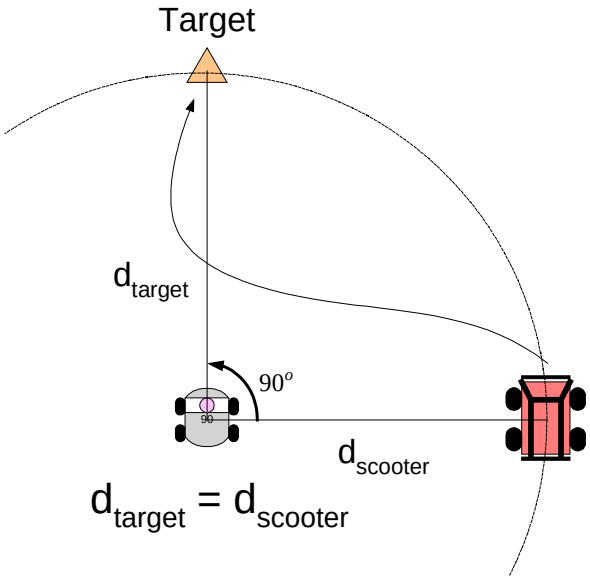

Figure 35 : Scooter's trajectory when located around the perimeter

- When Scooter started outside the circle, there is no ripple in the trajectory as illustrates in Figure 36, Scooter headed directory towards the target.

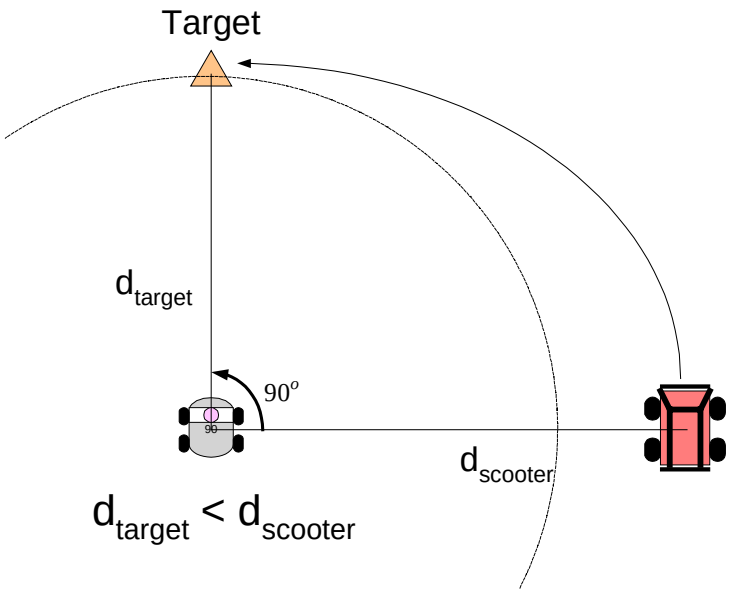

Figure 36 : Scooter's trajectory when located outside the perimeter

Again that the data from the graphs and from observing Scooter's trajectories leaded to the same conclusion, but in this time, the best performance occurred when Scooter is located outside the circle, not on the perimeter of the circle. One hypothesis for this event is that, when the angle between Scooter and the target became larger, Scooter will tend to move into the circle deeper than when the angle is small, that result in higher error and longer trajectory which finally reflex into lower performance. But this larger angle is compensated by larger between Skeeter-Scooter and the larger the distance, the better compensation to the degeneration of the algorithm.

As we stated earlier, one factor that can cause variations in the trajectories is the scanning delay of Skeeter's camera. It takes around five seconds for Skeeter to complete one 360 degrees scan and update the SES data to Scooter. Figure 34 and Figure 35 showed a good example of how this delay can cause the trajectory variation. In Figure 34, Scooter located closer to Skeeter than in Figure 35, and when Scooter navigated at the constant speed, the updates SES data will give larger error in Scooter's position when it moved closer to Skeeter than when it move further away from Skeeter. This error caused more ripple in the trajectory when Scooter located closer to Skeeter.

3). At angle 135 degrees different

We increased the angle to 135 degrees, the performance index is as shown in Table 3. The graphs that corresponded to the table is shown in

Dt \ Ds	2	3	4	5
2	0.305562	0.304619	0.369825	0.277377
3	0.244471	0.261687	0.367509	0.383805
4	0.24602	0.379345	0.324108	0.337755
5	0.168069	0.34867	0.348815	0.336308

Table 3 : Performance index at angle of 135 degrees

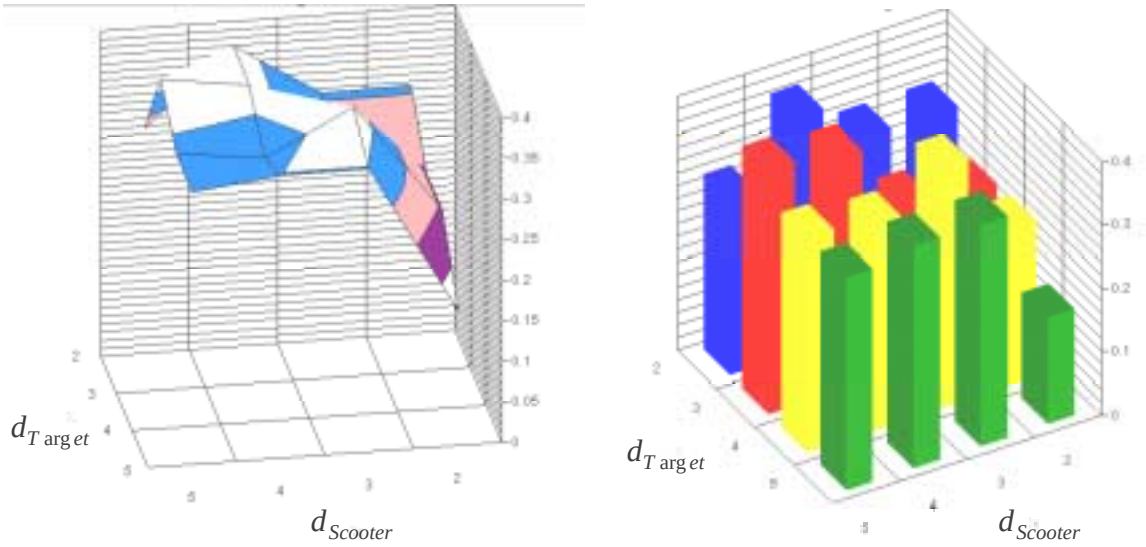

Figure 37 : Graphs plot from performance index at 45 degrees

From the graphs, we see that the performance index only average about 0.3. The algorithm performed well when the distance from Skeeter-Scooter is large then the performance become worst at smaller distance between Skeeter-Scooter. The lowest point is when the distance between Skeeter-Scooter is two meters and the distance between Skeeter-target is five meters.

The Scooter's trajectories in this scenario are almost the same in every configuration. The only different is that the trajectory will contain more ripples when the distance between Skeeter-target becomes larger. Two types of trajectories are shown in Figure 38, smaller distance between Skeeter-targer resulted in fewer ripples.

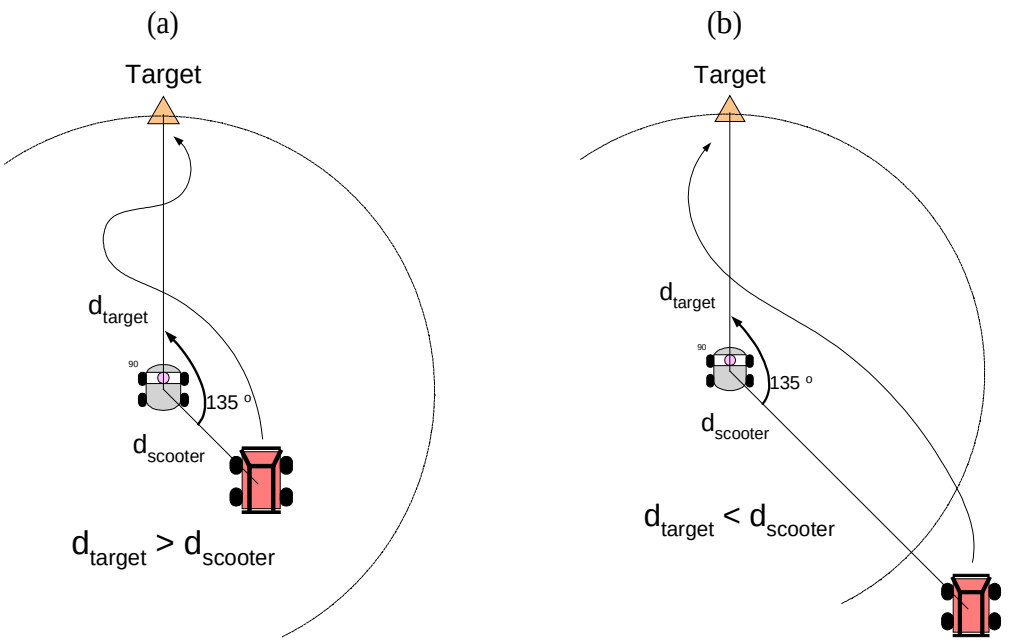

Figure 38 : Scooter's trajectories when, (a) Larger distance between Skeeter-target, and (b) Smaller distance between Skeeter-target

As seen from the graphs and described in the trajectories observation, the algorithm performance for this scenario is low due to larger angle between Scoter-target that forced Scooter to move into the circle and resulted in larger error in the calculation.

