A BEHAVIOR-BASED BLACKBOARD ARCHITECTURE FOR MULTI-ROBOT CONTROL

by

JONATHAN T. MCCLAIN

(Under the Direction of Walter D. Potter)

ABSTRACT

This thesis focuses on the development of a robust architecture for the control of multi-robot teams. In particular, the use of a behavior-based blackboard architecture for multi-robot control is presented, with comparisons to other popular architectures for control such as the hierarchical architecture and the subsumption architecture. The proposed architecture was tested by developing a prototype multi-robot system called the Collective, a team of mobile robotic tanks. Preliminary testing of the architecture was done by developing behaviors to accomplish a coordinated mapping task, where the goal for the robots was to combine efforts to produce a map of a static environment (in this case an office suite).

INDEX WORDS: Behavior-Based Robotics, Blackboard Architecture, Multi-Robot Systems, Robotic Mapping

A BEHAVIOR-BASED BLACKBOARD ARCHITECTURE FOR MULTI-ROBOT CONTROL

by

JONATHAN T. MCCLAIN

B.A., The Monmouth College, 2002

A Thesis Submitted to the Graduate Faculty of The University of Georgia in Partial Fulfillment

of the Requirements for the Degree

MASTER OF SCIENCE

ATHENS, GEORGIA

2004

A BEHAVIOR-BASED BLACKBOARD ARCHITECTURE FOR MULTI-ROBOT CONTROL

by

JONATHAN T. MCCLAIN

For God

ACKNOWLEDGEMENTS

The development of a project of this size is never accomplished alone. While I have the honor of having my name attached to this work, there are many others who have helped this idea become a reality. I would first like to thank the members of my thesis committee Dr. Adam Goodie for the many insightful suggestions and Dr. Michael A. Covington for taking the time to answer the numerous electronics questions I encountered throughout this project. I would like to thank my major professor, Dr. Walter D. Potter, for always leaving the door open, for donuts on the weekend, and for pushing me beyond my own expectations in every stage of my academic career here at The University of Georgia. I would also like to thank all the students at the Artificial Intelligence Center, specifically my fellow teammates David Barnhard and B. J. Wimpey for helping with robot construction, for the late nights debugging code, and for the many helpful suggestions in regards to this work. In addition, Jaymin Kessler contributed numerous long hours of debugging code; I thank him greatly for that. I would like to thank my parents, Pat and Laura Benchik, and my grandparents, Ed and Shirley Singleton, for all their love and support throughout my life. To my wife Trisha, no quantity of words could express how much you have given me, I love you dearly. Finally, I would like to thank God, for it was He who placed this path before me, and it is He that deserves all the praise and glory that comes from it.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Robotics as a whole can still be classified as an emerging field. While a large amount of success has been achieved in static environments such as manufacturing, robotics has yet to make a significant impact on tasks within more dynamic environments without incurring costs in the form of environment modifiers (e.g. a track in the floor) to simplify the task. Most of these tasks require robots to be able to move about an environment, creating a need for robust mobile robotic systems. In addition, while robotic engineering has made leaps and bounds towards robots with human-like capabilities, achieving the full potential of these robots still requires human intervention on a regular basis (e.g. Honda's Asimo Robot and the Mars Exploration Rovers). This intervention limits the effectiveness of these robots by not allowing them to be autonomous. However, it is possible for a group of simpler robots to accomplish the same task as a larger more complicated robot. In addition, the prospect of endowing these simpler robots with enough intelligence to make them autonomous is, at this point, more feasible than with other more complicated robots. At the same time, the lower cost of the individual robots, along with the fact that there is a group of robots rather than just one, allows for the possibility of accepting failure in a few of the robots, making the group as a whole more resistant to failures. All of these considerations raise the importance of research into multi-robot teams. This thesis explores the development of a team of mobile robots called *The Collective*, with a particular emphasis on the development of a behavior-based blackboard control architecture to manage the team in order to accomplish a mapping task.

1.1 PROJECT GOALS

The primary goal of this project was to develop a robust and efficient control architecture for multi-robot team experimentation. The project began by investigating the various control architectures for both single and multi-robot systems. Once the architecture was designed, developing a prototype system to test the viability of the architecture became a secondary goal. This process encompassed the creation of a team of robots with the following specifications: they had to be mobile; they had to have adequate sensory capabilities to accomplish a mapping task; and they had to be able to communicate with each other. Once the team was created and the control architecture implemented, the system was tested by programming the robots to accomplish a coordinated mapping task.

The mapping task is performed within a static office suite environment, in this case the office suite of the Artificial Intelligence Center at The University of Georgia. The environment consists of a number of offices connected by a hallway. The goal of the robot team was to create an accurate map of the environment. A secondary goal was to use the advantage of having multiple robots to increase the efficiency of the task compared to a single robot.

