To the Graduate Council:

I am submitting herewith a thesis written by Maureen Chandra entitled "Software Reconfigurability for Heterogeneous Robot Cooperation." I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.

Lynne Parker__
Major Professor

We have read this thesis and
recommend its acceptance:

_Robert Ward_____

_Bradley Vanderzanden_____

Acceptance for the Council:

____Anne Mayhew____
Vice Chancellor and Dean of
Graduate Studies

(Original signatures are on file with official student records.)

SOFTWARE RECONFIGURABILITY FOR HETEROGENEOUS ROBOT COOPERATION

A Thesis

Presented for the

Master of Science Degree

The University of Tennessee, Knoxville

Maureen Chandra

May 2004

Copyright © 2004 by Maureen Chandra
All rights reserved.

DEDICATION

This thesis is dedicated to my parents, Jumin Chandra and Jetty Chandra, my aunt and uncle, Julita Chandra and Duc Nguyen, my significant other and best friend, Muliawan Sjarif, and the rest of the family, for your love, inspiration, support and encouragement.

ACKNOWLEDGEMENTS

I wish to thank all those who helped me complete my Master of Science degree in Computer Science. I would like to thank Dr. Parker for her valuable guidance, ideas, advice and encouragement through the research and thesis writing process. I would like to thank Michael Bailey, Ben Birch, Balajee Kannan, Fang Tang, and Chris Reardon for their help and advice in the implementation process. I would also like to thank Dr. Vander Zanden and Dr. Ward for serving on my committee. Finally, I would like to thank my family and friends for their suggestions and encouragement, which have made this work possible.

ABSTRACT

Previous work in multi-robot cooperation has aimed at gaining autonomy and fault tolerance in the robot team. Most attempt to accomplish this by dynamically assigning roles or tasks to the robots and pre-designing the solution for heterogeneous robot teams with known sensing capabilities. However, pre-designed solutions fail when changes occur in the robot team composition or in the available environmental sensors at run-time. Automated solution design is thus needed to accomplish autonomy and fault tolerance in multi-agent systems. Very little work has been done in automating robot solutions at run-time due to the difficulty of adapting to many unexpected events such as sensor failures and changes in the environment. Therefore, the software reconfigurability approach is introduced. This approach takes into account all available sensors and capabilities on the robot team rather than on individual robots. It views those sensors and capabilities as building blocks and configures a solution by choosing from different ways of combining these blocks. Since the implementation of this approach cannot be completed in a short time, this thesis concentrates on 1) modularizing the robot behaviors to create these building blocks and 2) illustrating different ways to solve a task based on the same blocks. Our approach is based on creating distributed "schema" building blocks across multiple robots and providing a mechanism for the schemas to be reconnected in multiple ways to achieve different solution strategies. We illustrate the fundamental capabilities of this approach through a simple multi-robot application involving robots moving to assigned goal positions. We show how different schema configurations can achieve this task in multiple ways by using different combinations of robot behaviors. This accomplishment enables the development of a fault tolerant and autonomous robot system by providing different solutions to solving a task based on the available sensors and capabilities in the robot team, as well as constructs that allow the team solution to be automated at run-time.

TABLE OF CONTENTS

Chapter Page

1 INTRODUCTION 1

1.1 Motivation and Contribution 1
1.2 Software Reconfigurability 2
1.3 Outline 3

2 PROBLEM DESCRIPTION 5

2.1 The Mission 5
2.2 Types of Robots and Environmental Sensors (ES) used 6
2.3 Overview of Approach 6

3 APPROACH 9

3.1 Schemas 9
3.1.1 Perceptual Schemas (PS) 9
3.1.2 Communication Schemas (CS) 10
3.1.3 Motor Schemas (MS) 10
3.2 Architecture 11

4 IMPLEMENTATION 15

4.1 Types of Robot and Environment 16
4.2 Proof-of-concept Case Study 20
4.2.1 Case 1: Fully Capable Robots (Baseline) 20
4.2.2 Case 2: Communicating Own Current Position 25
4.2.3 Case 3: Communicating Other's Current Position 25
4.3 Classes Defined 26
4.3.1 Robot Class 27
4.3.2 ES (Environmental Sensor) Class 28
4.3.3 PS (Perceptual Schema) Class 28
4.3.4 CS (Communication Schema) Class 28
4.3.5 MS (Motor Schema) Class 31
4.4 Functions Defined 31
4.4.1 Main Function 32
4.4.2 Configure and Reconfigure Functions 32
4.4.3 Initialize_Connections Function 34
4.5 Limitations 34

5 RESULTS AND DISCUSSION 35

5.1 Data 36
5.1.1 Case 1: Fully Capable Robots (Baseline) 36

5.1.2 Case 2: Communicating Own Current Position 39
5.1.3 Case 3: Communicating Other's Current Position 41
5.2 ANALYSIS 41

6 RELATED WORK 48

6.1 TASK MANIPULATION 49
6.2 FAULT TOLERANCE 50
6.3 AUTONOMY IN HETEROGENEOUS ROBOT TEAMS 51
6.4 INFORMATION INVARIANTS 53

7 FUTURE WORK AND CONCLUSIONS 55

7.1 FUTURE WORK 55
7.2 CONCLUSIONS 56

LIST OF REFERENCES 58

APPENDICES 60

VITA 67

LIST OF TABLES

Table Page

Table 1. Environmental Sensor (ES) 7
Table 2. Possible Agents 7
Table 3. Perceptual Schema (PS) 12
Table 4. Communication Schema (CS) 12
Table 5: Motor Schema (MS) 12
Table 6. Approximation of the sensing cost for all sensors used on this implementation. 36
Table 7. Data collected from Case 1 experimentation where two robots move towards their unique, assigned goal without any communication. The results shown are the average and the standard deviation of data taken from 10 experiments. 38
Table 8. Data collected from Case 2 experimentation where one of the robots does not have enough information to localize and uses its camera to detect the other robot. The results shown are the average and the standard deviation of data taken from 10 experiments. 41
Table 9. Data collected from Case 3 experimentation where one of the robots does not use any of its sensors and waits for other robot to detect it. The results shown are the average and the standard deviation of data taken from 10 experiments. 43

LIST OF FIGURES

Figure Page

Figure 1. Simple architecture showing connections between ES, PS, CS and MS within $Agent_i$. This architecture shows communication from $Agent_j$ to $Agent_i$ and from $Agent_i$ to $Agent_k$. 13
Figure 2. Complex architecture showing connections between all of the available schemas. 14
Figure 3. Available schemas (building blocks) on AG1 and AG2. AG1 has only a laser scanner (ES), whereas AG2 has only a camera (ES2). 17
Figure 4. Available schemas (building blocks) on AG3 and AG4. Agent AG3 has only DGPS (ES3), whereas AG4 has both a laser scanner (ES1) and a camera (ES2). 17
Figure 5. Available schemas (building blocks) on AG5 and AG6. Agent AG5 has both DGPS (ES3) and a laser scanner (ES1), whereas Agent AG6 has both DGPS (ES3) and a camera (ES2). 18
Figure 6. Available schemas (building blocks) in AG7 and AG8. AG7 has all three sensors – DGPS (ES3), a laser scanner (ES1) and a camera (ES2), whereas AG8 has no available sensors 18
Figure 7. Picture of two Pioneer robots facing their goal points, which are indicated by black squares on the floor, in a hallway. 19
Figure 8. Map of the same hallway from Player-Stage simulation showing the two robots and their goal points. 21
Figure 9. Case 1: Fully Capable Robots (Baseline). White boxes indicate activated schemas. The schemas shaded in gray are not activated. 22
Figure 10. Case 2: Communicating Own Current Position. White boxes indicate activated schemas. The schemas shaded in gray are not activated. 23
Figure 11. Case 3: Communicating Other's Current Position. White boxes indicate activated schemas. The schemas shaded in gray are not activated. 24
Figure 12. *Robot* class defined for all agents. It includes all schemas although not all of the sensors are physically available. The relevant schemas can be activated or deactivated based on the needs of the team. 27
Figure 13. *ES* (Environmental Sensor) class defined for all environmental sensors. The subclasses of *ES* are *ES1_laser* for laser scanner, *ES2_camera* for camera and *ES3_dgps* for DGPS. 29
Figure 14. *CS* (Communication Schema) class defined for all communication schemas. The subclasses of *CS* are *CS1_self_pos* and *CS2_other_pos*. 30
Figure 15. *MS* (Motor Schema) class defined for all motor schemas. Currently there is only one motor schema, so there are no subclasses defined for this class. 31
Figure 16. The *main* function definition. 33

Figure 17. Global array variables that store information about which schemas are activated. 33
Figure 18. These snapshots show two fully capable robots moving towards their goal positions with their own laser scanner and with no communication between them. Here, the robot on the right is called Red, while the robot on the left is called Blue (read left to right, top to bottom). 37
Figure 19. These snapshots show the Needy robot (on the left) looking for the Helper robot (on the right) to calculate its position relative to the Helper. With the relative position combined with Helper's position communicated, Needy can calculate its current position and move towards the goal (read left to right, top to bottom). 40
Figure 20. These snapshots show the Helper robot (on the right) assisting the Needy robot (on the left) by attempting to detect and calculate Needy's position relative to itself. In this case, Needy is not using any of its sensors to go towards the goal. Needy waits for Helper to communicate its current position (read left to right, top to bottom)..... 42
Figure 21. Time cost for robots in Case 1, Case 2, and Case 3 experiments. 43
Figure 22. Communication cost for robots in Case 1, Case 2, and Case 3 experiments. 44
Figure 23. Sensing cost for the robots in Case 1, Case 2, and Case 3 experiments. 45
Figure 24. Success rate for the robots in Case 1, Case 2, and Case 3 experiments. 46
Figure 25. Success rate for Case 1, 2, and 3 experiments compared with that of the case robots do not collaborate in Case 2 and 3. In this case, since Needy is not capable of moving to the goal by itself, only one of two robots in the team can make it to its goal. Thus, the average success rate is 0.5 without cooperation. 47

Figure A - 1. PS (Perceptual Schema) class defined for all agents..... 61
Figure A - 2. Function definition for configure defined in Robot class. 62
Figure A - 3. Function definition for initialize_connections defined in Robot class..... 65

1 INTRODUCTION

1.1 Motivation and Contribution

Autonomy and fault tolerance are considered to be important issues in heterogeneous robot team coordination. Some of the previous works attempt to accomplish these characteristics by dynamically assigning roles or tasks to individual robots. If a robot is not able to finish its task or act its role, the task or the role is given to another more capable robot. However, most work pre-designs the robot team and the solution for the heterogeneous robots. Pre-designing solutions does not work when the system encounters sensor failures or environmental changes, because there is only one solution designed to work on a particular robot team composition. If one of the robots in a team experiences a sensor failure, the pre-designed solution no longer can accomplish the task.

For this reason, team solution strategies need to be automated during run-time. This is a difficult problem not only because many different robot team compositions and solutions can be designed based on the same robots and the same task, but also because the solution must adapt to unexpected events such as sensor failures and changes in the environment. To address this issue, a software reconfigurability approach in heterogeneous robot cooperation is introduced. By building distributed "schema" building blocks across multiple robots and presenting a system for the schemas to be reconfigured in multiple ways, our approach can develop different solutions in accomplishing a task. To illustrate the fundamental capabilities of this approach, we experiment on a simple multi-robot application involving robots moving to assigned goal positions. Using different combinations of robot behaviors, we show how different schema configurations can accomplish this task in multiple ways. This approach accomplishes a task with dynamic cooperation and automated solution design. It

develops a framework for an autonomous and fault tolerant multi-agent system because it allows for more than one solution to accomplish a task and for cooperative team solutions to be automated during run-time.

1.2 Software Reconfigurability

In heterogeneous robot teams, the robots usually have different types of sensors. Some of them might have a laser scanner, an omni-directional camera, or Differential Global Positioning (DGPS). Some might have combinations of these sensors or none at all. Since the sensing capabilities are known from the start, developers typically pre-design the software for the heterogeneous robot teams to accomplish a task. As discussed in Section 1.1, pre-designing the software does not usually achieve an autonomous and fault tolerant system. With software reconfigurability, developers do not need to pre-design the software solution for the heterogeneous robot teams to accomplish a task. Instead, the cooperative team solution is automated during run-time.

Software reconfigurability is defined as the ability of the software (solution) to be reconfigured when needed to successfully accomplish the team task. In this approach, the cooperative solution to accomplish the task based on the available robots is configured at run-time. If there are any changes in the robot team composition or in the environmental sensors available, the changes to the solution are also performed at run-time, which makes the system much more robust.

This approach is also fault tolerant because it can choose from different ways for the robots to assist each other in solving the task by taking into consideration all available sensors and computational capabilities in the robot team rather than on individual robots. Those sensors and capabilities are observed as fundamental building blocks; selecting from different combinations of these blocks configures a solution. If one of the robots in the team experiences a sensor failure or a change in the environment that disables one

of its sensors, this approach reconfigures a new solution by re-examining the remaining available sensors and capabilities in the robot team.