Evaluation Conclusions

From three scenarios described earlier, we can conclude that the algorithm performed better in a scenario where angle between Scooter and the target is small. And if the angle between the Scooter and the target become larger, the algorithm performed better when there is a compensation in Skeeter-Scooter's distance. We listed the advantages and disadvantages of this algorithm as follows,

Advantages of the algorithm

- Since this algorithm uses SES data (which may or may not include range) in the calculation, precise distance is not needed in this algorithm.
- The algorithm adapts quickly to error in the calculation and corrects it.
- Simple calculation and consumes little computational resources.

Disadvantages of the algorithm

- Scalability issue: the key to the algorithm is the range of the camera on Skeeter. The limitation of this algorithm is that it can only be applied when Skeeter can see both target and Scooter.
- Distance has an effect on precision of the algorithm's calculation. If Scooter is not located around the perimeter of the circle, algorithm produces an error in the calculation.

The Heading Calculation Agent

In an experiment, we will have an agent that will take care of calculating the heading for Scooter running inside the robot's processor as shown Figure 39. In this figure, the eye in the middle of the circle shows Skeeter's position, the cart shows the position of Scooter, the balloon shows the position of the target and the dotted line shows the new heading from Scooter toward the target.

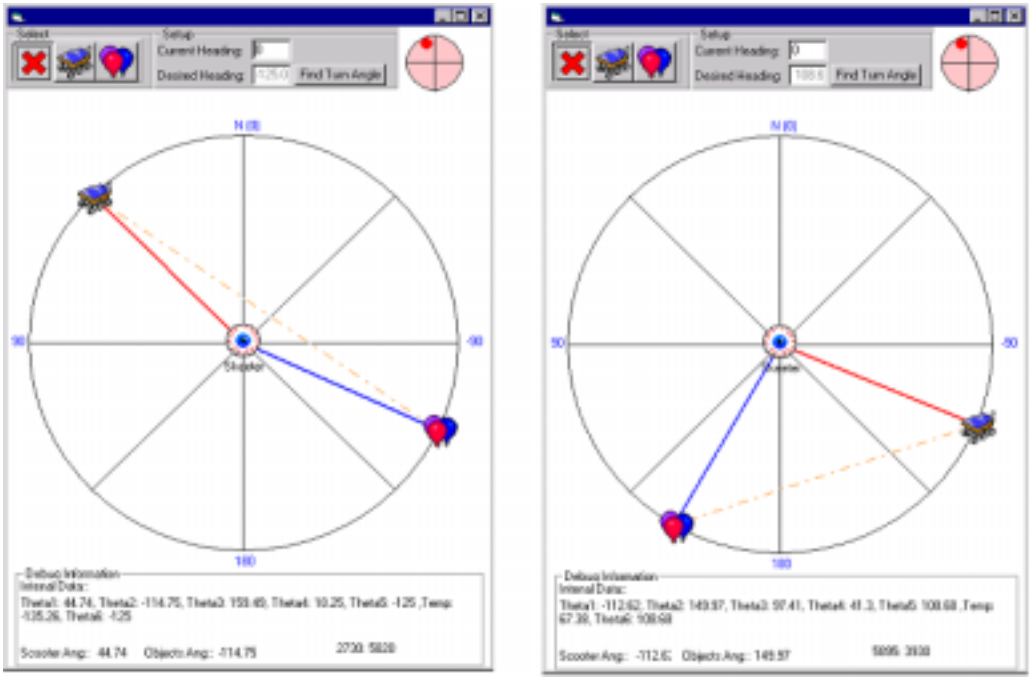

Figure 39 : The heading calculation agent

Commander Interface Agent

The Commander Interface Agent role is to receive commands from a human commander and interpret them into instructions that each robot will understand, and then send them to the

robot. On the other hand, when a robot broadcasts something to the group, the Commander Interface Agent, which is tapped into the robot's broadcasting channel, will receive the data and notify the human commander about the situation of the robots.

The Commander Interface Agent allows the human commander to choose a series of color of the targets before sending them to the robots and shows the progress of the task in real time as shown in the interface window in Figure 40.

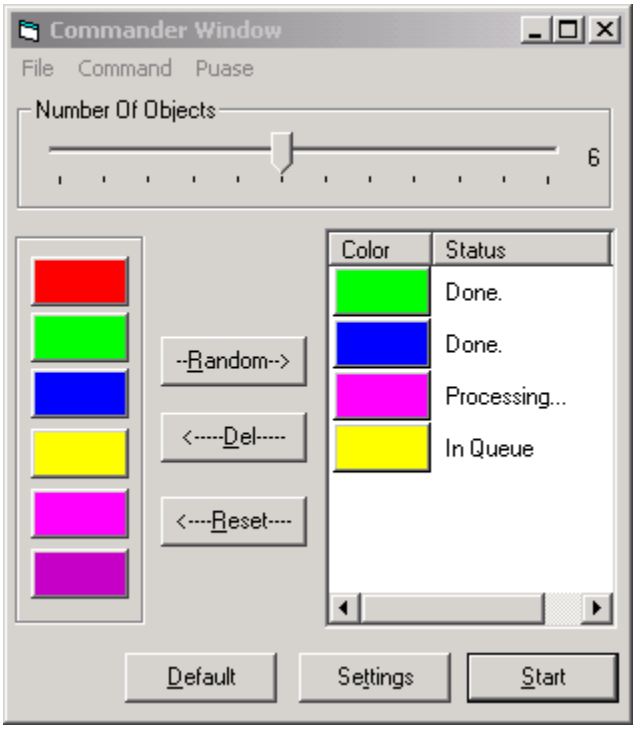

Figure 40 : Commander Interface Agent Window

Knowledge Sharing/ Data Flow between Agents

Figure 41 illustrates the communication flow between the robots and the Commander Interface Agent. The Commander Interface Agent will send out the mission objective, which is the color of the target, and will receive the status (conditions) and behavioral data from each robot. Skeeter will be responsible for creating the SES data and determining new headings for Scooter. Scooter will provide its' headings to Skeeter and will notify Skeeter when it has reaches the target.

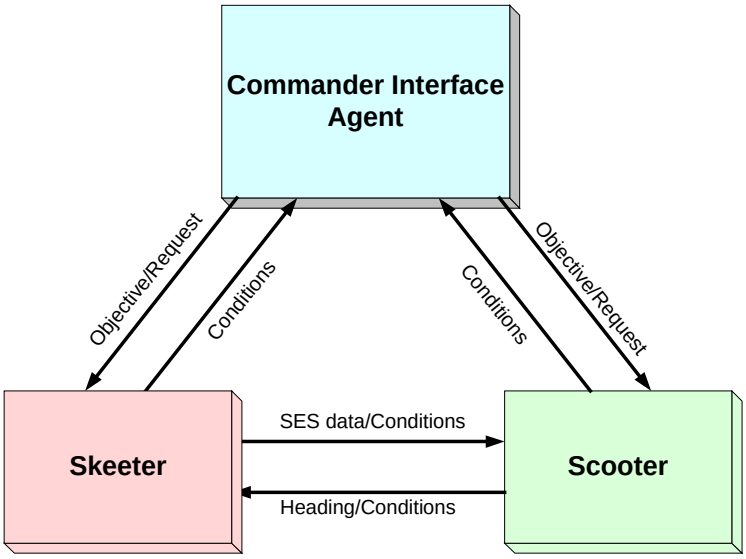

Figure 41 : Data flow within the robot group

One of the most interesting parts is when the robot broadcasts its' own conditions to the group, and other robots sense the change and try to also change or adapt their own behaviors to allow the group to achieve the given task. One example of this behavioral change is when Skeeter scans the area and cannot locate Scooter, it will broadcast a stress call to the group that Scooter has been lost. When Scooter receives this call, it will change its behavior to a Move-to-home behavior. It will head back towards the center of the circle and try to go back to Skeeter to be within Skeeter's scanning range.

Skeeter's SAN Network

Skeeter's SAN Network is the combination of condition nodes and competency module nodes that are connected together to make a planning/decision-making module of the robot. The circle-shape figures are the condition nodes and the rectangle-shape figures are the competency module nodes. If the figure is in a shaded rectangular area, it means that the particular figure is connected to other robots or the Commander Interface Agent and it can pass/receive the activation flow to/from other robots or the Commander Interface Agent using the Peer Agents. The arrow shows the order of the activation flow between each node in the network. Figure 42 illustrates the SAN Network process for Skeeter.