1.2 BEHAVIOR-BASED ROBOTICS

Behavior-based robotics is the dominant approach to robot design today. Based on the Behaviorist line of research in psychology, a behavior-based robotic system divides control of a robot into small, independent behavioral modules, whose combined efforts produce the final actions of the robot. A behavior-based robot lacks central control, creating a need to manage which behaviors have control of the robot at a given time. This management is called *arbitration* and is the key aspect of any behavior-based approach. Development of a behavior-based system is characterized by dividing the task up into simpler tasks, which when accomplished together accomplish the main task. The developer then designs behaviors to accomplish each of the subtasks, which are, in general, easier to accomplish than the main task. A general overview of behavior-based robotics is presented in Chapter 2.

1.3 USING BLACKBOARDS FOR CONTROL

A blackboard system in general, is a distributed, opportunistic approach to system design. It is characterized by a set of knowledge sources that can communicate with each other via an area of global memory called a blackboard. Each knowledge source is designed to solve a specific component of the problem that the system is presented with. From a behavior-based prospective, these knowledge sources are generally represented as individual behaviors. One of the key components of any blackboard system is the arbitration mechanism, which is the component of the system that coordinates behavior execution. The architecture developed for this project uses an attention-based arbitration mechanism, as pioneered by Xu and Van Brussel (1997). This approach is based on attention, as observed in humans, where the primary behavior is chosen based on the world state, rather than by some predetermined mechanism. General information about the blackboard architecture, as well as attentional arbitration control is presented in Chapter 3. The individual behaviors used to accomplish the mapping task are described in Chapter 5.

1.4 HARDWARE

The Behavior-Based Blackboard Architecture was tested using a multi-robot system called the Collective. The Collective consists of a team of several autonomous robotic tanks. Each tank is controlled by a network consisting of a Compaq iPAQ 3970, an Acroname Brainstem GP 1.0 microcontroller, and an Acroname Brainstem Moto 1.0 microcontroller. Actuators include two independently driven treads for locomotion and a firing mechanism that launches foam projectiles. Sensory facilities include four sonar ranging sensors, four light intensity sensors, and a compass for orientation information. The iPAQs are Bluetooth enabled, allowing the robots to communicate via the Bluetooth wireless protocol. Details regarding the Collective are presented in Chapter 4.

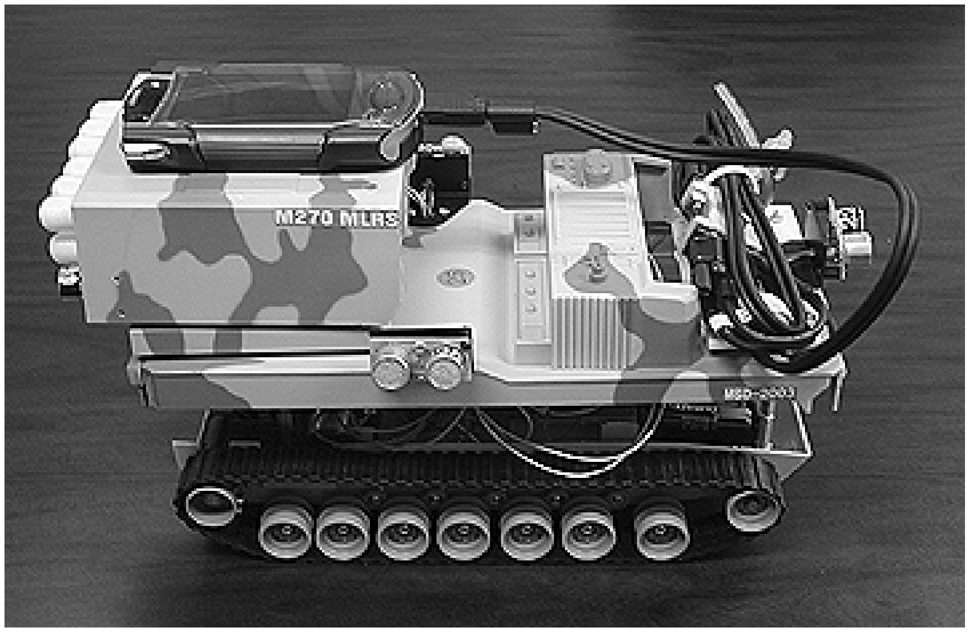


Figure 1: A member of the Collective multi-robot system

In summary, the primary goal of this project was to develop a robust control architecture for multi-robot teams. To test the control architecture, a prototype multi-robot system called the Collective was built. The system itself was given the task of mapping a static office environment as a test. Preliminary results show that the proposed architecture presents specific advantages over other types of control architectures for multi-robot teams.

CHAPTER 2

BEHAVIOR-BASED ROBOTICS

For the last ten years, robotics research has been dominated by various implementations of the behavior-based approach. This approach emphasizes the distributed, cooperative efforts of individual agents (behaviors) to cause the robot to act correctly within a specified environment. As with any scientific field, however, the behavior-based approach arose as a reaction to other lines of thought. Historically, the field of robotics has drawn (although possibly incorrectly) a dichotomy between deliberative and reactive control in robotics. Thus, to fully understand the present approach to robotics, one must understand it in the context of the history of the field as a whole. The following chapter will present a general history of robotics research, paying close attention to the issue of reactive versus deliberative control, which has dominated the literature since the field's inception.