1.3 Outline

Section 2 describes the mission, including the types of robots and environmental sensors used. It then gives an overview of the approach and how it uses the software reconfigurability concept. The approach is described in detail in Section 3. This chapter explains the designed schemas and how they are used. Afterwards, the architecture of all available schemas is given. This section explains how the schemas are connected to each other to enable multiple potential solution strategies.

Section 4 describes the types of robots and environment used for the simulation and for the physical robot experiments. It then explains the case study, which is divided into three parts: Case 1 (Fully capable robots), Case 2 (Communicating own current position), and Case 3 (Communicating other's current position). Since the implementation is in C++, this chapter also shows and explains the classes defined followed by the *main*, *configure*, *reconfigure* and *initialize_connection* functions. The limitations of the implementation are given at the end.

Section 5 presents the results of this approach, providing the data gathered from multiple runs on physical robots. Simulation results are not presented because the vision system using the camera and marker cannot be implemented in the Player-Stage simulation. Analysis of the data and a discussion of the results are provided. In Section 6, some of the related work in multi-robot cooperation is discussed. This work is divided into four sections: Task Manipulation, Fault Tolerance, Autonomy in Heterogeneous Robot Teams, and Information Invariants. This chapter summarizes the related research and then briefly explains how they relate to this thesis. In Section 7, conclusions are

drawn based on analysis of results, as well as possible ways to further develop this idea in the future.

2 PROBLEM DESCRIPTION

The objective of this research is to show that it is possible to have software reconfigurability in heterogeneous multi-robot teams and that this approach can enable the robot team to accomplish the mission in an autonomous and fault tolerant way. This chapter first describes the illustrative mission, including the type of robots and environmental sensors used, followed by an overview of the approach and how it uses the software reconfigurability concept.

2.1 The Mission

To show that the software reconfigurability concept works, we have selected a very simple proof-of-concept mission. The goal of this mission is for each robot to reach its pre-assigned, unique goal point. In order for a robot to reach its assigned goal, it needs to know its current position relative to its goal position so that it can move in the proper direction. This implies that each robot has to be aware of two positions - its current position and its assigned goal position. In some cases, a robot may be able to sense its current position using its own sensors. In other cases, the robot may not have enough information to determine its current position. Other more capable robots can help by communicating the needed information to this less capable robot.

This mission is to be tested both in simulation and on physical robots. This proof-of-concept assumes that the robots move in an obstacle-free environment, except for the walls and the robots themselves. If a robot has the required sensors needed to avoid obstacles, it uses those sensors to avoid the walls and other robots. Cooperative obstacle avoidance for other robots is left as a future extension of this research.

2.2 Types of Robots and Environmental Sensors (ES) used

The environmental sensors (ES) used in this case study are shown in Table 1. A robot can use a laser scanner with an environmental map (ES1) to localize itself and calculate its current position. In addition, this sensor can also be used to avoid obstacles and detect the position of another robot relative to itself. In that case, it is used without the environmental map. The camera (ES2), on the other hand, cannot be used for localization and calculating current position. It can, however, be used to detect the position of another robot relative to itself. The last sensor, DGPS (ES3), can be used for localization and to detect the robot's current position. Moreover, we define different types of robots (agents) according to the environmental sensors available to that agent, as shown in Table 2. The goal of the experiments in this research is to illustrate how different combinations of these agents can be used to accomplish the task in different ways, using the same underlying building blocks.

2.3 Overview of Approach

Our approach to the problem is based on Arkin's motor schema approach (Arkin, 1989), which uses perceptual schemas (PS) that process the input sensed from the environment, and motor schemas (MS) that process the motor control to move the robot. In addition, we add a new construct called communication schemas (CS), which communicate data as an input to another robot's perceptual schemas. We treat these schemas as building blocks that contain information needed for the robot team to accomplish the task. Instead of having the solution (software) designed before runtime, the robots must determine how to accomplish their tasks by evaluating the schemas (building blocks) available to the team and attempting to connect these schemas to obtain all the information needed for them to accomplish the task. Different ways of connecting the schemas are possible and our goal is to enable the robots to choose the most efficient and least costly configuration. This approach incorporates the use of

Table 1. Environmental Sensor (ES)

Name	Description
ES1	Laser Scanner
ES2	Camera
ES3	DGPS

Table 2. Possible Agents

Agent	Sensors
AG1	ES1
AG2	ES2
AG3	ES3
AG4	ES1 and ES2
AG5	ES1 and ES3
AG6	ES2 and ES3
AG7	ES1, ES2 and ES3
AG8	No ES

software reconfigurability and shows that the solution does not need to be pre-designed in advance.

3 APPROACH

Our software reconfigurability approach is based on the use of perceptual schemas (PS), motor schemas (MS) and communication schemas (CS). This chapter describes the schemas and the architecture used in this approach. The architecture shows different possible connections between the schemas (building blocks) and how we can use these to obtain different solutions to accomplish a task in a robot team.

3.1 Schemas

3.1.1 Perceptual Schemas (PS)

Perceptual schemas process the input sensed from the environment. They gather raw data from the environmental sensors and calculate the needed information from the data. We define five perceptual schemas in our approach. PS1 calculates the robot's current position. With the sensors we have defined, this position can be calculated either by using input data from a laser scanner combined with an environmental map (ES1) or from DGPS (ES3). In addition, PS1 can also receive input of this information from other perceptual schemas or communication schemas. For example, having an input of the position of another robot relative to itself combined with an input of the current position of that robot, a robot can calculate its own current global position. PS1 can also receive input from another more capable robot, which calculates and communicates its current position so that this less capable robot does not need to calculate the position itself.

PS2 outputs a robot's goal position and currently does not receive any input from any other schemas. For our case study, the goal position for each robot is hard-coded inside the schema. PS3 calculates the current position of another robot (agent) based on

two inputs - the position of another robot relative to itself and its own current position. PS4 calculates the position of another agent relative to itself. Currently, this calculation is based on input coming from either a laser scanner (ES1) or a camera (ES2). PS5 receives input from another agent's communication schema, CS2, which communicates the current position of that agent.

3.1.2 Communication Schemas (CS)

Communication schemas communicate data as an input to another robot's perceptual schemas. As mentioned in Section 3.1.1, PS1 can receive input from another robot's communication schema, CS2, which communicates the current position of that other agent to the current agent. CS1 communicates its current position as an input to another agent's PS5. As shown in Section 4, we use point-to-point communication, which means a robot communicates directly to another robot, rather than using broadcast communication.

3.1.3 Motor Schemas (MS)

Motor schemas send control signals to the robot's effectors, such as its wheels. Based on the robot's current position received from PS1 and its goal position received from PS2, it calculates the motor command that moves the robot towards the goal position. In cases where a robot does not have enough information to calculate its current position and needs help from another robot, they both need to cooperate with each other in keeping a line of sight for the robot with a camera or a laser scanner. The motor schema gives motor commands for the robot with a camera or a laser scanner to keep looking for the other robot until detected, so that it can then calculate the position of that robot relative to itself. The other robot is given motor commands to stop moving until it is detected. In our proof-of-concept application, we only define one motor schema. This

can be generalized to multiple motor schemas that collectively control a set of robot actuators.

3.2 Architecture

As shown in Table 3, Table 4 and Table 5, there are five perceptual schemas, two communication schemas and one motor schema. Since not all robots have the sensor(s) required for obstacle avoidance, we assume there is only one motor schema in all robots, which is the go to goal schema. The connection between ES, PS, CS and MS inside an agent is shown in Figure 1. The environmental sensors give input to the perceptual schemas for them to calculate their output. The output from perceptual schemas goes to either the motor schema, for it to calculate the motor control, or to communication schemas, which then provide input to another robot's perceptual schemas.

A more detailed architecture is shown in Figure 2. This figure shows all the available schemas inside an agent and how they can be connected to each other. The solid-line arrows going into a schema mean that it is sufficient for the schema to only have that specified input to produce output, whereas the dashed-line arrows mean that the schema needs all of the indicated inputs for it to calculate output. For example, PS4 can calculate output based on an input from either ES2 or ES1. However, MS can only produce output if it receives two inputs, one from PS1 and another from PS2. PS1 can produce output with input(s) from either ES1 (combined with Map, which is constantly available as input to PS1), ES3, $CS2_j$ (other agents' CS2), or (PS4 and PS5). PS5 needs an input from $CS1_j$ (other agents' CS1). CS1 and CS2 can send output to other agents' PS1 and PS5.

Table 3. Perceptual Schema (PS)

Name	Description
PS1	Calculates own current position
PS2	Calculates own goal position (temporarily hard-coded)
PS3	Calculates current position of another agent
PS4	Calculates position of another agent relative to itself
PS5	Receives current position of another agent communicated by that agent

Table 4. Communication Schema (CS)

Name	Description
CS1	Communicates own current position to another agent
CS2	Communicates current position of another agent to that agent

Table 5: Motor Schema (MS)

Name	Description
MS	Calculates motor command that leads the agent towards its goal

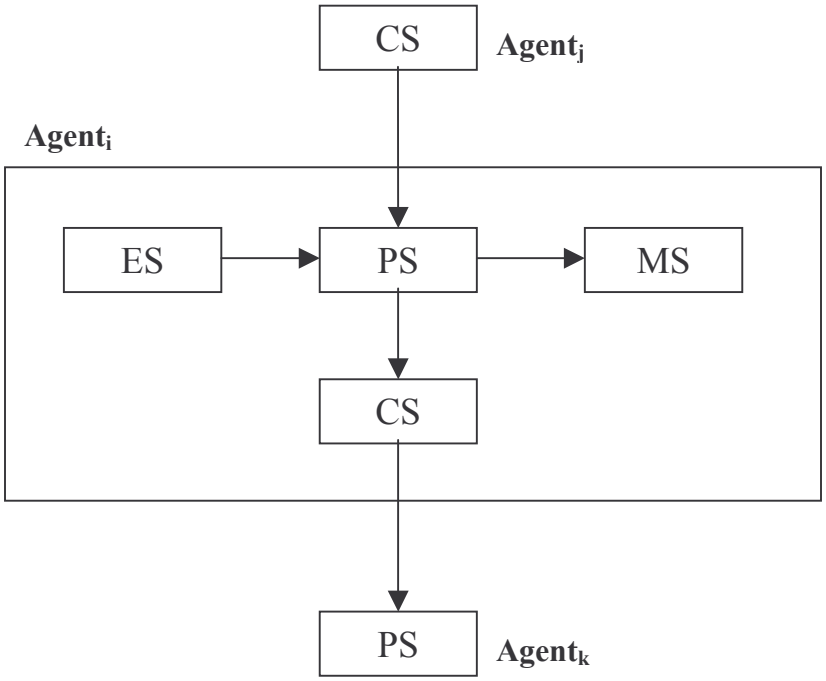
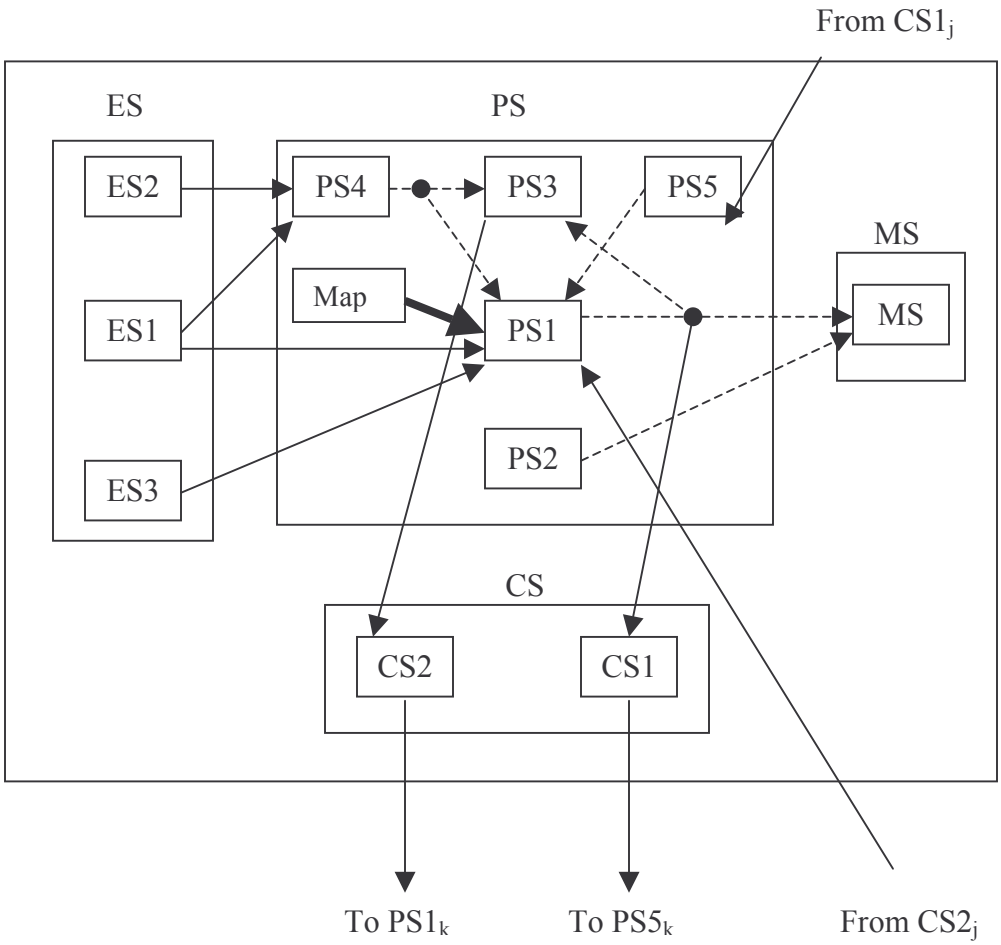


Figure 1. Simple architecture showing connections between ES, PS, CS and MS within Agent$_i$. This architecture shows communication from Agent$_j$ to Agent$_i$ and from Agent$_i$ to Agent$_k$.