We can describe the functionalities of this SAN Network as follows:

- If Scooter requests a heading, then send current heading data
- If Skeeter loses visual contact with Scooter, then call Scooter back
- If Skeeter cannot see the target, then ask Scooter if it can see the target and if Scooter cannot see the target, then notify mission failure to the Commander Interface Agent
- If timeout (uses more time than estimated), then notify mission failure to the Commander Interface Agent
- If Scooter reaches the target, then notify mission accomplished to the Commander Interface Agent.

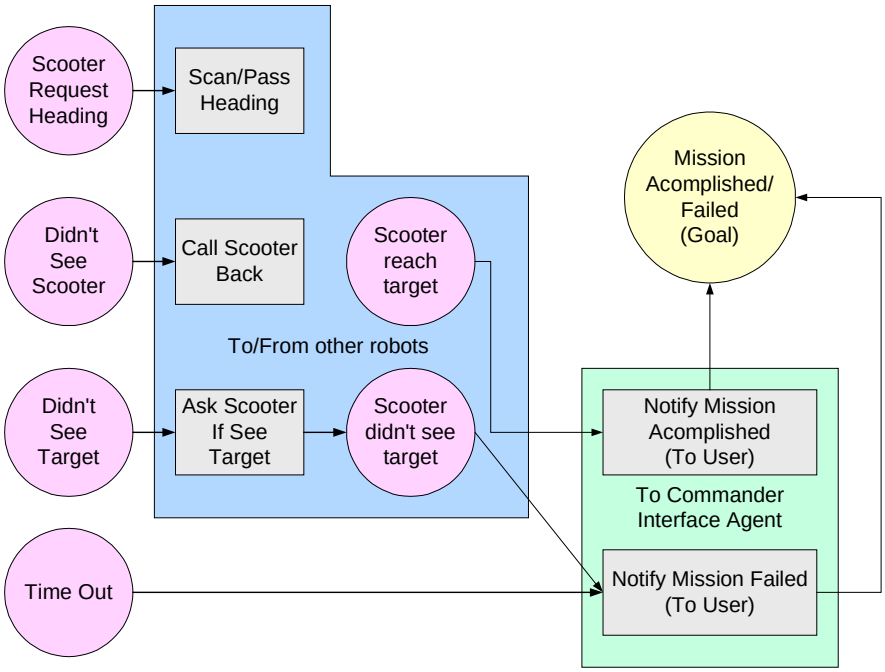

Figure 42 : Skeeter's SAN Network

Scooter's SAN Network

The SAN Network of Scooter, as shown in Figure 43, has similar concepts but contains different condition nodes and the competency module nodes.

The functionalities of Scooter's SAN Network are as follows:

- If lost then, request a new heading from Skeeter and upon receiving a new heading, moves towards the new heading
- If sees target, then go towards target
- If Skeeter calls it back, then return to Skeeter
- If Skeeter asks if it see the target, then reply to Skeeter
- If it sees an obstacle, then try to avoid it until the path is clear and continue towards the heading
- If target is found, then notify Skeeter.

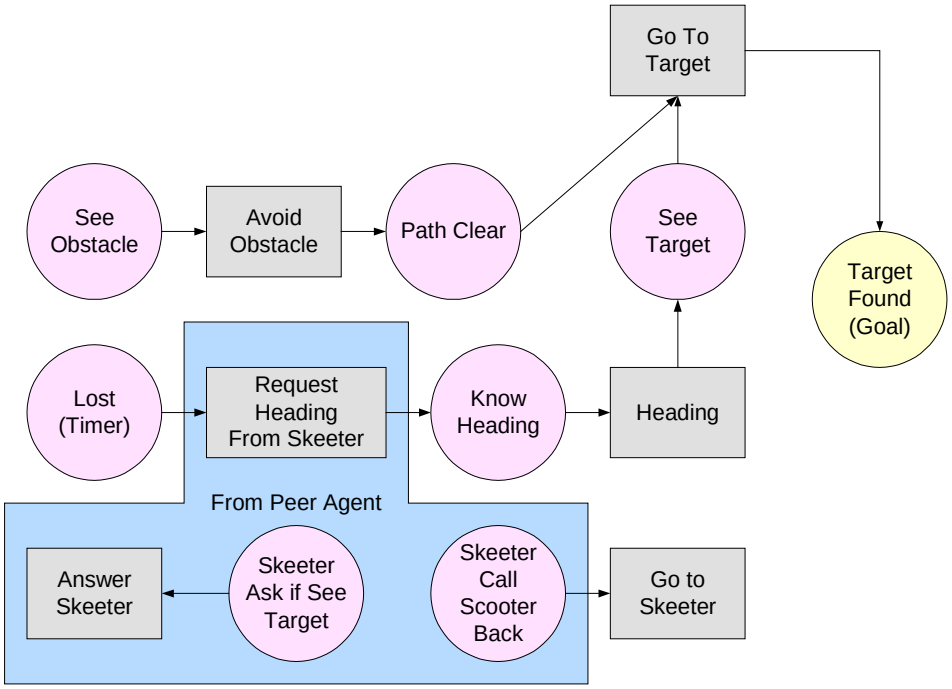

Figure 43 : Scooter's SAN Network

Network Layout

Figure 44 shows the communication network between the Skeeter, the Scooter and the Commander Interface Agent. Within each robot, there is a Spreading Activation Network (SAN) that acts as a decision-making agent of the robot and tells what should be done in a sequence, especially when particular conditions occur. Each robot will have the Peer Agent Manager to monitor and manage the communication between other robots and the Commander Interface

Agent. Peer Agent Clients will be created whenever the robot or the Commander needs to connect and gather data or information from other robots.

Between each Peer Agent Manager there is a broadcast channel that is used to share some behavioral and status data within the Peer Agent Managers. The human commander can tap into this channel using the Commander Interface Agent and monitor the performance of each robot.

SES data will be created by SES agent on Skeeter and stored within the robot. The Commander Interface Agent and other robots will create the Peer Agent Clients and connect to Skeeter's Peer Agent Manager if they require this data.

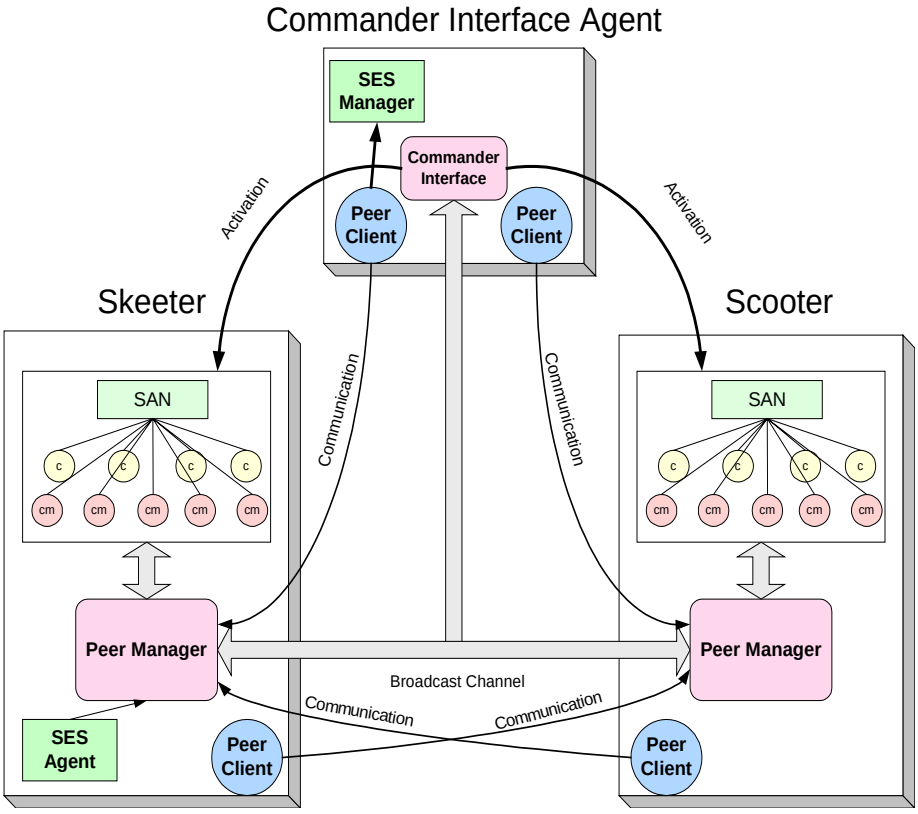

Figure 44 : Layout of the communication network

CHAPTER VI

CONCLUSIONS

In this thesis, the concept of the Peer Agent has been developed to use in robot communication. The Peer Agent consists of two parts: 1) the Peer Agent Manager, which attached itself to the robot's Self Agent and acts as an information server to other robots, and 2) the Peer Agent Client, which is created when other robots require the information. It acts as a communication channel that connects the information source and retrieves the data. When retrieving information, the Peer Agent will monitor the stability of the channel and will close the communication channel if it is damaged or terminated.