2.1 EARLY EFFORTS IN ROBOTICS

Robotics is a young field, with the first efforts to develop working robotic systems beginning in the 1950's. Surprisingly, early efforts in robotics reflect similar ideas to the behavior-based approach, with the focus relying on simple reflexes to produce correct action within the environment. The primary example of these early efforts is W. Grey Walter's *Machina Speculatrix* (1953) robotic design, which was later implemented as Grey Walter's tortoise. This robot explored its environment, and was capable of returning to a recharging station, designated by a light bulb, to recharge its batteries when they were low.

Control of the tortoise was divided up into simple reactive behaviors that were organized by priority. These behaviors were, “head toward weak light,” “back away from bright light,” and “turn and push.” The *Head toward weak light* behavior caused the robot to move towards a low intensity light source. *Back away from bright light* was an aversive behavior, which caused the robot to move away from a high intensity light source. Finally, the *Turn and push* behavior was an object avoidance behavior. Turn and push had priority over the other two behaviors.

The combination of these simple behaviors, along with an interesting aspect of the hardware used to construct the robot, causes the execution of the fairly complex behavior of exploring and recharging. This is because when the battery is fully charged, the robot perceives the light bulb as a high intensity source of light, causing the tortoise to be repelled from it and move outwards to explore the environment. However, as time passes and the battery power becomes low, the light bulb appears to the robot as a low intensity light source, causing the tortoise to be attracted to it. These characteristics create an oscillating behavior between light attraction and light repulsion based on the status of the battery. That is, complex behavior produced through simple reflexes.

2.2 THE INFLUENCE OF ARTIFICIAL INTELLIGENCE

The fields of Robotics and Artificial Intelligence both began formally around the same time, and in many cases, especially during the formative years of the fields, the primary researchers in one field also held major influence in the other. Thus, not surprisingly, approaches to robotics tended to coincide with approaches to artificial intelligence. In particular, methods to produce intelligence in robots were considered to be merely extensions of the methods used to produce intelligence in computers.

Thus, the early emphasis in artificial intelligence on symbolic representation resulted in most early robotics attempts being modeled around representing the robot's environment through a world model. This approach can be categorized as the *deliberative* approach. That is, robots perceive the environment using sensors, produce a symbolic representation of those perceptions in the form of a world model, and then use that representation to reason about what actions to take next. A good example of the deliberative approach is the Stanford Research Institute's Shakey robot (Nilsson 1969). One of the first mobile robots, Shakey used the STRIPS planning system (Fikes and Nilsson 1971) to develop a model of the world, deliberate on which actions to take, and then create a plan to execute that action. It was early experimental efforts like this that led to the advent of the Hierarchical Architecture, the most prominent control architecture within the deliberative approach.

The Hierarchical Architecture is the traditional architecture for robotics. It was the most popular approach in the 1980s, to the extent that the U.S. government developed a standard robot architecture, called NASREM, which reflected this model (Arkin 1998). The hierarchical model is particularly well suited to static environments, which has made it the primary approach in areas of robotics with this constraint, such as manufacturing.

Arkin (1998) describes the hierarchical model as consisting of four components: sensory processing, world modeling, task decomposition, and value judgment. Each of these components is divided into a hierarchy, with higher levels being more abstract than the lower levels. Thus, the lowest level of the hierarchy consists of the basic control of the robots sensors and actuators, while the highest level has an abstract idea of the task that needs to be accomplished. The intermediary layers help convert this high level plan into primitive controls for the actual robot.

The four main components work in a linear fashion (Brooks 1991a). First, the robot obtains raw data through its sensors; then it proceeds to construct a world model based on this raw data; finally the robot decides what actions to perform based purely on the world model. Thus, this approach does not allow a direct connection between sensation and action. Instead this approach places great importance on the accuracy of the world model.

2.3 THE BEHAVIOR-BASED PARADIGM SHIFT

Like many other approaches developed in the early years of artificial intelligence, deliberative approaches to robotics met with problems of scalability. While researchers obtained good success within constrained environments, such as Minsky's Microworlds (Minsky and Papert 1974), they encountered failure when they tried to expand systems to work within more complex environments. The primary setback associated with these problems had to do with the emphasis on deliberation. While early deliberative systems were able to reason successfully about environments with only a few constrained aspects, trying to reason successfully within more realistic, dynamic environments led to an exponential explosion in problem complexity. Put bluntly, the world is an extremely big problem instance. Thus, researchers found that using a purely deliberative approach resulted in un-acceptably slow reaction times, especially within dynamic environments, environments that change over time. Thus, while robotics found considerable success within constrained environments, such as in manufacturing, roboticists soon found that other means were necessary to begin to deploy successful robotic systems into more complex environments, such as an office building.

The behavior-based approach to robotics arose as a response to the problems associated with centralized control systems. *Behavior-Based Robotics* can be described as a distributed approach, where control is divided up into a number of individual behaviors, whose effects combine to produce a response to the environment. Thus, a behavior-based system lacks centralized control. The distributed nature of such systems creates a need to control the influence of the various behaviors, in particular when behaviors give contrary outputs. In robotics, controlling individual behaviors is called *arbitration* and is accomplished by an *arbitration mechanism*, which decides how the various behavioral outputs will combine to produce the response. It is this mechanism that characterizes the primary difference between all behavior-based approaches.