--- = All of the dashed lined arrows that feed into the same box need to be activated at the same time.
Note: Thick arrow from Map to PS1 indicates that Map is constantly available as input to PS1.

Figure 2. Complex architecture showing connections between all of the available schemas.

4 IMPLEMENTATION

The code for the implementation of our approach is written in C++ and is used on the robots available in the Distributed Intelligence Laboratory (DI Lab) at The University of Tennessee and on the Player-Stage simulation (Gerkey, 2001). This implementation makes use of prior work by Parker, Kannan, Tang and Bailey in (Parker *et al.,* 2001). It uses the vision system developed in that work for robot detection using a Sony pan-tilt-zoom camera and a cylindrical marker designed to provide unique visual ID, relative position and orientation information. Moreover, our implementation also re-uses parts of the code in that prior research to calculate the current position and motor commands for the robots. However, we have re-packaged this previous software by reorganizing it and modularizing it into schemas.

This implementation aims to show the feasibility of the approach of solving the simple task by modularizing the available schemas (building blocks) on all robots and reconfiguring them in multiple ways to accomplish the same task. Currently, the alternative connections are hard-coded and several solutions that solve the same task using same group of robots are implemented. By connecting the schemas in different ways between the robots, we can show that there is more than one solution for accomplishing the mission. When a problem occurs such that a solution cannot be used to finish the task, alternative solutions can be configured.

Future work is aimed at automating the decision process for choosing the most efficient solution through varying the schema connectivity. If we were successful in showing that the robots can accomplish a task by reconnecting the available schemas in the most efficient way, we would also show that software reconfigurability works. With a team of any combination of robots, we could then reconfigure the software (programmer solution) to accomplish a task. If one of the sensors in a robot fails, then

some schemas would just be unavailable and the solution would be reconfigured again with the currently available schemas.

4.1 Types of Robot and Environment

As shown in Table 2, there are eight combinations of possible robots (agents) based on the available environmental sensors used, which are given in Table 1. The available schemas for each of these robots are shown in Figure 3, Figure 4, Figure 5 and Figure 6. Not all of these available schemas are activated at all times. Depending on the solution (connections) chosen, only certain schemas are activated. Since DGPS is not available in the DI Lab, our experiments are focused on robots with two environmental sensors - a laser scanner and a camera.

The Pioneer robots used in this implementation are equipped with a SICK laser range scanner and a Sony pan-tilt-zoom camera, both of which are facing forward. They also have a wireless ad hoc networking capability that enables them to communicate with each other. A Pioneer robot with both laser scanner and camera is called AG4. If the laser scanner fails in that robot, it is only left with a camera and thus is called AG2. Likewise, if the camera in that robot fails, it has only a laser scanner, which makes it an AG1. If both of the sensors fail, the robot is called AG8. A Pioneer robot with all three sensors - DGPS, a laser scanner and a camera is called AG7. If the laser scanner fails, the robot is left with only DGPS and a camera, which makes it an AG6. If the robot has both DGPS and a laser scanner, it is called AG5. With only DGPS, the robot is called AG3.

The experiments are conducted in a known indoor environment using a map generated using a laser range mapping algorithm. Figure 7 shows a picture of the robots and their goal points in a corridor, which is used for these experiments. The goal points are indicated by black squares on the floor. In the Player-Stage simulation, the same

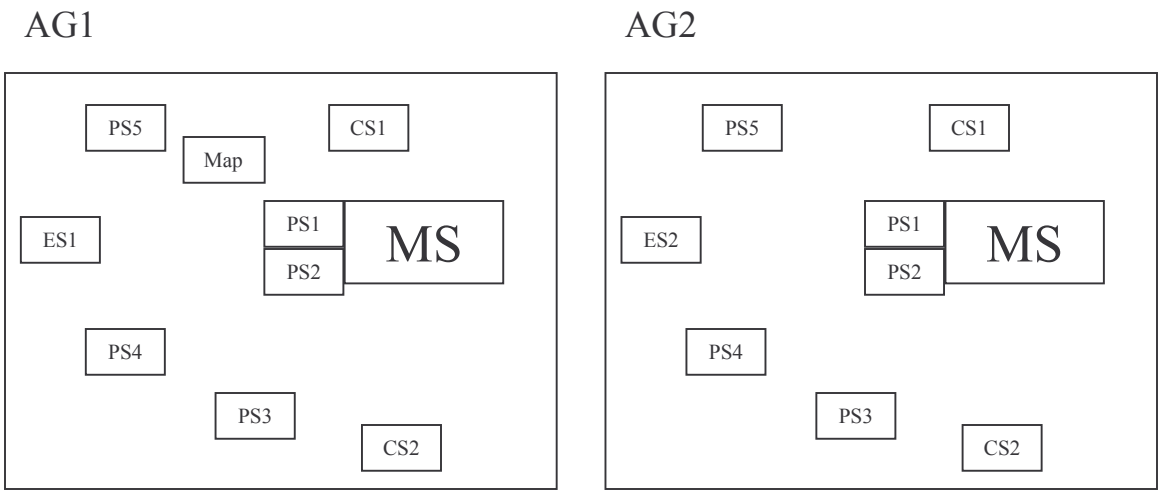


Figure 3. Available schemas (building blocks) on AG1 and AG2. AG1 has only a laser scanner (ES), whereas AG2 has only a camera (ES2).

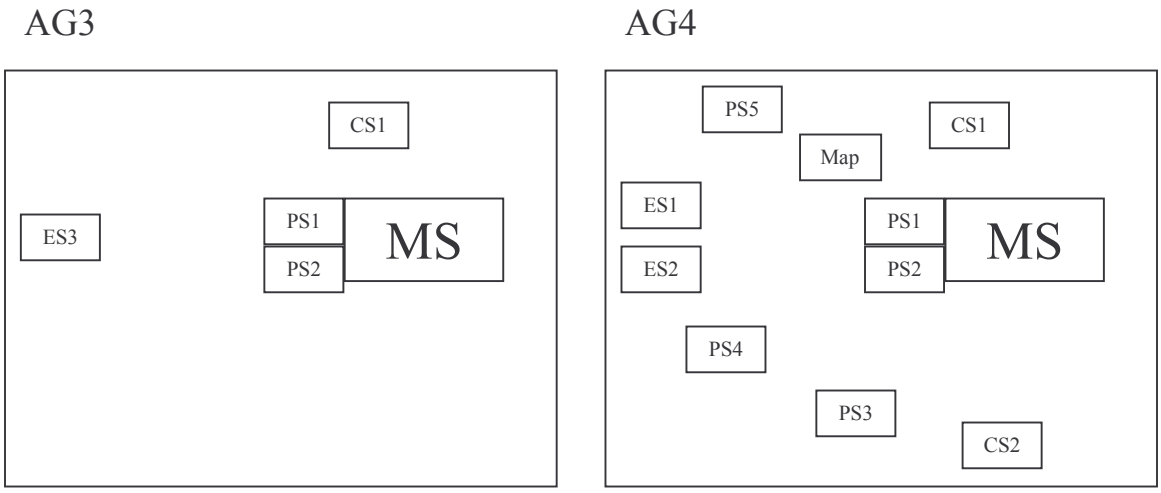


Figure 4. Available schemas (building blocks) on AG3 and AG4. Agent AG3 has only DGPS (ES3), whereas AG4 has both a laser scanner (ES1) and a camera (ES2).

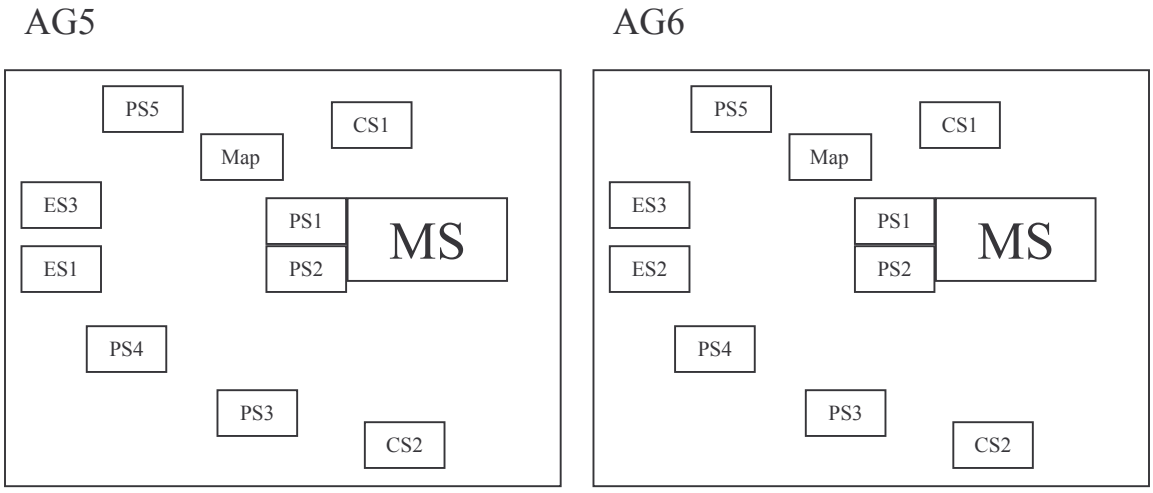


Figure 5. Available schemas (building blocks) on AG5 and AG6. Agent AG5 has both DGPS (ES3) and a laser scanner (ES1), whereas Agent AG6 has both DGPS (ES3) and a camera (ES2).

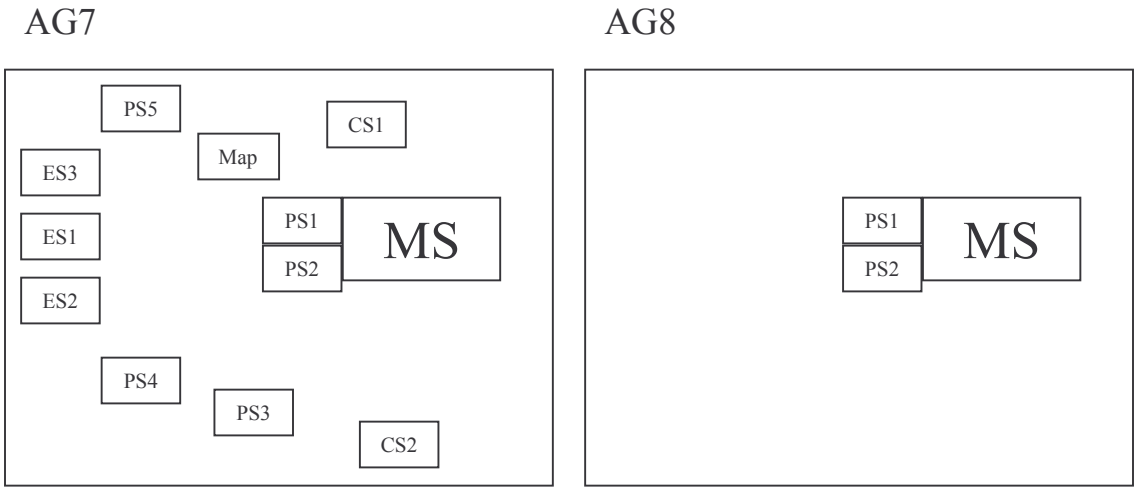


Figure 6. Available schemas (building blocks) in AG7 and AG8. AG7 has all three sensors – DGPS (ES3), a laser scanner (ES1) and a camera (ES2), whereas AG8 has no available sensors



Figure 7. Picture of two Pioneer robots facing their goal points, which are indicated by black squares on the floor, in a hallway.

environment is used. The zoomed-in map of the environment – used both on the physical robots and in simulation is given in Figure 8. This map is of the same area shown in Figure 7.

4.2 Proof-of-concept Case Study

The case study uses two robots, which are both Pioneers. Using these two robots, three cases are implemented and compared to each other. In all cases, the schemas are only activated if there are input(s) coming into them and/or output(s) going out of them. If they do not have any connections, it means that they are not activated and are not used. The schemas that are not activated are shaded in gray in Figure 9, Figure 10, and Figure 11.