Between the robots, there can be two types of communication channels:

1. Channels between the Peer Agent Managers and the Peer Agent Clients: This channel is used for sending typical information, e.g., sensory information or SES information, etc., between robots in the group.
2. Channel between each Peer Agent Manager: This channel is used to broadcast some task-related information between the Self Agent of the robots, e.g., current behavior or current state of the robot, etc. The Commander Interface Agent will be able to monitor this channel and use this information to determine next tactic for the group.

Each channel has a different purpose and should not conflict with one another. This will enhance the performance of the communication within the robot group.

Every request made by the Peer Agent Client to the Peer Agent Manager will be tagged with a priority value. The priority value is calculated from the priority of the requested robot, type of request and source robot's task. The Peer Agent Manager will list and sort the entire current request by priority and will execute the request that has the highest priority first.

In our experiment, we used two heterogeneous all-terrain mobile robots, Skeeter and Scooter, and proved that sharing information can occur between these robots. Skeeter will create the SES and shared the data with Scooter. In the meantime, Scooter will use the information to calculate its' new heading that can lead it towards the target. The human commander will be able to monitor each robot state by tapping into the broadcast channel between the robots' Peer Agent Managers.

Another interesting issue is when we use the broadcast channel between the Peer Agent Managers to share robots' behavior with the group. Each robot's SAN network, which acts as a robot brain, can detect changing in other robots' behavior and will try to adapt to a more suitable state and force the behavioral change within the robot itself. In the experiment, we showed that the change in condition of one robot can stimulate the condition change within another robot e.g., if Skeeter loses track of Scooter, it will panic and try to call Scooter back, then Scooter, when sensing that Skeeter is panicking, will try return to its' home position.

We assumed many assumptions to be able to produce an algorithm that roughly calculates Scooter's heading. These assumptions were not met all the time and the algorithm produced an error but we proved that it still robust enough to be able to produce acceptable results.

We proved that communication between robots could be used to solve tasks, which may be too difficult for only one robot to achieve and it leads to a more efficient group performance.

Recommendation for Future Work

As discussed throughout this thesis, future research direction included the following issues,

1. Replace Skeeter's camera head with human commander or more sophisticate devices to increase scalability and speed of the algorithm.
    - We can replace Skeeter with a human equipped with a Personal Digital Assistance (PDA) such as Pocket PC or Palm Pilot and then he can tell Scooter where the target is instead of Skeeter. And Scooter can accept command directly from the human commander.
    - Instead of using Skeeter' camera to scan the environment from a horizontal view, we can use other kind of devices, such as airplane's camera (pointing down) or satellite's camera to scan in an vertical view. This can increase the visual range and the algorithm can be applied in a larger scale.

2. Improve the performance of the algorithm.

    As described in Experiment Chapter, the performance of the algorithm varied with the distance between Skeeter, Scooter and the target. If we are able to correlate distance information from ranging sensors, such as Sonar or LIDAR, with

information from SES, we should be able to greatly improve the performance of the algorithm.

3. Integrate the algorithm with Perception-based Navigation

   Instead of using SES data to calculate the heading, we can replace SES with LES from Perception-based Navigation and apply the algorithm; we should be able to expand the working area of the algorithm.

4. Improve object avoidance behavior

   Sometimes when the robot moves close to an object, the object avoidance behavior is activated resulted in robot moving back and forth in a “jerking” motion. If we can improve the object avoidance behavior to make robot run smoother, it will improve the performance of the algorithm.

APPENDIX A

AGENTS USED IN THIS EXPERIMENT

In this thesis, the IMA agents used in the experiment can be categorized into four groups, IMA Agents for Skeeter, Scooter, Human Commander and Peer Agents, as follows:

IMA Agents for Skeeter

1. SAN Manager for Skeeter: This is a graphical interface agent that is used to create SAN Network for robots and monitor the status of each node in the network.
2. SAN Node Agents for Skeeter: These are the agents that represent each condition node or competency module in the robot's SAN Network:
    - Timeout Node: This agent will keep track of the time in a robot's task, if the task takes more time than user had specified, this agent will alert the SAN by become true.
    - Find Scooter Node: This agent will monitor the position of Scooter from the SES data. If it cannot find Scooter, its state will become false.
    - Find Target Node: This agent will monitor the position of the target from the SES data. If it cannot find Scooter, its state will become false.
    - Mission Accomplished Module: If the SAN activates this module, it will send an event to the Commander Interface Agent that the mission has been accomplished.
    - Mission Failed Module: If the SAN activates this module, it will send an event to the Commander Interface Agent that the mission has failed.
    - Scan Module: This agent is responsible for generating the SES data for Skeeter using the omni-directional camera head.
3. Peer Agents: Skeeter will have one Peer Agent Manager to manage the communication channels.

IMA Agents for Scooter

1. SAN Manager for Scooter: This is a graphical interface agent that is used to create a SAN Network for robots and monitor the status of each node in the network.
2. SAN Node Agents for Scooter: These are the agents that represent each condition node or competency module in the robot's SAN Network:
    - See Obstacle Node: This agent will become true when the robot senses that there is an obstacle in its path.
    - Lost Node: This agent will become true when Scooter feels that it is lost, usually we use a timer to make a simple lost behavior.
    - See Target Node: If Scooter sees the target, this agent will become true.
    - Target Found Node: This is a goal node for Scooter.
    - Heading Module: This agent is used to control Scooter to head into a desirable direction.
    - Goto Skeeter Module: Scooter will use this agent to navigate its way back to Skeeter (home position).
    - Avoid Obstacle Module: This agent will help Scooter to navigate its way around the obstacle.
    - Goto Target Module: This agent is used to control Scooter to head towards the target direction.
3. Peer Agent: Scooter will have one Peer Agent Manager and one Peer Agent Client that are used to retrieve SES data from Skeeter.

IMA Agent for Human Commander

1. Commander Interface Agent: This is a user interface agent that accepts a series of targets from the user and sends them to the robots.
2. SES Manager Agent: This agent will gather SES data from Skeeter and display them on a graphical interface for the user.
3. Peer Agent: Commander Interface Agent will have one Peer Agent Client that is used to gather SES data from Skeeter.

APPENDIX B

DETAIL OF AGENTS

This section will provide a detailed description of each agent used within the scope of this thesis and show how each agent interacts with each other.

The Peer Agent Manager

The Peer Agent Manager has five pages in its interface as follows:

1. Data Page: This page is for the user to label the address for the sensors information or other sources of information within the robot. When the Peer Agent Client connects and register itself to the Peer Agent Manager, the Peer Agent Manager will encode all the addresses into a string (called “Competency String”) and send it to the Peer Agent Client. The Peer Agent Client then will know what kind of information can be retrieved from this robot. The user can add node addresses, delete them, or save them into file and retrieve them back for later use. Figure 45 shows the graphic interface of this page.

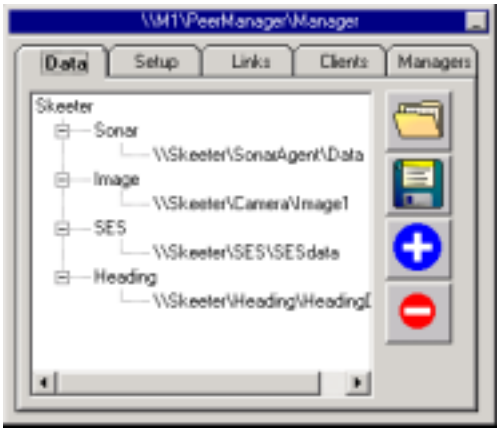

Figure 45 : Source addresses page

2. Setup Page: The second page contains the tool that is used for finding the sources of information within the robot. Thus, one robot can be the combination of many computers working with each other. Users can find the address of information within the robot domain by adding every

computer that locate within the robot domain and execute a search for address of information. After the computers in the robot domain have been added and the search has been executed, every sources of information will be displayed in the left window and the user can add them to the Data page (First page) for further uses. Figure 46 shows the setup page.

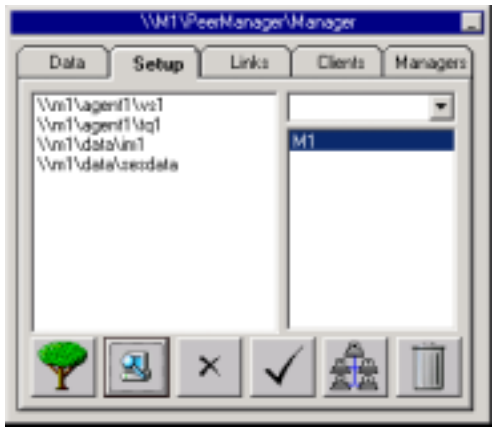

Figure 46 : Setup page

3. Links page: The third page contains links that the Peer Agent Manager uses to connect to the Peer Agent Client and with other Peer Agent Managers. This is the first page that should be setup. There are a total of five links: two links communicate with the Peer Agent Clients, two links connect with the Self Agent, and the fifth link broadcasts task-related information within the group of connected Peer Agent Managers. This links page is shown in Figure 47.