The most prominent behavior-based control architecture to date is the Subsumption Architecture, first proposed by Rodney Brooks in 1986. Of particular note is that the primary step of world modeling, that is so prominent in the hierarchical approach, is missing under this architecture. In its place are layers of behavioral modules that take sensory inputs and transform them directly into inputs for the robots actuators. Each of these modules is completely independent of the others. If one of the behaviors was used as the sole behavior in the robot, it would execute the behavior perfectly. Thus, developing a robotic system with a subsumption architecture begins by building "a very simple, complete autonomous system, and [testing] it in the real world" (Brooks 1991b, p. 143). Once this module works perfectly, it is added to the system as a behavioral layer. To add more functionality to the robot, more behavior modules can be developed and added to the system as a behavioral layer, without changing any of the underlying layers. The advantage of this is that the already working system is not altered (Brooks 1986).

Brooks' approach to behavior arbitration is called *Subsumption*. A lower-level behavior defers to higher-level behaviors in that higher level behaviors are able to subsume control of the robot's actions. Thus, with this hierarchical control system in place, Brooks asserts that "out of the local chaos of [the behaviors] interactions, there emerges, in the eye of an observer, a coherent pattern of behavior" (Brooks 1991b, p. 144).

2.4 SCALABILITY AND HYBRID APPROACHES

The deliberative and behavior-based approaches in their purest forms lie as opposites on the spectrum of robotic control. While the behavior-based approach has found considerable success with various low-level tasks, questions remain as to whether it can be scaled up to human-level performance. Unlike the basic animal-like reactive behaviors, which behavior-based approaches have focused on, many researchers argue that to obtain human-level performance, symbolic (i.e. deliberative) processing is necessary. Strict behaviorists however, still believe that human level intelligence, like anything else, can be broken down into a collection of behaviors that interact to produce human-level performance. Brooks backs this view by stating "we believe that in principle we have uncovered the fundamental foundation of intelligence" (Brooks 1990).

Few researchers would question the fact that behavior-based reactive control is necessary in a robotic system. However, many also believe that a deliberative component is necessary in order to maximize the effectiveness of a behavior-based system. The resulting *hybrid* systems represent a cross between behavior-based and deliberative approaches. However, the boundary between deliberative and reactive control is still not very well understood, leading to differences in the development of these hybrid systems. The present blackboard architecture, described in the next chapter, not only allows for pure behavior-based reactive control, but is also flexible enough to allow for the addition of deliberative aspects as well.

CHAPTER 3

THE BLACKBOARD ARCHITECTURE

The Collective uses a blackboard architecture to control the actions of each of the robots within the system. This chapter describes the basic workings of the multi-robot blackboard system used in the Collective, as well as the basic blackboard management routines necessary in all implementations of this system regardless of the task the system is designed to accomplish.

The Blackboard Architecture can best be described as opportunistic problem-solving. The term *blackboard* comes from the idea of a number of experts surrounding a physical blackboard working to solve a problem. The entire current problem state is present on the blackboard and each expert makes changes to it according to their expertise (Martin 1996). Thus, given the correct number and type of experts, the problem will eventually be solved through cooperative, although separate, efforts. Formally, a blackboard is a system that "treats problem-solving as an incremental, opportunistic process of assembling a satisfactory configuration of solution elements" (Hayes-Roth 1985).

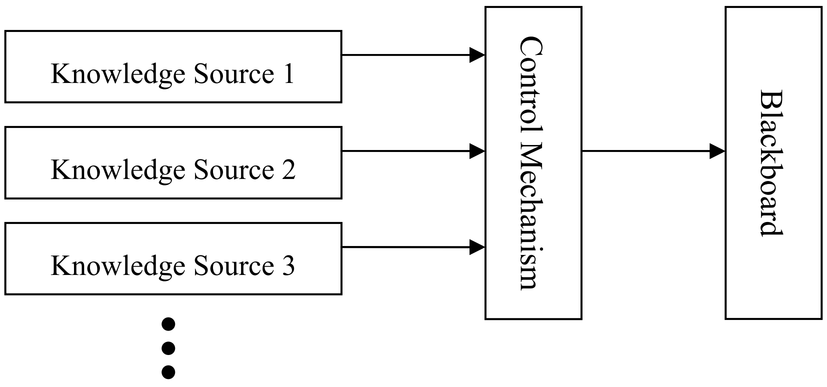


Figure 2: A typical blackboard architecture

Carver and Lesser (1992) characterize a blackboard system as having three main components, a blackboard, a set of knowledge sources, and a control mechanism. The blackboard is a global section of memory that is accessible to all of the knowledge sources. The blackboard contains the data, as well as partial solutions to the problem at hand. In a robotic system, the blackboard could be seen as a representation of the world state, through sensor input, actuator positions, world maps, and other pertinent information.