As an identifier, each robot is equipped with a marker, as shown in Figure 7. The vision system used for robot detection is described in (Parker *et al.*, 2004). It uses a Sony pan-tilt-zoom camera and a cylindrical marker designed to provide unique visual ID, relative position, and orientation information for the robots.

4.2.1 Case 1: Fully Capable Robots (Baseline)

As a baseline, Case 1 uses two fully capable robots. These robots both have a laser scanner and a camera and thus are called AG4. Each robot localizes itself using its laser scanner and map and reaches its own unique goals independently. This case is the baseline case since it is the most ideal solution but only works if the robots both have their own laser scanners and maps. If one of the robots loses its laser scanner, this solution ceases to work.

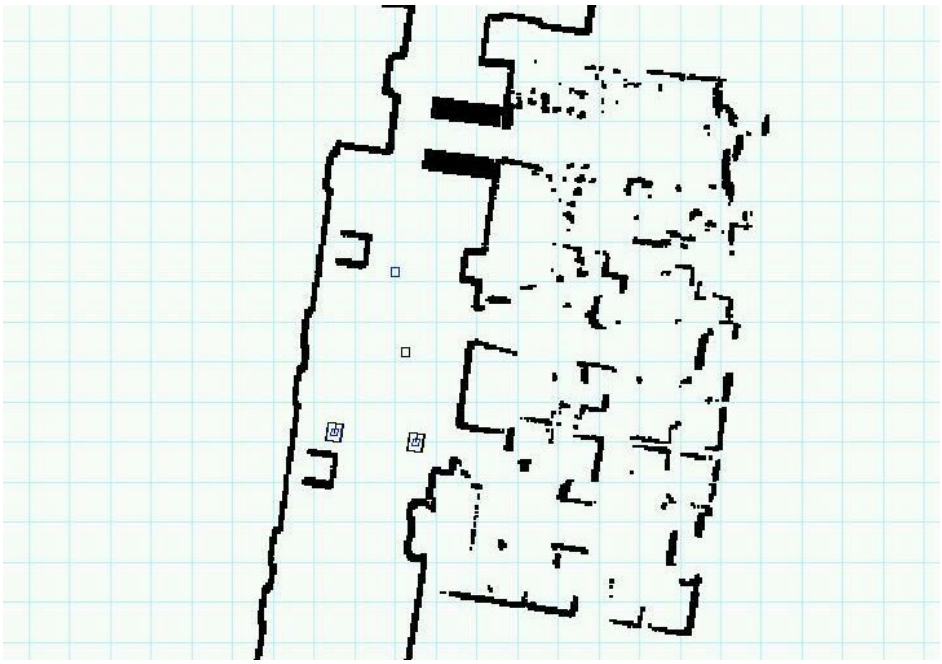
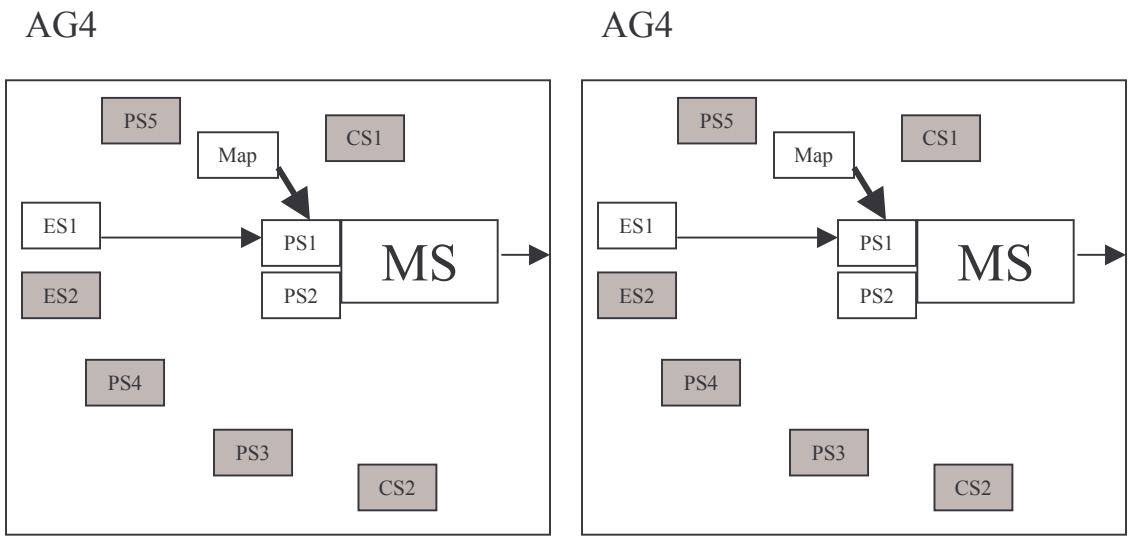
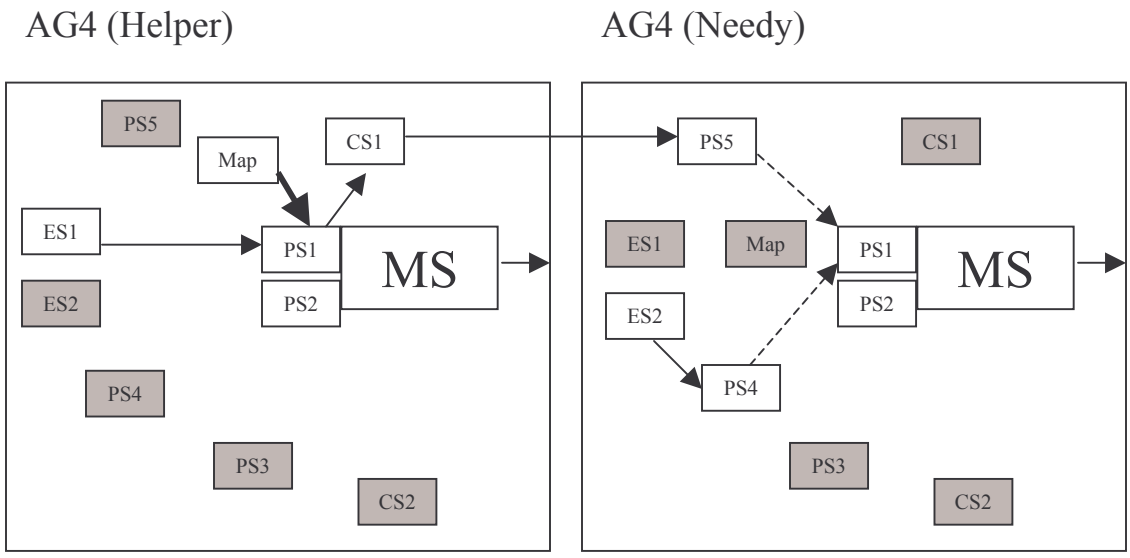


Figure 8. Map of the same hallway from Player-Stage simulation showing the two robots and their goal points.



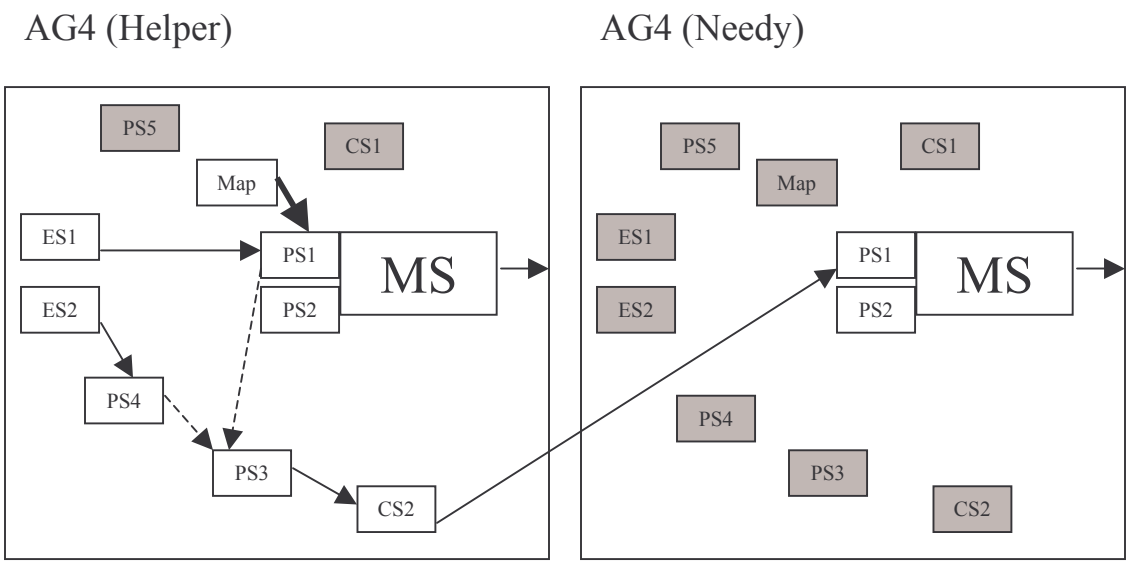
--- = All of the dashed lined arrows that feed into the same box need to be activated at the same time.
Note: Thick arrow from Map to PS1 indicates that Map is constantly available as input to PS1.

Figure 9. Case 1: Fully Capable Robots (Baseline). White boxes indicate activated schemas. The schemas shaded in gray are not activated.



--- = All of the dashed lined arrows that feed into the same box need to be activated at the same time.
Note: Thick arrow from Map to PS1 indicates that Map is constantly available as input to PS1.

Figure 10. Case 2: Communicating Own Current Position. White boxes indicate activated schemas. The schemas shaded in gray are not activated.



--- = All of the dashed lined arrows that feed into the same box need to be activated at the same time.
Note: Thick arrow from Map to PS1 indicates that Map is constantly available as input to PS1.

Figure 11. Case 3: Communicating Other's Current Position. White boxes indicate activated schemas. The schemas shaded in gray are not activated.

Figure 9 shows the connections between the schemas in Case 1. PS1 and PS2 are attached to MS to show that they are part of a behavior, which is a Go to goal behavior. The only activated schemas in both agents are ES1, PS1, PS2 and MS. The others are not connected to any other schemas and thus are not activated.

4.2.2 Case 2: Communicating Own Current Position

The second case involves a robot whose laser scanner is not available, but still has use of its camera. This robot is called the Needy, while the fully capable robot is called Helper. Helper helps Needy by communicating its own current position. Needy then uses its camera to detect the Helper's position relative to itself and calculate its own current position based on that relative position and the Helper's communicated current position.

In Figure 10, Helper uses its laser scanner (ES1) and map to calculate its own current position in PS1. Then, it communicates its current position to Needy using CS1. Needy receives the Helper's current position through PS5 and passes it to PS1. Needy also uses its camera (ES2) to calculate the relative position of Helper in PS4. The relative position is then passed to PS1. Together with the Helper's current position passed by PS5, it is then able to calculate its own current position. Now that both Helper and Needy know their own current positions and goal positions, their motor schemas can calculate their motor control and lead them to their goal points.

4.2.3 Case 3: Communicating Other's Current Position

In Case 3, Needy has access to neither its laser scanner nor camera. Helper, which is still fully capable, helps Needy by communicating Needy's current position. Helper uses its camera to detect Needy's position relative to itself and then based on that relative

position and its own current position, the Helper calculates and communicates Needy's current position.

In Figure 11, Helper uses its laser and map to calculate its own current global position in PS1. It also uses its camera (ES2) to calculate the relative position of Needy in PS4. With its own current position input from PS1 and the relative position of Needy from PS4, Needy's current global position can be calculated in PS3. Then, Needy's current position is communicated using CS2 by the Helper to Needy, which receives its own current global position from the Helper directly in PS1. Since both of the robots know their own current and goal positions, MS can calculate their motor controls and lead them to their goal points.

4.3 Classes Defined

Since we are using C++ to implement this approach, the schemas are defined as classes. They are also connected to each other using input and output variables. Each of these schemas or subclasses has a *compute_output* function, which calculates output based on input received from environmental sensors or other schemas. For that purpose, each subclass has input and output variables. After an activated schema or subclass receives an input into its input variable(s) from either environmental sensors or other schemas, *compute_output* function calculates the output based on that input. Afterwards, the output is stored in the output variable(s). Connecting two subclasses (schemas) together is simply assigning one's output variable to another's input variable.

4.3.1 Robot Class

The *Robot* class is defined for all possible agents (robots). As shown in Figure 12, it includes all defined schemas although not all of the sensors are physically available. For example, when a robot loses its laser scanner, its *ES1_laser* and other related schemas are not available anymore. The schemas that are available can either be activated or deactivated. The activated schemas would be connected to each other and the remaining schemas are deactivated since they are not connected to any other schemas.