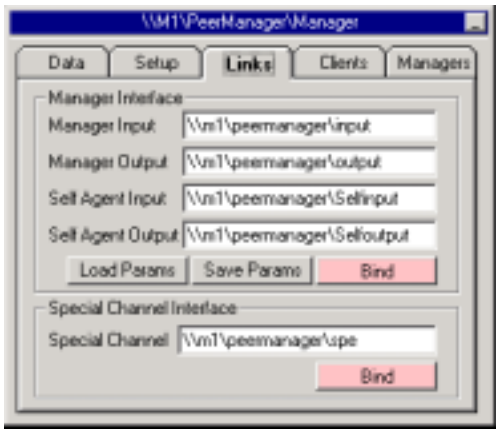

Figure 47 : Links page

4. Clients page: This page is the monitor page that will display the current status of the connected Peer Agent Clients. It will display the ID, address and the status of each client. Figure 48 shows the interface of this page.

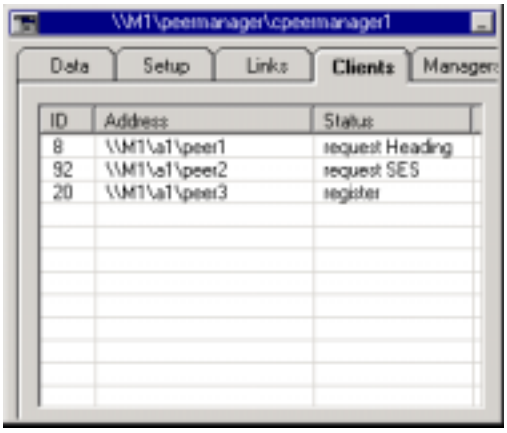

Figure 48 : The Peer Agent Client status page

5. Broadcast channel page: The last page is used to display the broadcast channel information that between each Peer Agent Manager. It will contain the name and message of the source robot as shown in Figure 49

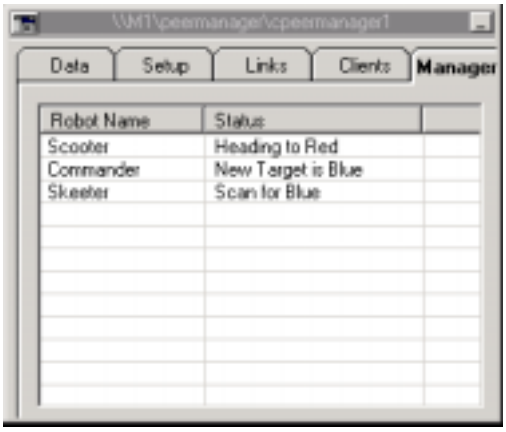

Figure 49 : Broadcast channel page

The Peer Agent Client

The Peer Agent Client has three pages in its interactive window.

1. Path page: This page contains the path of the source of information and the target destination of the information. When the user clicks connect, the Peer Agent Client will automatically connect both source and target together and start retrieving the data from source and put it in the target destination. Figure 50 shows the graphic interface of this page.

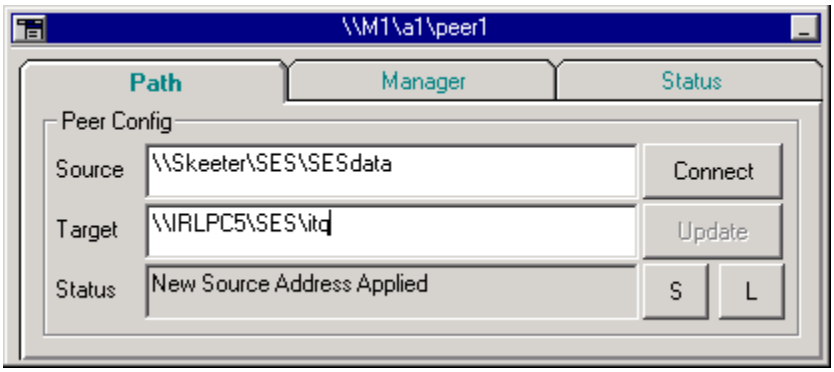

Figure 50 : Path page of the Peer Agent Client

2. Manager page: This page contains the links with the Peer Agent Manager. When the user clicks connect, the Peer Agent Client will try to connect with the Peer Agent Manager. After the connection has been established, the Peer Agent Manager will send the "Competency string" to the client to display them for the user on the bottom of the page as illustrated in Figure 51.

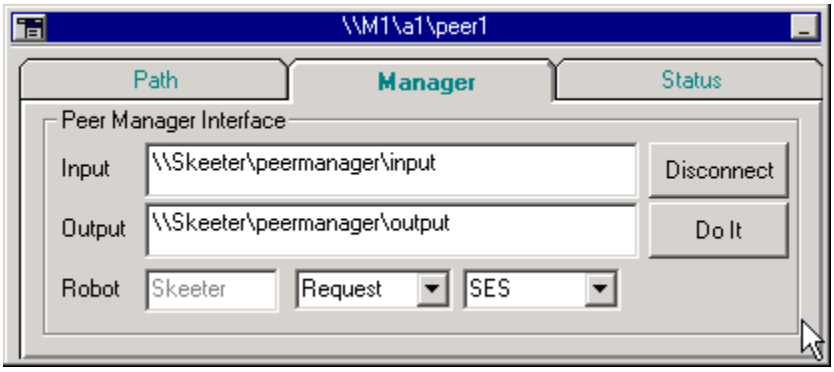

Figure 51 : Manager page of the Peer Agent Client

3. Status page: This page displays the current status of the Peer Agent Client by showing the name of the source and target, the address, and also the type, as shown in Figure 52. If the Peer Agent Client is also linked with the Peer Agent Manager, this page will show the name of

the robot that this client is connecting with and the relationship between the Peer Agent Client and the Peer Agent Manager.

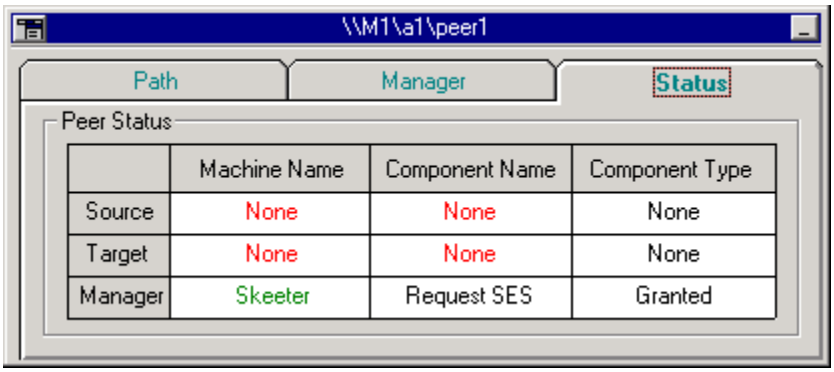

Figure 52 : Status page

The SAN Manager

The SAN Manager is a graphical user interface for the user to create a SAN Network for robots. The network consists of the condition nodes, competency modules and links between each node in the network. The user will be able to drag-drop each node into the designing area and create links between nodes to generate the action selection network in the robots as illustrated in Figure 53.

The user will also be able to specify each node's weight, which the SAN network uses to determine the sequence of the action flow. The higher the node's weight, the higher the node's priorities to SAN network, which will execute the higher priority node first if there are more than one node activated at the same time.

Figure 53 : SAN Manager

As described in Chapter V, we can construct the SAN network of Skeeter and Scooter as shown in Figure 54 and Figure 55, respectively. The circle shape with the letter “C” inside represents a condition node, and a square shape with the letters “CM” inside represents a competency module.