The set of knowledge sources comprises the problem-solving component of the system. Each of the knowledge sources is tailored to a specific function. Specifically, a knowledge source makes changes to the solutions on the blackboard whenever it is possible to make a contribution. One particularly important aspect of the knowledge sources is that they are completely independent of each other. Any communication between them solely occurs through changes to the blackboard. In a behavior-based robotic system, the knowledge sources consist of independent behavioral modules that respond to the world state on the blackboard by putting new motion commands for the actuators on the blackboard.

The distributed nature of a blackboard system can lead to problems when knowledge sources do not agree. The act of controlling knowledge source execution to prevent conflicts is called arbitration. The control (arbitration) mechanism component of a blackboard system is responsible for managing the execution of the knowledge sources (behaviors) to prevent conflicts between them.

3.1 THE ADVANTAGES OF THE BLACKBOARD ARCHITECTURE

Both the hierarchical and subsumption architectures suffer from specific flaws. The hierarchical approach is criticized as being too reliant on an internal world model. In a dynamic (or even static) environment, this reliance can lead to problems because inaccuracies in the world model lead directly to mistakes in decision-making. In addition, the deliberative aspects of a hierarchical system lead to speed issues, especially in highly dynamic environments. If the system is not fast enough to deal with the changing status of the world then errors will most surely result.

The main criticism against the subsumption architecture is that it fails to meet one of its primary goals of modularity. This is because “high-level behaviors usually need to access the internal states of low-level behaviors” (Xu and Van Brussel 1997, p. 116). By design, the subsumption architecture does not lend itself to creating a world model. Hence, designing a system that does not maintain a world model to create a world model (map) is contradictory and inefficient. Furthermore, the subsumption architecture does not lend itself to multi-robot communication. In fact, the subsumption architecture emphasizes cooperation without communication. However, communication between robots can increase the efficiency of most tasks and especially the mapping task by allowing the robots to avoid mapping the same region twice.

In contrast, a blackboard control architecture allows a robotic system to maintain a model of the world (the blackboard), while still maintaining the ability to react quickly and directly to the world state by creating behavioral modules (knowledge sources), which respond to raw sensory outputs on the blackboard. A blackboard architecture is also strictly modular, in that each behavior can only communicate with other behaviors through the blackboard. This makes designing a blackboard system much easier to accomplish, because individual behaviors can be added to the system without affecting the functionality of the other behaviors that are already present. Finally, the blackboard architecture is particularly well suited to multi-robot communication because all knowledge is localized in a central location, the blackboard. This makes it easy to transmit the blackboard to another robot, in a distributed blackboard such as the Collective, or to place the blackboard in a central location (central server).

3.2 MANAGING BEHAVIORS THROUGH ATTENTION

One of the biggest complications involved in the blackboard architecture is managing conflicts between behaviors. Because of the independence of the knowledge sources, it is possible that two different knowledge sources will respond to the data present on the blackboard and output conflicting information. An example of this in a robotic system would be a conflict between an explore behavior and an object avoidance behavior. On one hand, the explore behavior may be trying to enter unknown territory by moving forward, while on the other, the avoidance behavior may realize that there is an object in front of the robot and may be trying to move backwards. Lack of an adequate control mechanism in this case could lead to problems.

The traditional solution to this problem is to develop a knowledge source execution system based on a conflict-resolution strategy, or arbitration. This is a way of deciding which behavior should take precedence when conflicting with others. Some simple schemes include order based execution, or schedulers, although any type of scheme that results in a clearly dominant knowledge source in any given situation would suffice.

One particularly interesting approach to behavior execution control in robotic systems was presented in Xu and Van Brussel (1997). Unlike earlier systems, which used outside controllers to decide which behavior has precedence, their approach consisted of building the conflict resolution strategy directly into the behaviors using the concept of attention. The concept is based on the idea of attention in humans. For example, when walking through an open area, people normally pay very little attention to avoiding objects in their path. However, as the area becomes more crowded with objects, a person's attention will almost exclusively be centered upon avoiding them. The LiAS robot, presented by Xu and Van Brussel (1997), uses this concept of attention to regulate the execution of behaviors to avoid conflicts. Each behavioral model maintains its own rating of attention, and as the world model changes, so does the attention rating of each module. Each module is given an execution time that is equivalent to its attention rating. Thus, even when a behavior is of low importance it still has some influence on the actions of the robot. However, this effect is overrun by the amount of execution time provided to the behavior with the most attention. As the importance of a behavior increases, so does the amount of execution time it receives. In this way, Xu and Van Brussel were able to create a smooth transition between behaviors, while avoiding the problem of contradicting behaviors with equal weights.

The blackboard architecture in the Collective system uses the same idea of attention in a slightly different way. Rather than take outputs from the behaviors and combine them to create the final output, the Collective decides on a specific behavior to give control to for that cycle. This allows us to create behaviors that output strictly limited movement commands, simplifying the process of localization. In Xu and Van Brussel's approach, the combination of behaviors results in smooth, and efficient motion, but also results in adding a large amount of complexity to the robot localization problem. Given the limited sensory capabilities of the robots in the Collective, along with the necessity of maintaining good location information for cross-robot mapping, highly accurate localization took precedence over efficiency in this project.