The *Robot* class has three defined functions: *configure*, *reconfigure*, and *initialize_connections*. The *configure* function decides which schemas should be activated and the *reconfigure* function updates the most recent available schemas and re-evaluates the entire system to decide which schemas are activated. The *initialize_connections* function initializes the connections between the activated schemas.

```
class Robot {
  ES1_laser                es1_laser;
  ES2_camera               es2_camera;
  ES3_dgps                 es3_dgps;
  PS1_global               ps1_global;
  PS2_goal                 ps2_goal;
  PS3_global_other         ps3_global_other;
  PS4_rel_other            ps4_rel_other;
  PS5_global_comm          ps5_global_comm;
  CS1_self_pos             cs1_self_pos;
  CS2_other_pos            cs2_other_pos;
  MS                       ms;
public:
  virtual void configure(int argc, char** argv);
  virtual void reconfigure(){}
  virtual void initialize_connections();
};
```

Figure 12. *Robot* class defined for all agents. It includes all schemas although not all of the sensors are physically available. The relevant schemas can be activated or deactivated based on the needs of the team.

4.3.2 ES (Environmental Sensor) Class

The *ES* class is defined for all environmental sensors available for the robots. The subclasses as shown in Figure 13 include *ES1_laser*, *ES2_camera* and *ES3_dgps*. Each of these subclasses has different variables that are needed to store the raw data from each sensor. A *compute_output* function is defined for each of the subclasses and the purpose of that function is to store the data received from the sensors into output variables such as the *output_localize* and *output_laser* arrays.

4.3.3 PS (Perceptual Schema) Class

The *PS* class is defined for all perceptual schemas available for the robots. The subclasses include *PS1_global*, *PS2_goal*, *PS3_global_other*, *PS4_rel_other*, and *PS5_global_comm*. *PS1_global* outputs the robot's global position, *PS2_goal* outputs the robot's goal position, *PS3_global_other* outputs another robot's global position, *PS4_rel_other* outputs a position relative to another robot, and *PS5_global_comm* outputs another robot's global position communicated by that robot. The input for each subclass varies depending on the solution (connections) chosen. The *PS* class is shown in detail in Figure A - 1.

4.3.4 CS (Communication Schema) Class

The *CS* class is defined for all communication schemas available for the robots. As shown in Figure 14, the subclasses of *CS* class include *CS1_self_pos* and *CS2_other_pos*. *CS1_self_pos* communicates the robot's own global position to another robot. *CS2_other_pos* communicates the other robot's global position to that robot.

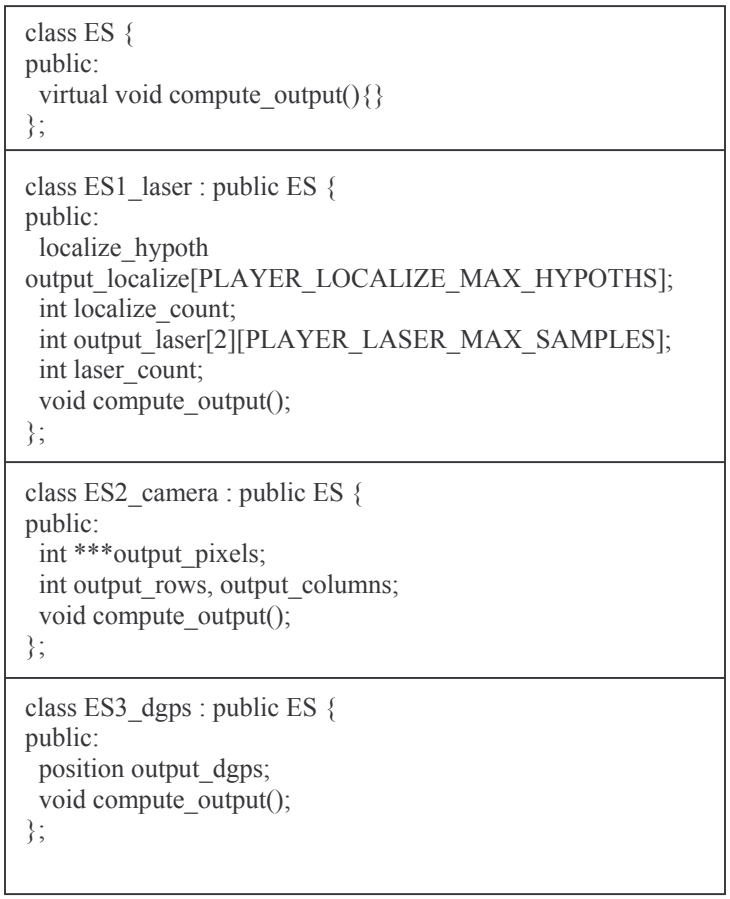


Figure 13. *ES* (Environmental Sensor) class defined for all environmental sensors. The subclasses of *ES* are *ES1_laser* for laser scanner, *ES2_camera* for camera and *ES3_dgps* for DGPS.

```
class CS {
public:
  position input;
  position output;
  virtual void communicate_output(){}
};

class CS1_self_pos : public CS {
public:
  void communicate_output();
};

class CS2_other_pos : public CS {
public:
  void communicate_output();
};
```

Figure 14. *CS* (Communication Schema) class defined for all communication schemas. The subclasses of *CS* are *CS1_self_pos* and *CS2_other_pos*.

4.3.5 MS (Motor Schema) Class

The *MS* class, as shown in Figure 15, is defined for the motor schema available for the robots. Currently there is only one motor schema, which is the Go To Goal Schema. The *move_robot* function moves the robot towards the goal, and the *stop_robot* function stops the robot from moving when it reaches its goal. In situations where a robot needs to keep a line of sight with another robot, the *move_robot* function gives motor commands that coordinate their movement such that one of the robots with a camera or a laser scanner attempts to detect the other robot, which stops moving until it is detected.

4.4 Functions Defined

This section discusses the important functions defined in the implementation. There are three function discussed: *main*, *configure* (*reconfigure*) and *initialize_connections* functions. *Main* function calls the *configure* (*reconfigure*) and *initialize_connections* functions. The *configure* (*reconfigure*) function activates certain schemas based on the solution (connections) chosen. The *initialize_connections* function connects the activated schemas.

```
class MS {
public:
  position input1;
  position input2;
  void move_robot();
  void stop_robot();
};
```

Figure 15. *MS* (Motor Schema) class defined for all motor schemas. Currently there is only one motor schema, so there are no subclasses defined for this class.

4.4.1 Main Function

As we can see from Figure 16, the *main* function first creates an instance of *Robot* class. After it parses the arguments, it calls the configure function, which configures the solution and decides which schemas to be activated. Afterwards, inside an infinite loop, it calls the *initialize_connections* function, which connects all the activated schemas. This function is called infinitely until the robot reaches its goal. For future work, the robot can also monitor the sensors. If any of the sensors fail, the main function can call the *reconfigure* function, which reconfigures the solution based on the most current available schemas.

4.4.2 Configure and Reconfigure Functions

The information of the activated schemas is stored in global array variables shown in Figure 17. If we want to check if ES1 is activated or not, we can check the value of ES[1]. A value of 1 inside ES[1] means that it is activated and 0 means it is not activated.

The *configure* function configures the solution and decides which schemas to activate. Currently, the decision is based on user input and the schemas activated are hard-coded depending on the input received. In the future, the decision can be based on the most efficient solution (connections) based on the cost of computation and communication. Although not yet implemented, an empty function called *reconfigure* is defined. If any of the sensors fails, it reconfigures the solution based on the most current available schemas. The function definition of the *configure* function is given in Figure A - 2.

```
int main(int argc, char** argv) {
  Robot rob;

  parse_args(argc, argv);

  rob.configure(argc, argv);

  sleep(15);
  while(1) {
    rob.initialize_connections();
    if(!monitor_ok)
      rob.reconfigure();
  }

}
```

Figure 16. The *main* function definition.

```
int activated_ES[4] = {0,0,0,0};
int activated_PS[6] = {0,0,0,0,0,0};
int activated_CS[3] = {0,0,0};
int activated_MS = 0;
```

Figure 17. Global array variables that store information about which schemas are activated.

4.4.3 Initialize_Connections Function

The *initialize_connections* function connects schemas that have been activated inside the *configure* (*reconfigure*) function. As shown earlier in Figure 1 and Figure 2, the process always starts from *ES* classes, which receives input from environmental sensors. Afterwards, the *compute_output* on those classes is called inside the *initialize_connections* function. The *compute_output* function calculates and stores the output in the output variables.

Connections are based on the activated schemas as well as additional information passed by the *configure* (*reconfigure*) function, such as *ps1_global.single_input* and *ps4_rel_other.laser_input*. The variable *ps1_global.single_input* tells us whether or not PS1 is expecting one input or two inputs. For example, if ES1 is activated and PS1 is expecting one input, then there is a connection between ES1 and PS1. If PS4 and PS5 are activated and PS1 is expecting two inputs, then there is a connection from both PS4 and PS5 going to PS1. The variable *ps4_rel_other.laser_input* informs us about input expected by PS4. If it is expecting an input from a laser scanner, then a connection from ES1 going to PS4 is made. If an input is expected from a camera, then there is a connection from ES2 to PS4.

4.5 Limitations

Currently, the robots cannot detect their own hardware and thus are not aware of their available environmental sensors. As a result, there is no sensor failure detection in the system and we have to manually input sensor information to the robots. Moreover, as mentioned earlier, we also have to provide information about which schemas to activate. In a sense, the solution or the connections between the schemas are currently hard-coded. Future work will enable the robots to choose the most efficient solution autonomously.

5 RESULTS AND DISCUSSION

This chapter presents results and analysis of our implementation on the physical robots. We do not present simulation results because the vision system using the camera and marker cannot be implemented in the Player-Stage simulation. The results are shown for each of the case studies described previously. Snapshots of the robots moving towards the goal are also given in Section 5.1. The starting position and goal position for each robot are the same for all cases.

The time needed for the robots to accomplish the task (time cost) and the number of communication messages sent (communication cost) were recorded during the experiments. All messages communicated have the same format. A sample message containing Needy's current global position and orientation communicated by Helper is Y$1500$3000$90#. Therefore, the size of each message sent through the communication server is 8 bytes.

Sensing cost refers to the computational cost needed to process the raw data gathered from the sensor. Table 6 shows the approximation of the sensing cost for all sensors used in the implementation. Success rate is defined to be the ratio of completed tasks to the number of attempted tasks in an experiment. Experiments in which all robots reached their goals has a 1.0 success rate, while the ones in which all robots fail has a 0.0 success rate. A success rate of 0.5 is given to an experiment in which only one of the two robots reaches its goal while the other robot fails. Ten experiments were run for each case.

Table 6. Approximation of the sensing cost for all sensors used on this implementation.

Sensors	Sensing Cost	Description
None	0	No sensing cost when no sensors are used.
Camera	1	The cost to process camera data is represented with 1.
Laser and map	2	The cost to localize with laser and map is more than the cost to process data from a camera.
Laser and map + Camera	3	The cost for combination of laser and map with camera is the sum of the individual costs.

5.1 Data

5.1.1 Case 1: Fully Capable Robots (Baseline)

As shown in Figure 18, the fully capable robots move towards their goal without the need for any communication. The final snapshot shows a gap between the robot and its goal position because the robot is assumed to have reached its goal when it is within 500 mm of the goal position. The robot on the right, which is further from its goal point, is called Red, and the robot on the left is called Blue.

Table 7 shows the data collected from the experiments with two fully capable robots moving towards their goal using their own sensors without any communication. The results shown are the average of data taken from 10 experiments. The first column in the table contains the data collected from Red, while the second column contains data collected from Blue. The third column contains the data combined from both robots as a team. The time cost for the robot team is the time taken for all robots to reach their goals.



Figure 18. These snapshots show two fully capable robots moving towards their goal positions with their own laser scanner and with no communication between them. Here, the robot on the right is called Red, while the robot on the left is called Blue (read left to right, top to bottom).

Table 7. Data collected from Case 1 experimentation where two robots move towards their unique, assigned goal without any communication. The results shown are the average and the standard deviation of data taken from 10 experiments.

	Red		Blue		Team	
	Avg	**Std Dev**	**Avg**	**Std Dev**	**Avg**	**Std Dev**
Time Cost (in seconds)	39.1	0.88	28.1	1.85	39.1	0.88
Communication Cost	0	0	0	0	0	0
Sensing Cost	2	0	2	0	4	0
Success Rate	1.0	0	1.0	0	1.0	0

5.1.2 Case 2: Communicating Own Current Position

In case 2, Red is called the Helper and Blue is called Needy because Blue does not have access to its laser scanner and map to localize. Instead, it uses its camera to detect the Helper and wait for the Helper to communicate its position. As shown in Figure 19, Needy attempts to detect Helper in order to calculate its relative position. Using its relative position and Helper's current position communicated, Needy can calculate its current position and move towards the goal. Helper is not moving while Needy is searching so that the line of sight is more likely to be maintained. As soon as Needy detects Helper, it communicates a message to Helper stating that it can move towards its own goal. When Needy loses sight of Helper, it communicates a message to Helper asking it to stop moving while it is searching.