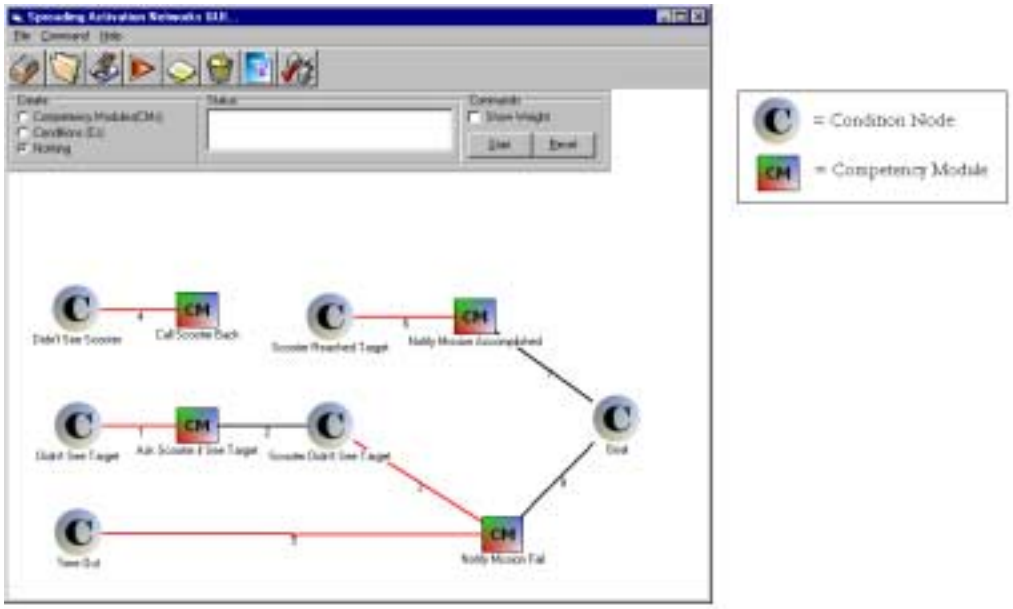

Figure 54 : SAN Manager for Skeeter

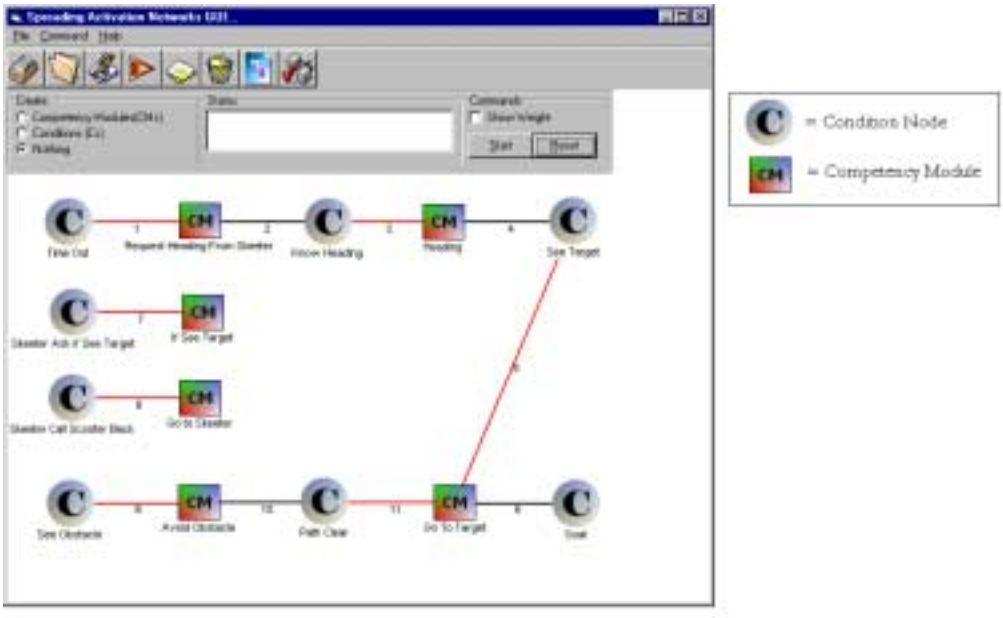

Figure 55 : SAN Manager for Scooter

The SAN Manager is a display and sequencing agent that shows the user which node has been activated and which node has not. The real agent used to connect the robot's hardware or to calculate the data will run separately outside this SAN Manager within each robot, the SAN Manager will has a input and output channel that communicate with these agents and monitor their states.

SAN Nodes on Skeeter

Since Skeeter has only microprocessor on board the robot, every agent used in this experiment are running on board the laptop's processor.

Timeout Node

The Timeout Node counts down the time that robots are allow to work on the task. If the time runs out, this agent's state will become true and notify the SAN Manager. Figure 56 shows the interface of the Timeout Node in which the user can input the specific time the robots need on a given task and monitor how much time is left. The user also able to program this agent to stop or loop back and start counting again after time has run out.

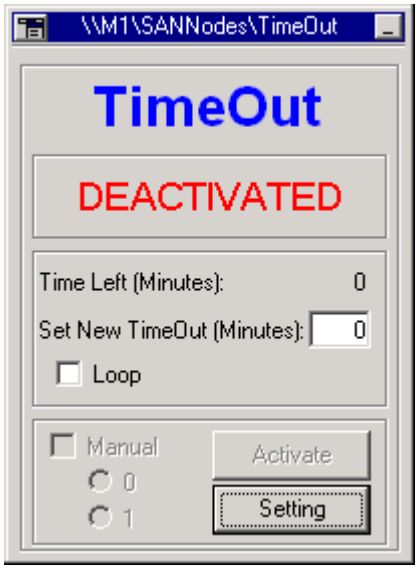

Figure 56 : TimeOut Node's interface

Detect Node

The Detect Node accepts SES data from the Scan node and searches for a specific object, e.g. Scooter, blue objects, or green objects, within the SES data. If the object is found within the SES data, its state will become true and notify the SAN Manager. Figure 57 shows the graphical interface of this agent.

Figure 57 : Detect Node interface

Mission Accomplished Module

The SAN Manager will activate the Mission Accomplished Module when the robot wants to notify the human commander, via The Commander Interface Agent, that the given task has succeeded. Figure 58 illustrates the interface for Mission Accomplished Module.

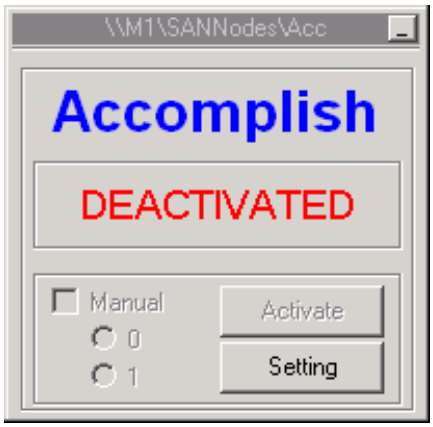

Figure 58 : Mission Accomplished Module interface

Mission Failed Module

The SAN Manager will activate the Failed Module when the robot wants to notify the human commander, via The Commander Interface Agent, that the given task has failed. Figure 58 illustrates the interface for the Mission Accomplished Module.

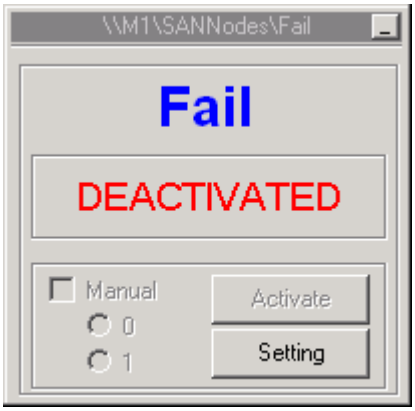

Figure 59 : Mission Failed Module interface

The user can change between each node/module by right-click of the mouse on the title of the node and select the desirable node (except Scan Node) as illustrated in Figure 60.

Figure 60 : Changing between nodes

Scan Node

Scan Node is an agent that runs on Skeeter and is responsible for using the omni-directional camera head to create SES data. In each camera, it will capture a frame and compare it with the database of colors, which have been predefined by the user. If the image contains objects that match the specific colors, this agent will post those objects on a screen with an angle relative to Skeeter. In Figure 61(a) shows a scan agent's graphical interface. Figure 61(b) shows a post screen from the agent.

(a) (b)

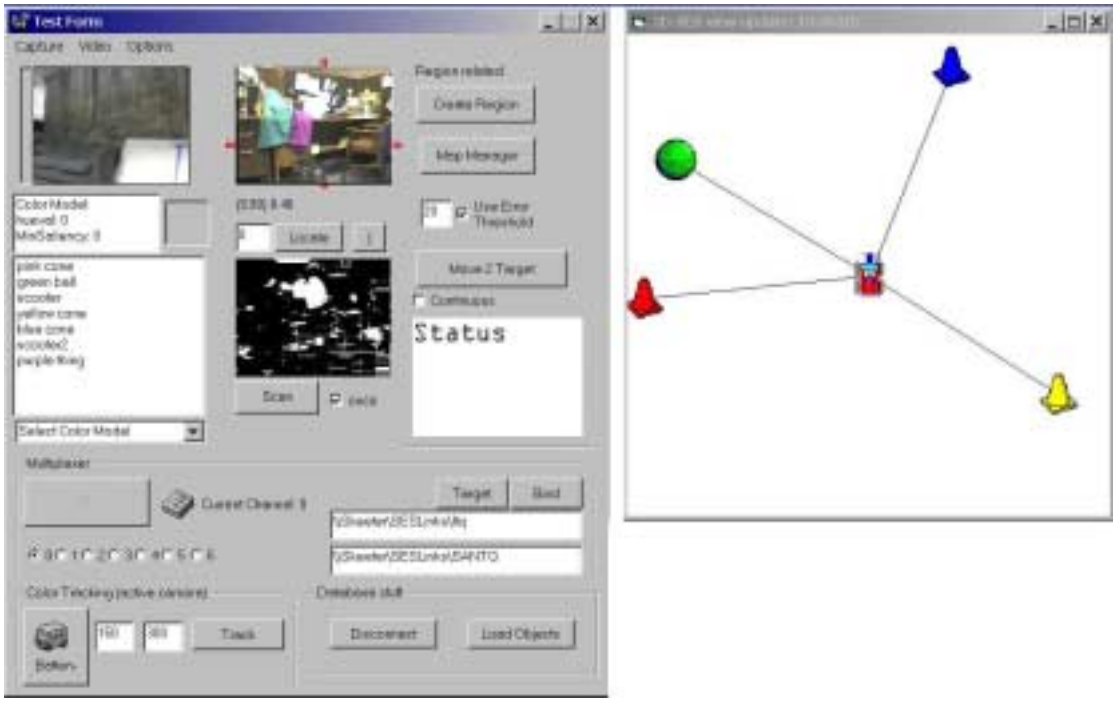

Figure 61 : Scan Node example on Skeeter

Before the user can use this agent, the user must initialize the multiplexer that connects to the camera-head first, then load a database into the agent. From Figure 61(a), the top left-most frame shows a real time image captured from the camera and the middle frame shows the debugging frame used by user.