The conflict resolution scheme works as follows. Each behavior is allowed to place action commands onto the blackboard. The world state is then ascertained using a series of if-then rules (e.g. If there is an object within 3 inches, then object avoidance has priority), and it is decided which behavior has the prominent attention rating. Only the actions given the highest attention rating are executed. Hence, the overall behavior of the robot becomes less efficient in exchange for greater accuracy in localization.

3.3 BLACKBOARD SYNCRONIZATION

The Collective uses a distributed blackboard approach, where each individual member of the system maintains its own copy of the blackboard. Each individual has immediate access to the knowledge of all individuals, the collective of one mind. This is opposed to another solution, where the blackboard could be stored in a single location, and accessed wirelessly by all the robots. The distributed approach was chosen because it eliminates the need for any central server on which to store the blackboard. This allows the system to be moved from one location to another without the added effort of moving infrastructure as well. Another approach would be to store the blackboard on a single robot within the system, which would act as a "master node" for the entire system, but this was considered to be too error-prone given that the success of the entire system would hinge upon the functionality of one robot. The present approach allows any member of the system to fail without compromising the functionality of the system. If a robot fails, the Collective merely continues on without the added input from that member.

However, with multiple copies of the blackboard existing at the same time, a procedure is necessary to make sure that these copies are synchronized without any loss of information. While the blackboard itself is maintained in main memory, before synchronization, data that need to be transferred to the rest of the team are placed in a text file. This feature does not lead to any loss in performance however since the file processing does not occur on a disk, but rather in main memory, since everything is stored in main memory on an iPAQ. This approach simplifies the synchronization procedure by allowing the entire blackboard file to be sent to all the other members of the system using SendFile.exe by BTAccess. This approach places the blackboard management burden on file processing procedures, which are well documented, rather than on the Bluetooth interface.

The process for synchronizing all the blackboards in the system proceeds using a *token-ring* approach (IEEE 2000). That is, transfer of files between the robots proceeds in the same order every time, with each robot waiting their turn to transfer. Once all the files have been received, the blackboards are synchronized together. The details of both the transfer procedure and the synchronization procedure will be expanded upon in the next section.

3.4 BLACKBOARD MANAGEMENT ROUTINES

The following section describes the individual routines for managing the blackboard within the Collective. These routines are the key components of the system in that they remain part of the system, no matter what the individual task is. Thus, these procedures represent the interface between the individual behaviors of the system, which change depending on the task at hand, and the outside world. These basic routines include placing something on the blackboard, taking something off, reading information from the blackboard, synchronizing two blackboards, and transferring blackboards between robots. See Appendix A for the actual code implementations of each of these functions.

Before beginning the outline of the blackboard processing routines, it is important to understand the basic format of each robot's blackboard. The entire blackboard is stored in a single file. All data within the file has a label that specifies what type it is (e.g. "xy" followed by a number signifies an individual coordinate). In addition, the data are partitioned into individual sections of each type for ease in processing.

Placing a piece of information begins by searching the blackboard for the correct section, and then looking to see that information is not being repeated (e.g. that a coordinate has not already been explored). If the information is already on the blackboard, it is replaced by the new information, making the blackboard as current as possible. Otherwise, the information is inserted into the file in the correct section.

Retrieving information from the blackboard occurs through removing or reading it from the file. The remove procedure eliminates all information associated with the tag (e.g. a specific coordinate) given. Reading information differs in that the information is not removed from the file. Both procedures return the information associated with the tag for processing.

The blackboard file transfer procedure is illustrated in Figure 3, which shows the transfer procedure in a collective of three robots. Each robot's blackboard has a distinct file name associated with it. When engaged in the process, the robots can be in one of two states, either sending the file or waiting. Only one robot is in the sending state at a time. Robot 1 begins the process by sending its blackboard to Robot 2. Both robots 2 and 3 are in the wait state during this process. When robot 2 checks and finds that robot 1's blackboard has been received, it proceeds to send its file to robot 3 while the other robots enter the wait state. When robot 3 receives the file, it does the final synchronization and sends the file back to both robots 1 and 2. If for some reason robot 1 were to become incapacitated, robot 2 would overcome this problem by automatically sending the file after a certain amount of time had passed without receiving the file. The same goes for any robot in the system. If a robot does not receive the correct file after an extended period of time, the robot would assume robot failure and proceed normally. Thus, the procedure is fault tolerant by allowing for the failure of any robot within the system. Once all the files have been transferred, each robot proceeds to synchronize the files with the synchronization procedure.