In this implementation, Needy turns around in place when attempting to detect Helper. For this reason, the starting position for both robots is placed closely together because the camera can only reach a limited range of sight. The camera we are using can only view up to 60 degrees facing forward. Thus, the distance between the starting and goal position is made shorter for Needy and longer for Helper in order to keep Helper in sight. If Needy fails to detect Helper after 2 minutes, Helper gives up helping and goes towards its own goal. This ensures that if Needy does not accomplish the task, Helper still does.

Table 8 shows the data collected from Case 2 experiments, in which Needy uses its camera to detect Helper. The average time for Needy to reach the goal is higher than that for Helper because often times Helper reaches its goal first but still communicates its current position to Needy. At this point, Needy needs to keep looking for Helper, which stops moving because it has reached its goal, to localize and move towards the goal at the same time. Most of the time, this takes a while because Helper's position and its goal position are in different directions.



Figure 19. These snapshots show the Needy robot (on the left) looking for the Helper robot (on the right) to calculate its position relative to the Helper. With the relative position combined with Helper's position communicated, Needy can calculate its current position and move towards the goal (read left to right, top to bottom).

Table 8. Data collected from Case 2 experimentation where one of the robots does not have enough information to localize and uses its camera to detect the other robot. The results shown are the average and the standard deviation of data taken from 10 experiments.

	Helper		Needy		Team	
	Avg	**Std Dev**	**Avg**	**Std Dev**	**Avg**	**Std Dev**
Time Cost (in seconds)	108.33	30.51	125.67	28.71	125.67	28.71
Communication Cost	46.83	1.12	57.83	7.60	104.66	7.76
Sensing Cost	2	0	1	0	3	0
Success Rate	1.0	0	0.6	0.52	0.8	0.41

5.1.3 Case 3: Communicating Other's Current Position

As shown in Figure 20, Helper assists Needy by attempting to detect and calculate Needy's position relative to itself. Based on the relative position and its own current position, Helper can calculate and communicate Needy's position. Needy is not using any of its sensors to go towards the goal; instead it waits for Helper to communicate its current position. Needy only moves when Helper has communicated its current position. In 10 experiments, Helper always accomplished the task, while Needy failed once. Data collected from the experiments is shown in Table 9. The sensing cost for Needy is 0 because no sensors are used.

5.2 Analysis

As shown in Figure 21, the time taken for the robots to reach their goals increases from Case 1 to Case 3. In Case 1, the robots directly move towards the goal without any



Figure 20. These snapshots show the Helper robot (on the right) assisting the Needy robot (on the left) by attempting to detect and calculate Needy's position relative to itself. In this case, Needy is not using any of its sensors to go towards the goal. Needy waits for Helper to communicate its current position (read left to right, top to bottom).

Table 9. Data collected from Case 3 experimentation where one of the robots does not use any of its sensors and waits for other robot to detect it. The results shown are the average and the standard deviation of data taken from 10 experiments.

	Helper		Needy		Team	
	Avg	Std Dev	Avg	Std Dev	Avg	Std Dev
Time Cost (in seconds)	279.33	76.86	255.44	78.39	279.33	76.86
Communication Cost	57.33	14.20	1	0	58.33	14.20
Sensing Cost	3	0	0	0	3	0
Success Rate	1.0	0	0.9	0.32	0.95	0.22

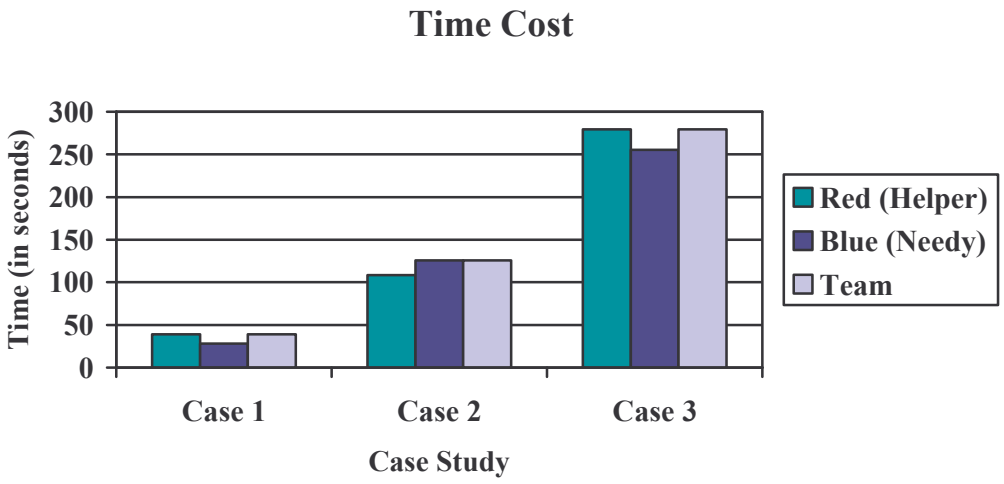


Figure 21. Time cost for robots in Case 1, Case 2, and Case 3 experiments.

distraction. However, in Cases 2 and 3, Helper needs to wait and help Needy before moving towards the goal. Thus, it takes longer for all robots to reach their goal positions. Figure 22 shows the communication cost for the robots in all cases. Case 1 does not have any communication cost. Case 2 has more communication cost than Case 3, because Needy has to communicate Stop and Go messages to Helper to keep its line of sight. Also, even after Helper reached its goal, it still communicates its current position to Needy. In Case 3, however, Needy only moves when it obtains its current position communicated by Helper. Thus, Helper does not need to communicate a Stop and Go message to Needy. The sensing cost in all case studies is shown in Figure 23. Sensing in Case 1 costs the most because both robots use laser scanner and map, which has the most expensive sensing cost. By combining the time, communication and sensing cost information, future research can develop techniques for choosing the most optimal and efficient solution from all possible choices.

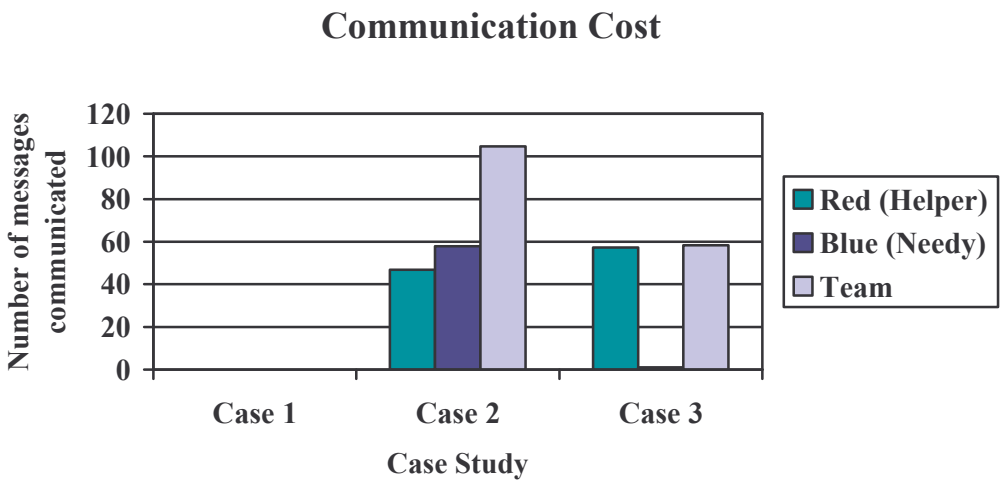


Figure 22. Communication cost for robots in Case 1, Case 2, and Case 3 experiments.

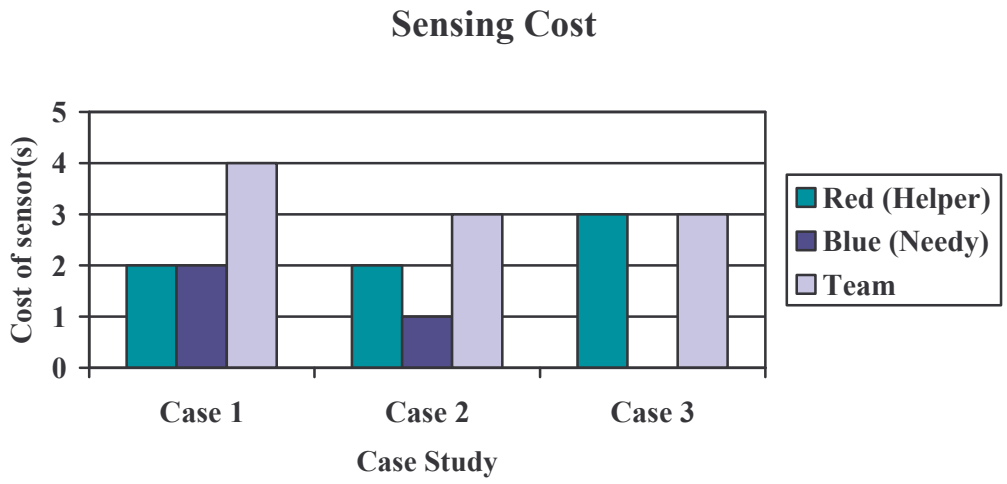


Figure 23. Sensing cost for the robots in Case 1, Case 2, and Case 3 experiments.

As mentioned previously, success rate refers to the ratio of completed to attempted tasks in an experiment. In Figure 24, the success rate for robot team in Case 1 is 1.0 because the robots are not dependant on each other and can always reach their goal points using their own sensors. From 10 experiments conducted in Case 2, Needy failed 4 times. In Case 3, Needy only failed once out of the 10 experiments. Thus, the robot team in Case 3 has higher success rate than that in Case 2. In all three cases, Helper always has a success rate of 1.0 because if the robot with a camera fails to detect the robot with a laser scanner after two minutes, Helper gives up helping and moves directly towards its own goal position.

The failures of Needy in Case 2 and Case 3 are caused by the wrong calculation of its current position. Sometimes the lighting and the background can cause false detection of the vision-based marker detection. From the snapshots shown previously, we can see that in the hallway, the lights are located on the left, above Needy. There are no lights on the right side, where Helper is located. This can contribute to the failures in

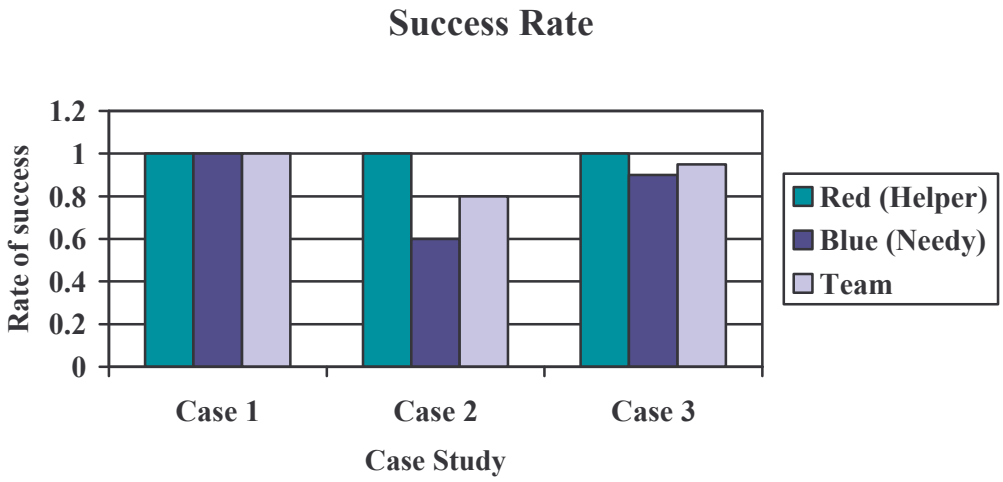


Figure 24. Success rate for the robots in Case 1, Case 2, and Case 3 experiments.

obtaining the correct information from the marker. In that case, the current position calculated is incorrect and causes the robot to assume a different position rather than the one at which it is currently located.

If the Helper and Needy robots did not cooperate using our reconfigurable software technique in Case 2 and Case 3, Needy could not reach its goal position because it does not have enough information to calculate its current position. Helper, however, could always reach its goal position. Thus, the success rate in this scenario is always 0.5 since only one of the two robots can accomplish the task. Figure 25 shows the comparison of the success rate for the robot team in our three case studies with that of the case where the robots do not cooperate in Case 2 and Case 3. This figure confirms that our software reconfigurability approach produces results that have higher success rates than are possible without the approach.