SAN Nodes for Scooter

Almost all of Scooter's agents are running onboard Scooter's processor, e.g. Avoid-Obstacle Agent, Detect-Target Agent, Goto Target Agent. We use a laptop to connect to Scooter's processor via TCP/IP protocol, as shown in Figure 62. This agent will communicate with agents that running inside Scooter' processor and retrieve the data out to show to the user. The data that communicate between the agents is the state of the condition, which is either true or false.

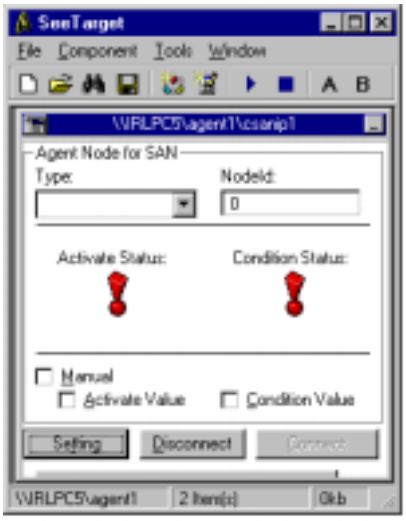

Figure 62 : Agent that communicate with Scooter' processor via TCP/IP

The reason we are conducting the experiment in this manner is because Scooter's processor uses the Unix Operating System while the laptop uses Microsoft Window Operating System. The easiest way to communicate across the network for Scooter is to use the TCP/IP protocol.

In addition to the agents that are running on the Scooter' processor, a few agents are running on the laptop's processor such as Heading Agent and Lost Node, which are described in the following sections.

Heading Node

The Heading Agent is the agent that controls the heading of the robot using either a Compass or Odometry (by counting the encoded data on the robot's motors). Figure 63 shows the graphical interface of the Heading Agent. The user has the ability to change the Linear, Angular and Avoid-Obstacle velocity of the robot and has a choice between using the robot's compass or DMU (Drive Motor Unit) or Odometry for heading reference. This agent connects directly with the Heading Calculation Agent and accepts a new heading data from it.

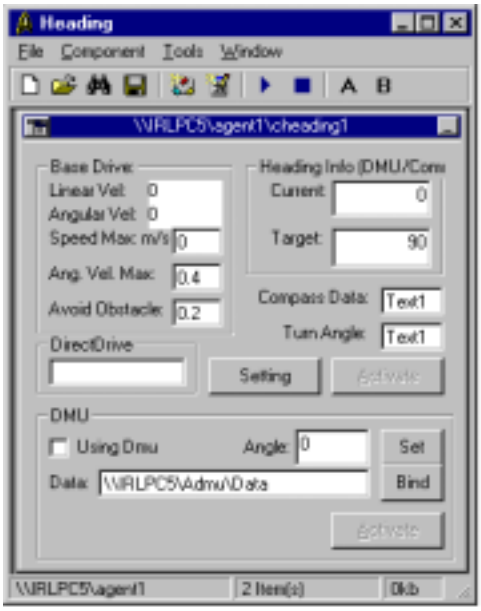

Figure 63 : The Heading Agent graphical interface

Heading Calculation Agent

The Heading Calculation Agent is the agent that calculates a new heading for Scooter to be able to navigate itself towards the target. This agent accepts SES data from the SES-making Agent that is running in Skeeter and calculates a new heading from Scooter to the target. In Figure 64, the eye represents Skeeter, the cart represents Scooter and the balloons represent the target. The lines represent the angle from Skeeter to Scooter and from Skeeter to the target. The dotted line represents a new heading for Scooter. Calculation happen every time SES data is updated.

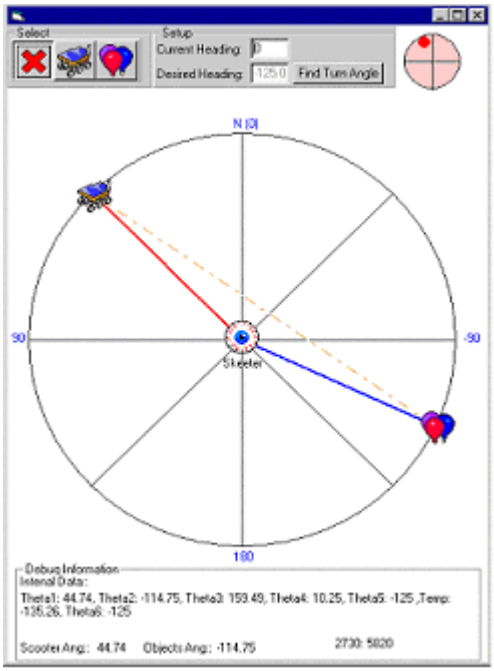

Figure 64 : Heading Calculation Agent user interface

Commander Interface Agent

The Commander Interface Agent, explained previously in Chapter V, interacts with the human commander and displays some task-related information to the human. The human will be able to select the color of the target and monitor the progress of the given mission. First, the user has to setup the links between the Commander Interface Agent and the robots. Then, the user can select a series of colors from the left part of the window and stack them together before sending them to the robots. If the target has been found, it will be labeled as “Done”, or if the robots are still trying to find the particular target, it will label the target as “Processing…”, etc. At the top of the window, the user can specify a number of objects, which can be in a series before sending them to the robots. Figure 65 shows the graphical interface of the Commander Interface Agent.

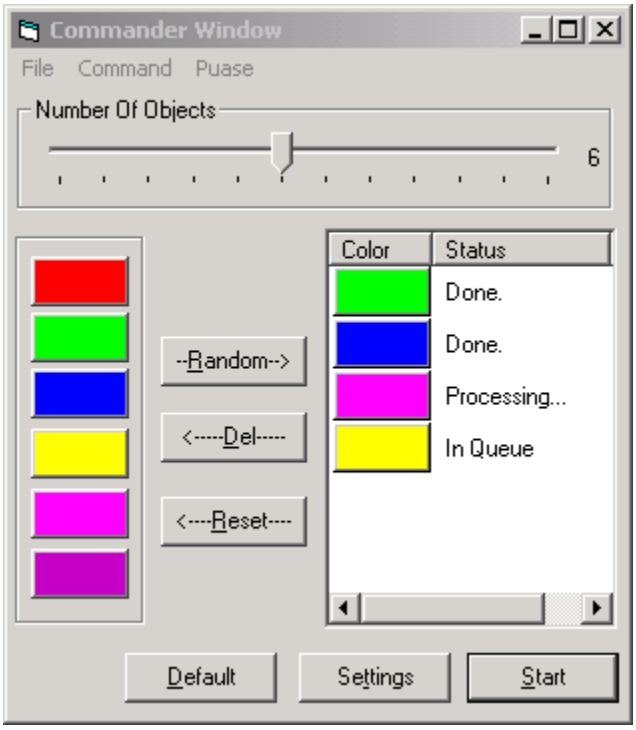

Figure 65 : The Commander Interface Agent

APPENDIX C

TEST RUN DATA

This section provides all the data gathered during the performance evaluation test run. Table 4, Table 5, and Table 6 represent data gathered from each scenario, which are 45, 90, and 135 degrees respectively.