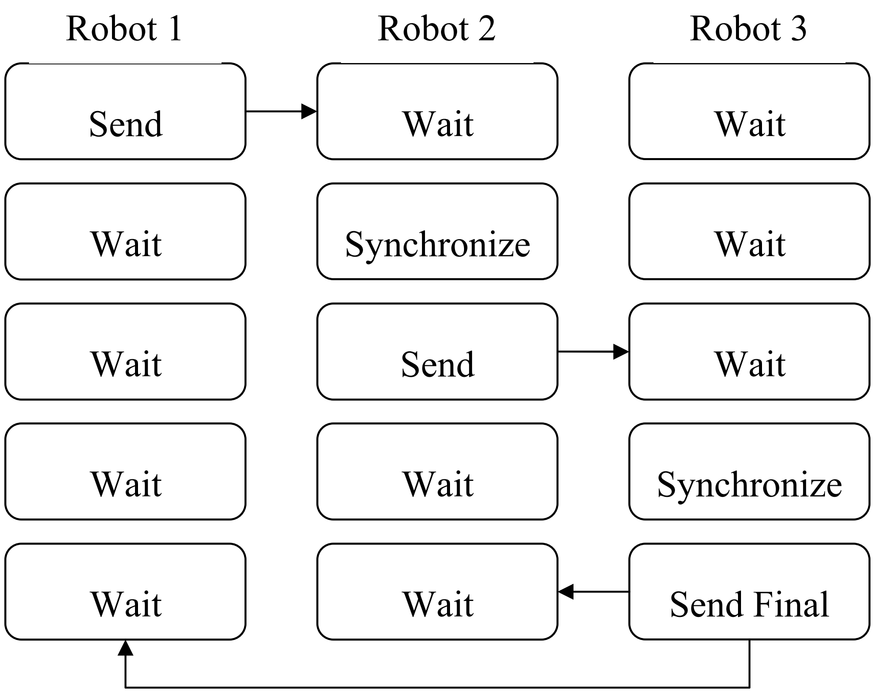


Figure 3: The blackboard transfer and synchronization procedure

When synchronizing two blackboards, a robot's own blackboard is considered the master copy. Thus, any discrepancies between blackboards results in the robot keeping its own information, and not the other. This specification, along with the way the blackboard transfer procedure is structured, results in a global blackboard that is consistent for all robots. This is done by simply not adding any information to the blackboard that already exists.

CHAPTER 4

HARDWARE DESIGN

4.1 "THE COLLECTIVE" MULTI-ROBOT SYSTEM

The multi-robot system used in this project consists of a collection of robotic tanks, first described in McClain et al. (2004), called *The Collective* after the Star Trek term for the Borg, a race of aliens whose minds are interconnected at all times. The system possesses capabilities for moving about and exploring environments, along with the ability to engage in various team-based military-like tasks (e.g. search and destroy). The Collective is designed for small-scale applications and is ideal for proof-of-concept type projects for multi-robot systems. It is a custom system, created by radically altering several commercially available products and is intended for indoor use only.

4.2 THE ROBOTIC CHASSIS

Each member of the Collective is a Motorworks Missile Launcher radio-controlled tank that has been converted into an autonomous robotic vehicle. Each tank runs on a standard 9.6 volt battery, has two treads driven by separate DC motors, and also has a rotating turret with a built in projectile launcher. The projectile launcher is capable of individually firing 12 foam missiles, allowing for multiple attack sequences without reloading. In addition, it is operated by a small DC motor, allowing for simple programmatic operation. The tank is able to climb up to a 30° incline, as well as cross small ditches and other obstacles. All of the robots within the system possess the same physical modifications. A diagram of the entire custom control system for each robot in the Collective can be seen in Figure 4. Figure 5 depicts the original Motorworks Missile Launcher and a completely modified member of the Collective.