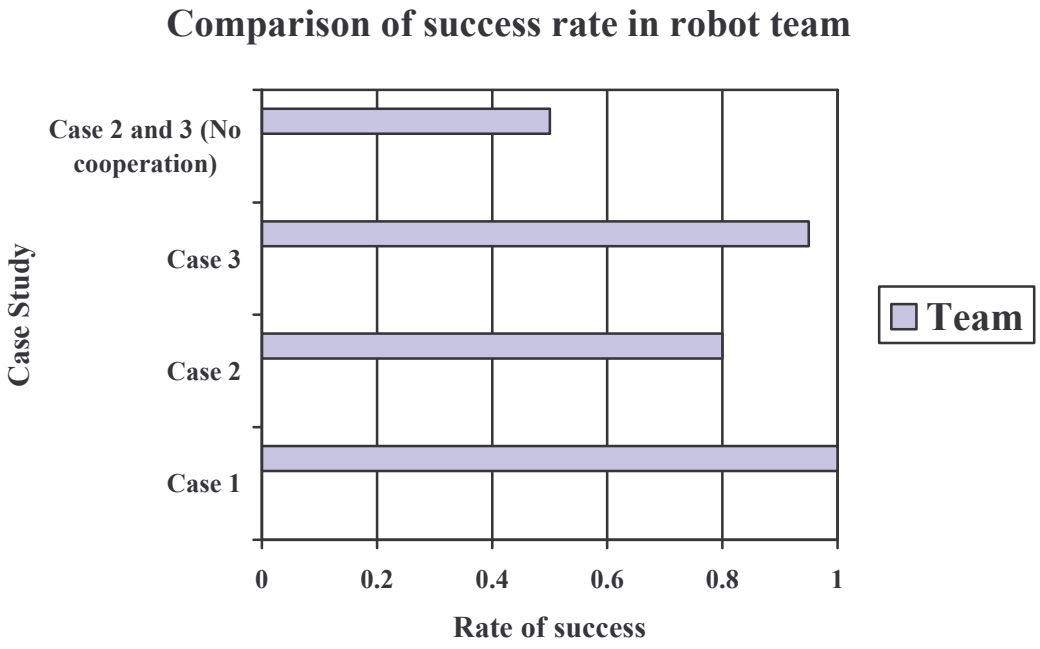


Figure 25. Success rate for Case 1, 2, and 3 experiments compared with that of the case robots do not collaborate in Case 2 and 3. In this case, since Needy is not capable of moving to the goal by itself, only one of two robots in the team can make it to its goal. Thus, the average success rate is 0.5 without cooperation.

6 RELATED WORK

A survey of the current research in multi-robot systems is discussed in (Arai *et al.*, 2002). The seven principle topics of multi-robot system addressed by Arai *et al*. are Biological Inspirations; Communications; Architecture, Task Allocation, and Control; Localization, Mapping and Exploration; Object Transport and Manipulation; Motion Coordination; and Reconfigurable Robots. The topics most closely related to this thesis are Communications, Localization, Mapping and Explorations and Motion Coordination. This chapter discusses previous work in multi-robot cooperation in four areas: Task Manipulation, Fault Tolerance, Autonomy in Heterogeneous Robot Teams, and Information Invariants.

In task manipulation, roles or tasks are dynamically assigned to the robot team. When any unexpected events occur, other solutions can still be implemented to accomplish the task. Task manipulation approach also aims for fault tolerance and autonomy in heterogeneous robot teams. Fault tolerance and autonomy have been the primary focus of research in heterogeneous robot teams. Fault tolerance refers to the ability of the robot team to recover from any failure or faults that happen during run-time. Autonomy is the capability of the robot team to make a decision among itself and maintain its activity in accomplishing the task in any events. Information invariants determine the information requirements to perform robot tasks. For example, in our approach, a robot's current position can be obtained either with information from its own sensors, or from information based on a position of other robot relative to itself combined with the current position of that other robot. Using their idea, optimization techniques can be used to select the most efficient solution in accomplishing the task.

6.1 Task Manipulation

In (Chaimowicz *et al.*, 2001), the authors describe a framework for coordinating multiple robots in the execution of a cooperative manipulation tasks. The framework uses models and metrics of computation, control, sensing, and information in order to dynamically assign roles to the robots and define the most suitable control hierarchy given the requirements of the task and characteristic of the robots. To show the implementation of the framework, the authors developed an object oriented simulator called MUROS that allows the user to create different types of robots and to define different simulators, various control algorithms and communication protocols.

The framework proposed using a dynamic reconfigurable leader-follower approach where the robots can change roles during the task execution, allowing the team to adapt better to unexpected events such as obstacle detection, sensor failures, etc. Results from experiments with two physical robots and with the simulator are used to explore the multi-robot coordination in a cooperative manipulation tasks. Future works will include the use of new metrics for the leadership change and the study of important aspects of the framework such as the impact of communication failures in the exchange of information.

In a sense, this prior research also uses software reconfigurability idea because it reconfigures its solution (software) during run-time. In this leader-follower approach, the robots change roles, either as leader or follower, when run into unexpected events. However, it does not handle the situation where all sensors in a robot failed and it cannot be either leader or follower. The robot with failed sensors end up doing nothing because of its inability to play any roles in the task execution. The approach of this thesis takes into account all available sensors on the robots and determines a possible way for the robots to help each other in order to accomplish the task. For example, in the situation mentioned previously, if the leader robot has a camera and can see the robot that lost all

its sensors, the leader can send the other robot motor commands so that it can keep following the leader.

6.2 Fault Tolerance

Fault tolerance is one of the important issues in heterogeneous robot team coordination. To avoid interference in accomplishing the task, each robot inside the team needs to be able to detect its own failure and if possible, recover from it. If it is not possible to recover, it needs to communicate that to the team, which then needs to redistribute the tasks if the failed robot becomes totally inoperable.

Long, Murphy and Parker (Long *et al*., 2003) introduce an approach that allows teams of robots to share knowledge about the working environment, sensor and task state, to diagnose failures and also communicate to redistribute tasks in the event that a robot becomes inoperable. The recovery strategies in their approach include: reconfiguration of the perceptual schema, recalibration of the sensor, or other corrective actions. Reconfiguration involves searching for a new sensor logically equivalent to the failed sensor. Recalibration is a complex task that reconfigures the sensing device on certain occasions when needed, e.g. a low-light situation, instead of looking for a new sensor right away. Other corrective actions involve moving or shaking the robots in case the sensor is affected by something physically, for example if a cloth is placed over its camera. The implementation of this approach is done in a simplified search operation.

The architecture used is the Sensor Fusion Effects (SFX) core architecture. The key idea of this architecture is Sensing Manager, which consists of Error Classifier and Error Recovery. When a failure is detected, the Error Classifier runs some tests to find out the cause of the failure. Once the cause is known, Error Recovery attempts to recover. If it is not possible, it communicates that to the team, or a human operator. This approach was demonstrated on both single-agent and multi-agent systems. In all cases,

the system diagnosed and recovered from operator-introduced faults to complete the mission. The current implementation assumes the robots are located within the same region, and that the actual environment is equivalent for both robots. Since the implementation was done indoor, future work would involve testing the work in outdoors and in non-static environments.

The software reconfigurability idea is fault tolerant because it allows for more than one solution to solve a task. It can choose from different ways for the robots to help each other in solving the task by looking at all available sensors in the robot team rather than on individual robots. If a sensor failure is detected, it re-examines the remaining available sensors in the robot team to reconfigure a new solution. Although sensor failure detection is not yet implemented in this thesis, it will be an important part to develop in the future.

6.3 Autonomy in Heterogeneous Robot Teams

Parker states that the most common approach to cooperative team design is to custom-build a specific robot team for a specific mission (Parker, 1998). More recently, different kinds of robots can be built in advance and then applied to specific applications when needed. However, what is really desired in heterogeneous team cooperation is to have an automated and dynamic team design. Her article proposed an approach to automated team design that combines the theories of *sensori-computational systems* and *information invariants*, building upon the earlier work of Donald *et al.* The first step of the approach defines the information invariants required by the mission and then mapping these invariants to the equivalence class of *sensori-computational* systems that can achieve that invariant. The next step selects a subset of sensori-computational systems that is the most efficient solution, based upon predetermined subsystem costs. The final step distributes the required team components to individual team members, based upon the mission metrics.

The case studies in which this approach is studied are: keeping formation, "mock" hazardous waste cleanup, and cooperative observation. In each of these case studies, the mission metric, the information invariants and the costs are identified. The ongoing work would then be to map the information invariants for these case studies to the equivalent classes of sensori-computational systems required to achieve these invariants, to optimize the solution to a minimal subset of components, and to map these components to individual team members.

As mentioned before, this article, which ultimate goal is to have an automated cooperative team design, relates most to the software reconfigurability approach in this thesis. With software reconfigurability, developers do not need to pre-design the software (solution) for the heterogeneous robot teams to accomplish a task. Instead, team design can be automated during run-time. Moreover, this approach is fault tolerant because even though one of the sensors or robots fails, the team can still reconfigure their software (solution) based on the remaining sensors or robots. As in this article, our approach also makes use of the information invariants idea in reconfiguring the software and optimizing the solution, although the optimization part is not yet implemented in this thesis.

Another related work that focuses on autonomous heterogeneous robot cooperation is described in (Parker *et al.*, 2004). This article shows a general approach for enabling more capable robots to assist in the navigation of sensor-limited robots. In this approach, the authors use cooperation among teams of heterogeneous robots that involves a navigational assistant called the *Helper Robot* guiding the *Simple Robots* to their desired positions. The *Helper Robot* uses a laser scanner for localization, along with a vision system for autonomously teleoperating the *Simple Robots* into position. The *Simple Robots* make use of simple vision capability for color blob tracking to achieve follow-the-leader formation-keeping over long distances.

Their approach has been successfully implemented on a team of physical robots. Example results of the implementation have been presented for a large team of *Simple Robots* and *Helper Robots* (totaling 100 robots). The future work is aimed at incorporating fault-tolerance into the system, so that if robots fail during the deployment process, they are able to successfully recover at the team level. This thesis makes use of this prior research. However, this thesis significantly extends this prior work by identifying and constructing building blocks that allow easy reconfiguration of schemas distributed across multiple robots. This thesis enables multiple solution strategies to be selected simply by reconnecting the schema modules.

6.4 Information Invariants

Information Invariants is a concept that has become a foundation of several current works in heterogeneous robot team cooperation. It involves the problem of determining the information requirements to perform robot tasks. Donald develops the basic concepts and tools behind information invariants in plain language by showing a number of motivating examples (Donald, 1995). Then, it provides a fairly detailed analysis that admits more sophisticated models of sensors and computation.

According to Donald, information invariants can serve as a framework in which to measure the capabilities of robot systems and the equivalences that can be derived between communication, external state, internal state, computation and sensors, can prove valuable in determining what information is required to solve the task, and how to direct a robot's actions to acquire that information to solve it. His theory compares members of a certain class of sensor systems to transform one system into another. However, it does not permit one to judge which system is simpler, better or cheaper. Therefore, one cannot identify the simplest system to find the most efficient solution to a task. This is a potential work to be developed in the future.

In other work by Donald, Jennings and Rus, information invariants are measured and analyzed using a pair of cooperating mobile robots for manipulation (Donald *et al.*, 1997). This article also discusses also the development and analysis of synchronous and asynchronous manipulation protocols for a small team of cooperating mobile robots that can push large boxes. The goal is to investigate the information requirements for robot tasks. The results from this work still leave a number of open issues. For example, can robots externalize, or record state in the world? Can we record programs in the world in the same way we may externalize state? Is there a universal manipulation circuit that can read these programs and perform the correct strategy to accomplish a task? These questions will open up future research in information invariants.

Both of these articles show the idea of information invariants and how to use it to accomplish our missions. In (Parker, 1998), information invariants required by different missions are defined. She proposes to use the notion of information invariants to define equivalence classes of sensori-computational systems that serve as fundamental building blocks (or basis functions) of more complex cooperative robotic behaviors. Optimization techniques can then be used to generate cooperative robot teams that best fit the specifications of the desired mission. The same approach can be implemented in the future, but as mentioned previously, this thesis does not include optimization using information invariants.

7 FUTURE WORK AND CONCLUSIONS

7.1 Future Work

As mentioned previously, the robots in this implementation are not able to detect their hardware configuration and thus, are not aware of their available environmental sensors. Because of this, sensor failure detection is not available in the system and we have to manually input sensor information to the robots. Software reconfigurability is not fault tolerant without sensor failure detection because the robot team needs to be aware of changes in its environmental sensors so that the robots can decide whether the software needs to be reconfigured in order to accomplish the task. When reconfiguration is needed, each robot needs to recognize its own sensors in order to re-evaluate the available schemas (building blocks) on all robots.

Moreover, after all available schemas are identified, we also have to provide information about which schemas to be activated. Currently, the solution or the connections between the schemas are hard-coded. Different connections are obtained by different inputs from the user. However, since the aim of this approach is to gain automated solution design, the robots need to have the capability of choosing the most optimal and efficient solution during run-time. As shown previously in Figure 21, Figure 22, Figure 23, the time, communication and sensing cost of different solutions can be obtained for a robot team. Based on the costs, an algorithm to select the least costly and the most reliable solution can be developed in the future.

In the current implementation, obstacle avoidance is only accomplished by robots with the needed sensors, which in this case is a laser scanner. Avoiding obstacles using a color camera is also possible, but it is not yet implemented. If a robot loses all of its sensors, it cannot avoid any obstacles and are exposed to the danger of bumping into the

walls or falling down the stairs. For this reason, the future implementation of collaborative obstacle avoidance is recommended. The more capable robots can help the less capable robots in avoiding obstacles and thus decreasing the chance of damaging any team member.