Within the chart,

Ds is the distance (meter) between Skeeter and Scooter,

Dt is the distance between (meter) Skeeter and the target,

Dmin is the optimal distance (meter) that Scooter should take to approach the target,

Tmin is the optimal time (second) that Scooter should take to approach the target,

Dact is the actual distance (meter) that Scooter took to approach the target,

Tact is the actual time (second) that Scooter took to approach the target

Ds / Dt	2				3				4				5			
	Dmin	Tmin	Dact	Tact	Dmin	Tmin	Dact	Tact	Dmin	Tmin	Dact	Tact	Dmin	Tmin	Dact	Tact
2	1.2	24	1.4	28.2	2.1	42	2.65	45.4	2.75	55	3.28	58.5	4.5	90	7.59	132
3	2	40	4.1	65.5	2.25	45	2.98	49.3	2.6	52	3.31	59.9	3.5	70	5.44	94
4	2.9	58	6.1	95.4	2.5	50	4.32	73.3	2.75	55	3.35	59.8	3.25	65	3.3	69.5
5	3.6	72	9.13	138	3.25	65	5.93	97	3.5	70	5.24	90.1	3.6	72	4.07	75

Table 4 : Data at 45 degrees angle

Ds / Dt	2				3				4				5			
	Dmin	Tmin	Dact	Tact	Dmin	Tmin	Dact	Tact	Dmin	Tmin	Dact	Tact	Dmin	Tmin	Dact	Tact
2	2.8	56	5.18	107	3.6	72	8.19	142	4.5	90	5.15	95.6	5.4	108	6.01	108
3	3.6	72	7.44	126	4.25	85	8.6	150	5	100	8.7	153	5.8	116	6.6	127
4	4.5	90	8.21	138	5	100	9.3	156	5.6	112	10.2	179	6.4	128	7.9	140
5	5.4	108	12.9	209	5.8	116	10.8	185	6.4	128	11.2	194	7	140	12	212

Table 5 : Data at 90 degrees angle

Ds / Dt	2				3				4				5			
	Dmin	Tmin	Dact	Tact	Dmin	Tmin	Dact	Tact	Dmin	Tmin	Dact	Tact	Dmin	Tmin	Dact	Tact
2	3.75	75	7.34	125	4.6	92	8.94	155	5.6	112	9.73	174	6.5	130	12.8	238
3	4.7	94	10.2	177	5.5	110	11.7	198	6.5	130	11.4	202	7.7	154	13.3	232
4	5.5	110	12.1	203	6.6	132	11.3	203	7.5	150	13.9	249	8.5	170	15.7	273
5	6.5	130	17	295	7.5	150	13.7	235	8.5	170	15.4	269	9.5	190	17.3	310

Table 6 : Data at 135 degrees angle

REFFERENCES

[1] L. E. Parker, “Current State of the Art in Distributed Autonomous Mobile Robotics”, In Proceedings of the fifth international symposium on distributed autonomous robotic systems, 2000.

[2] T. Fukuda, S. Nakagawa, Y. Kawauchi, and M. Buss, “Structure Decision for Self Organizing Robot-based of Cellular Structure – CEBOT”, IEEE International Conference on Robotic and Automation, Scottsdale, AZ, pp. 695-700, 1989.

[3] Y. U. Cao, A. S. Fukunaga and A. B. Kahng, “Cooperative Mobile Robotics: Antecedents and Directions”, Proceedings for IEEE/RSJ IROS Conference, Japan, 1995.

[4] D. W. Gage, “How to Communicate with Zillions of Robots”, Proceedings of SPIE Mobile Robots VIII, Boston, Vol. 2058, pp. 250-257, September 1993.

[5] M. J. Mataric, “Designing and Understanding Adaptive Group Behavior”, Volen Center for Complex Systems, Computer Science Department, Brandeis University, Waltham, MA, September 1995.

[6] M. J. Mataric and B. B. Werger, “From Insect to Internet: Situated Control for Networked Robot Teams”, Interaction Laboratory, Robotics Research Labs, Department of Computer Science, University of Southern California, Los Angeles, July 2000.

[7] D. Goldberg, and M. J. Mataric, “Robust Behavior-Based Control for Distributed Multi-Robot Collection Tasks”, Interaction Laboratory, Robotics Research Labs, Department of Computer Science University of Southern California, Los Angeles, July, 2000.

[8] B. MacLennan, “Evolution of Communication in a Population of Simple Machines”, Technical Report CS-90-99, University of Tennessee, Knoxville, TN, January 1990.

[9] H. Yanco, L. Stein, “An Adaptive Communication Protocol for Cooperating Mobile Robot”, in J.-A. Meyer, H. Roitblat & S. Wilson, eds, From Animals to Animats: International Conference on Simulation of Adaptive Behavior, MIT Press, pp. 478--485.

[10] A. Cangelosi, “Evolution of Communication and Language Using Signals, Symbols, and Words”, IEEE Transactions on Evolutionary Computation, Vol. 5, No. 2, April 2001.

[11] G. M. Saunders and J. B. Pollack, “The Evolution of Communication Schemes over Continuous Channels”, Proceedings of the SAB Conference on the Simulation of Adaptive Behavior, 1996.

[12] R. Grabowski, L. E. Navarro-Serment, C. J. J. Paredis and P. K. Khosla, “Heterogeneous Teams of Modular Robots for Mapping and Exploration”, Autonomous Robots, Vol. 8, No. 3, pp. 293-308, 2000.

[13] G. Dudek, M. Jenkin, E. Milios and D. Wilkes, “Experiments in Sensing and Communication for Robot Convoy Navigation”, Proceedings for IEEE/RSJ International Conference Intelligent Robots and Systems Human Robot Interaction and Cooperative Robots, 1995.

[14] J. Wang, “On Sign-board Based Inter-Robot Communication in Distribute Robotic System”, Proceedings for IEEE International Conference on Robotics and Automation, pp. 1045 -1050 vol.2, 1994.

[15] T. Morita, S. Aramaki, S. Kurono and K. Kagekawa, “A Knowledge Representation for the Communication Between Robots”, IEEE International Workshop on Robot and Human Communication, 1993.

[16] R.Schank "Conceptual Dependency: A theory of natural language understanding, " Cognitive Psychology 3, 552—631, 1992.

[17] K. Ohkawa, T. Shibata and K. Tanie, “Method for Generating of Global Cooperation Based on Local Communication”, Proceedings for IEEE/RSJ International conference on Intelligent Robots and Systems, 1998.

[18] C. N. Duarte and B. B. Werger, “Defining a Common Control Language for Multiple Autonomous Vehicle Operation”, Proceedings for IEEE Ocean 2000 Conference, 2000.

[19] M. J. Mataric, R. T. Vaughan, K. Stoy and G. S. Sukhetme, “Whistling in the Dark: Cooperative trail following in uncertain localization space”, Proceedings for 4th International Conference on Autonomous Agents, June 2000

[20] T. Arai, E. Yoshida and J. Ota, “Information Diffusion by Local Communication of Multiple Mobile Robots”, Proceedings for IEEE Conference on Systems, Man and Cybernetics, pp. 535 - 540, 1993.

[21] T. Fujii, H. Asama, T. Fujita, Y. Asakawa, H. Kaetsu, A. Matsumoto and I. Endo, “Knowledge Sharing Among Multiple Autonomous Mobile Robots Through Indirect Communication using Intelligent Data Carriers”, Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems, Vol. 3, pp. 1466 –1471, 1996.

[22] R. T. Pack, et al, “A Software Architecture for Integrated Service Robot Development”, 1997 IEEE Conf. on Systems, Man, and Cybernetics, Orlando, pp.3774-3779, September 1997.

[23] K. Kawamura, et al, “Intelligent Robotic Systems in Service of the Disabled”, IEEE Transactions on Rehabilitation Engineering, Vol.3, pp.14-21, 1995.

[24] K. Kawamura, et al, “Design Philosophy for Service Robots”, Robotics and Autonomous Systems, Vol.18, pp.109-116, 1996.

[25] D. M. Wilkes, et al, “Designing for Human-Robot Symbiosis”, Industrial Robot, Vol.6, No. 1, pp.49-58, 1999.

[26]R. A. Arkin, Behavior-Based Robotics, MIT Press, Cambridge, MA, 1998.

[27] R. A. Brooks, “A Robust Layered Control System for a Mobile Robot”, IEEE of Robotics and Automation, Vol. 2, No. 1, pp.14-23, 1986.

[28] K. Kawamura, R. A. Peters II, C. Johnson, P. Nilas and S. Thongchai, “Supervisory Control of Mobile Robots using Sensory EgoSphere”, 2001 IEEE International Symposium on Computational Intelligence in Robotics and Automation, CIRA 2001, July 29-August 1 2001, Banff, Alberta, Canada. Page 531-537.

[29] ActivMedia Robotics, “Pioneer 2 Mobile Robots -- Operation Manual”, May, 2000.

[30] Real World Interface, Inc. http://www.irobot.com/rwi.

[31] K. Kawamura, D. M. Wilkes, S. Suksakulchai, A. Bijayendrayodhin, K. Kusumalnukool, “Agent-Based Control and Communication of a Robot Convoy”, International Conference on Mechatronics Technology, Singapore, June 2001.

[32] D. Jung, and A. Zelinsky, “Ground Symbolic Communication between Heterogeneous Cooperating Robots”, Autonomous Robots, Vol. 8, pp. 269-292, 2000.

[33] P. Maes, "A Spreading Activation Network for Action Selection", in Intelligent Autonomous Systems, Vol. 2, pp. 875--885, 1989.

[34] D. M. Gaines, “SAN-RL: Combining spreading activation networks and reinforcement learning to learn configurable behaviors”, SPIE Conference, Boston, MA, October, 2001.

[35] K. Kawamura, R. A. Peters II, D. M. Wilkes, A. B. Koku and A. Sekmen, “Toward Perception-Based Navigation Using EgoSphere”, Proceedings for SPIE Conference, Boston, MA, October, 2001.