As discussed earlier, the starting and goal positions for the robots are presently set such that it helps in maintaining line of sight for the robot with a camera. In Case 2, Needy turns in place until it detects the Helper. This does not work when the robots are placed too far from each other, since the camera used has limited range of sight. Additionally, Helper does not move until detected by Needy, while Needy only turns in place without moving any closer to Helper. A solution to this problem is to either put Needy or Helper in wandering mode until they find each other. In addition, this implementation has only attempted the collaboration of two robots. Experiments on more than two robots with multiple schemas connected across multiple robots will need to be done in the future.

Since our results show some failures in accomplishing the task due to the false detection of the visual marker, a more robust and fault-tolerant marker detection method can be developed in the future. Moreover, not all sensors have been investigated, including DGPS and a laser scanner without a map. A laser scanner without a map can be used to detect the relative position and orientation of another robot. With the right algorithms, it can also be used for obstacle avoidance.

7.2 Conclusions

This thesis has introduced the software reconfigurability approach, which enables the software solution in accomplishing a task to be reconfigured during run-time. Implementation of this approach includes modularizing the code to create schemas (building blocks) and showing different ways a task can be solved based on the same

blocks. The task chosen to illustrate a proof-of-concept is for each robot to go towards its assigned goal position. Three case studies are used to show different solutions for solving the same task using the same robot team and schema modules. The implementation results show that all experiments in our case studies have higher success rate than the one without the approach. This proves that our approach of solving a simple task by collecting the available schemas (building blocks) on all robots and connecting them in different ways to accomplish the task in multiple ways is feasible.

In conclusion, our results show that the distributed schemas and modularized code provides a valid architecture basis for the software reconfigurability approach. Using the schemas and architecture shown in this thesis, future work can be developed to autonomously reconfigure the available schemas when unexpected events occur. A framework for an autonomous and fault tolerant multi-agent system can be developed because this approach produces more than one solution to accomplish a task and solution design is automated during run-time.

LIST OF REFERENCES

LIST OF REFERENCES

Tamio Arai, Enrico Pagello and Lynne E. Parker. *Editorial: Advances in Multi-Robot Systems*. IEEE Transactions on Robotics and Automation, vol. 18, no. 5, pp. 655-661, 2002. Available: citeseer.nj.nec.com/582200.html.

Ronald Arkin. *Motor schema-based mobile robot navigation*. International Journal of Robotics Research, vol. 8, no. 4, pp. 92-112, 1898.

Luiz Chaimowicz and Vijay Kumar and Mario Campos. *Framework for coordinating multiple robots in cooperative manipulation tasks*. Proceedings of SPIE Vol. 4571, Newton, Massachusetts, USA, 2001.

B. R. Donald. *On Information invariants in robotics*. Artificial Intelligence, vol 72, pp. 217-304, 1995.

B. R. Donald and J. Jennings and D. Rus. *Information invariants for distributed manipulation*. International Journal of Robotics Research, vol 16, no. 5, pp. 673-702, 1997.

Brian P. Gerkey, Richard T.Vaughan, Kasper Stoy, Andrew Howard, Gaurav S Sukhtame and Maja J Mataric. *Most Valuable Player: A Robot Device Server for Distributed Control*. Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS), Wailea, Hawaii, pp. 1226-1231, October 2001. Available: citeseer.nj.nec.com/gerkey01most.html.

M. Long, Robin R. Murphy and Lynne E. Parker. *Distributed Multi-Agent Diagnosis and Recovery from Sensor Failures*. Proceedings of IEEE International Conference on Intelligent Robots and Systems, 2003. Available: http://www.cs.utk.edu/~parker/publications/IROS03-DistSFXEH.pdf.

Lynne E. Parker. *Toward the Automated Synthesis of Cooperative Mobile Robot Teams*. Proceedings of SPIE Mobile Robots XIII, vol 3525, pp. 82-93, 1998. Available: citeseer.nj.nec.com/parker98toward.html.

Lynne Parker, Balajee Kannan, Fang Tang, Michael Bailey. *Tightly-coupled Navigation Assistance in Heterogeneous Multi-Robot Teams*. Manuscript submitted for publication. Distributed Intelligence Laboratory, Department of Computer Science, University of Tennessee, Knoxville. 2004

APPENDICES

APPENDIX 1

```
class PS1_global : public PS {
public:
  int single_input;
  int laser_input;
  localize_hypoth input_localize[PLAYER_LOCALIZE_MAX_HYPOTHS];
  int localize_count;
  vector input1;
  position input2;
  position input_pos;
  position output_pos;
  PS1_global() { single_input = 1;  }    /* Default to having a single input instead of two */
  void compute_output();
};

class PS2_goal : public PS {
public:
  position output_pos;
  void compute_output();
};

class PS3_global_other : public PS {
public:
  vector input1;
  position input2;
  position output_pos;
  void compute_output();
};

class PS4_rel_other : public PS {
public:
  int laser_input;
  int input_laser[2][PLAYER_LASER_MAX_SAMPLES];
  int ***input_pixels;
  int input_rows, input_columns;
  int laser_count;
  vector output_vec;
  PS4_rel_other() { laser_input = 1; }  /* Default to having input from laser instead of camera */
  void compute_output();
};

class PS5_global_comm : public PS {
public:
  position input_pos;
  position output_pos;
  void compute_output();
};
```

Figure A - 1. PS (Perceptual Schema) class defined for all agents.

APPENDIX 2

```
void Robot::configure(int argc, char** argv){

 /* Instantiate Proxies */
 robot = new PlayerClient(host,port);
 pp = new PositionProxy(robot,0,'a');
 localp = new LocalizeProxy(robot, 0, 'r');
 lp = new LaserProxy(robot,0,'r');
 tp = new TruthProxy(robot,0,'r');
 cp = new CommsProxy(robot,0, 'a');


 /* Robot with laser & map to localize itself and not broadcasting
    its position */
 if(atoi(argv[argc-1]) == 11){
   activated_ES[1] = 1;
   activated_PS[1] = 1;
   activated_PS[2] = 1;
   activated_MS = 1;
   ps1_global.single_input = 1;

/* initializing localization */
   robot->Read();
   pp->SetSpeed(40,0);
   while(localp->hypoth_count==0){
     robot->Read();
     localp->Print();
   }

 /* Robot with laser & map to localize itself and broadcasting
    its position to help another robot */
 } else if(atoi(argv[argc-1]) == 12){
   activated_ES[1] = 1;
   activated_PS[1] = 1;
   activated_PS[2] = 1;
   activated_CS[1] = 1;
   activated_MS = 1;
   ps1_global.single_input = 1;

  /* initializing localization */
   robot->Read();
   pp->SetSpeed(40,0);
   while(localp->hypoth_count==0){
     robot->Read();
     localp->Print();
   }
```

Figure A - 2. Function definition for configure defined in Robot class.

```
/* Robot with laser but no map to localize */
} else if (atoi(argv[argc-1]) == 13){
  activated_ES[1] = 1;
  activated_PS[1] = 1;
  activated_PS[2] = 1;
  activated_PS[4] = 1;
  activated_PS[5] = 1;
  activated_MS = 1;
  ps1_global.single_input = 0;
  ps4_rel_other.laser_input = 1;

/* Robot with laser and map to localize and
   calculate another robot's
   position using its camera */
} else if (atoi(argv[argc-1]) == 14){
  activated_ES[1] = 1;
  activated_ES[2] = 1;
  activated_PS[1] = 1;
  activated_PS[2] = 1;
  activated_PS[3] = 1;
  activated_PS[4] = 1;
  activated_CS[2] = 1;
  activated_MS = 1;
  ps1_global.single_input = 1;
  ps4_rel_other.laser_input = 1;
  /* initializing localization */
  robot->Read();
  pp->SetSpeed(40,0);
  while(localp->hypoth_count==0){
    robot->Read();
    localp->Print();
  }

/* Robot uses no sensors and waits for
   another robot to send out its global
   position */
} else if (atoi(argv[argc-1]) == 21){
  activated_PS[1] = 1;
  activated_PS[2] = 1;
  activated_MS = 1;
  ps1_global.single_input = 1;
```

Figure A - 2. Continued.

```
/* Robot with camera to calculate relative
   distance to other robot, and together
   with that robot's global position communicated
   calculate its own global position */
 } else if (atoi(argv[argc-1]) == 22){
   activated_ES[2] = 1;
   activated_PS[1] = 1;
   activated_PS[2] = 1;
   activated_PS[4] = 1;
   activated_PS[5] = 1;
   activated_MS = 1;
   ps1_global.single_input = 0;
   ps4_rel_other.laser_input = 0;

  /* Robot with GPS, currently using TruthProxy in Stage
     and not available to use with real robots */
  } else if(atoi(argv[argc-1]) == 3){
   activated_ES[3] = 1;
   activated_PS[1] = 1;
   activated_PS[2] = 1;
   activated_CS[1] = 1;
   activated_MS = 1;
   ps1_global.single_input = 1;

  } else {
   printf("Wrong command\n");
   exit(1);
  }

}
```

Figure A - 2. Continued.

APPENDIX 3

```
void Robot::initialize_connections(){

 if(activated_ES[1] && activated_PS[1] && ps1_global.single_input) {
  printf("Connecting ES1 and PS1\n");
  es1_laser.compute_output();
  ps1_global.localize_count = es1_laser.localize_count;
  int i;
  for(i=0;i<ps1_global.localize_count;i++)
   ps1_global.input_localize[i] = es1_laser.output_localize[i];

 } else if(activated_ES[3] && activated_PS[1] && ps1_global.single_input) {
  printf("Connecting ES3 and PS1\n");
  es3_dgps.compute_output();
  ps1_global.input_pos = es3_dgps.output_dgps;

 }

 if(activated_ES[1] && activated_PS[4] &&  ps4_rel_other.laser_input){
  printf("Connecting ES1 and PS4\n");
  es1_laser.compute_output();
  ps4_rel_other.laser_count = es1_laser.laser_count;
  int i;
  for(i=0;i<ps4_rel_other.laser_count;i++){
   ps4_rel_other.input_laser[0][i] = es1_laser.output_laser[0][i];
   ps4_rel_other.input_laser[1][i] = es1_laser.output_laser[1][i];
  }

 } else if(activated_ES[2] && activated_PS[4] &&  !ps4_rel_other.laser_input) {
  printf("Connecting ES2 and PS4\n");
  es2_camera.compute_output();
  ps4_rel_other.input_pixels = es2_camera.output_pixels;
  ps4_rel_other.input_rows = es2_camera.output_rows;
  ps4_rel_other.input_columns = es2_camera.output_columns;
 }
```

Figure A - 3. Function definition for initialize_connections defined in Robot class.

```
if(activated_PS[4] && activated_PS[5] && !ps1_global.single_input) {
   printf("Connecting PS4 and PS5\n");
   ps4_rel_other.compute_output();
   ps5_global_comm.compute_output();
   ps1_global.input1 = ps4_rel_other.output_vec;
   ps1_global.input2 = ps5_global_comm.output_pos;
  }

  ps1_global.compute_output();

  if(activated_PS[3]) {
   printf("Connecting PS4 and PS1 with PS3\n");
   ps4_rel_other.compute_output();
   ps3_global_other.input1 = ps4_rel_other.output_vec;
   ps3_global_other.input2 = ps1_global.output_pos;
   ps3_global_other.compute_output();
  }

  if(activated_CS[1] ) {
   printf("Communicating CS1\n");
   cs1_self_pos.input = ps1_global.output_pos;
   cs1_self_pos.communicate_output();

  } else if (activated_CS[2]) {
   printf("Communicating CS2\n");
   cs2_other_pos.input = ps3_global_other.output_pos;
   cs2_other_pos.communicate_output();
  }

  printf("move or stop robot\n");
  ps2_goal.compute_output();
  ms.input1 = ps1_global.output_pos;
  ms.input2 = ps2_goal.output_pos;


  cur_dist = calculate_distance(ms.input1, ms.input2);

  if( !my_goal_reached && (cur_dist  > 500  ||
      (cur_dist <= 500 && needy_case2 && wrong)))
     ms.move_robot();
  else
     ms.stop_robot();

}
```

Figure A - 3. Continued.

VITA

Maureen Chandra was born in Medan, Indonesia on June 1, 1979. She went to Budi Murni Elementary School before she moved to Jakarta, Indonesia at age of 10. In Jakarta, she went to Tarakanita Elementary and Junior High School. She graduated from Tarakanita High School in 1997. From there, she went to the University of South Carolina, Spartanburg. In 1999, she transferred to the University of Tennessee, Knoxville and received a B.S. in Computer Science with a minor in Mathematics in 2001.

Chandra is currently pursuing her Master's degree in Computer Science at the University of Tennessee, Knoxville, TN.