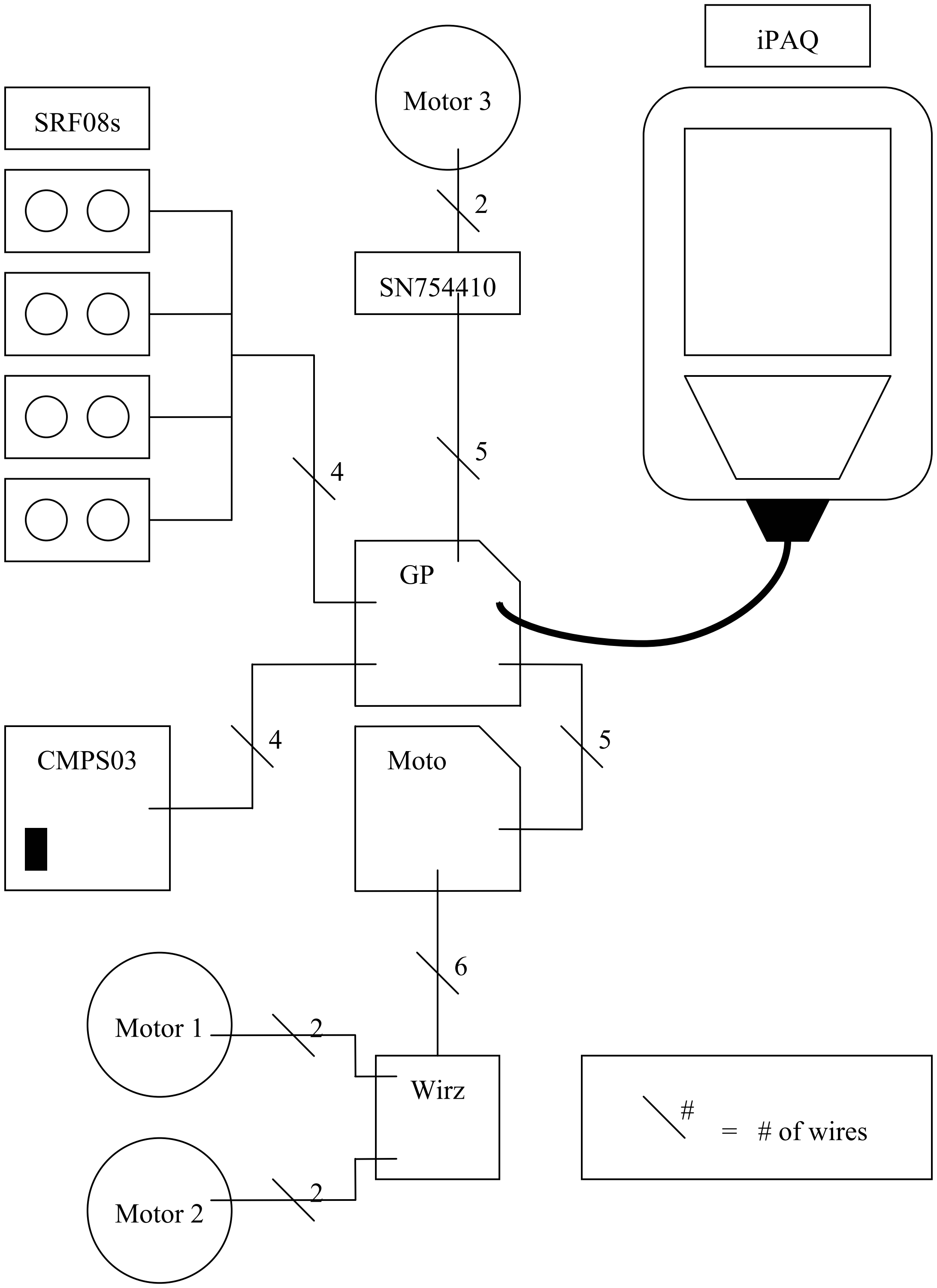


Figure 4: The hardware architecture for a member of the Collective

The control hardware for each member of the Collective, first introduced in Barnhard et al. (2004), consisted of an iPAQ/Brainstem network. Each robot is outfitted with an Acroname Brainstem micro-controller network for low-level sensor and motor control. The main controller for each robot is a Bluetooth enabled Compaq iPAQ 3970 running Windows CE (Pocket PC 2002). Both speed and directional control are provided for each tread independently, and firing control is available for the projectile launcher. With respect to sensory capabilities, each robot is provided with four Acroname SRF08 ultrasonic range sensors mounted at right-angles to each other on the chassis, as well as an Acroname CMPS03 magnetic compass sensor for orientation information. The details of each of these components will be provided in subsequent sections.

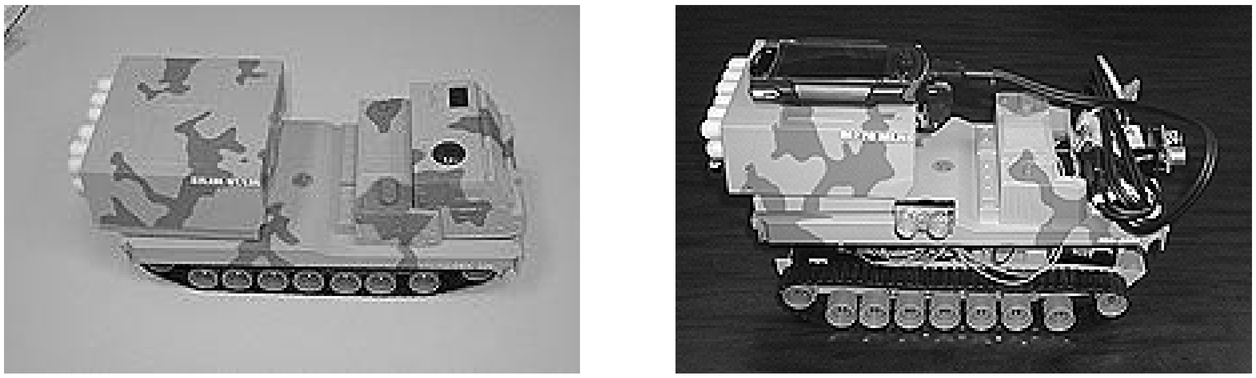


Figure 5: The Motorworks Missile Launcher before and after modification

4.3 SENSORS AND ACTUATORS

The DC motors that drive the treads of the tanks are controlled by the Brainstem controller network via a Wirz 203 Motor Driver Board. This board provides a simple interface to a Texas Instruments SN754410 Quadruple H-Bridge Driver, which is an integrated circuit (IC) consisting of four half H-bridges which can be combined together to create two full H-bridges. An H-bridge (so called because the circuit diagram looks like a letter H) is a specialized circuit designed to allow a motor that runs on a separate power supply to be controlled via standard +5v logic inputs (see Figure 6 for a diagram of a typical H-bridge circuit). In particular, a full H-bridge allows for a motor to be run in both directions by reversing the direction of current-flow through the motor via logic signals.

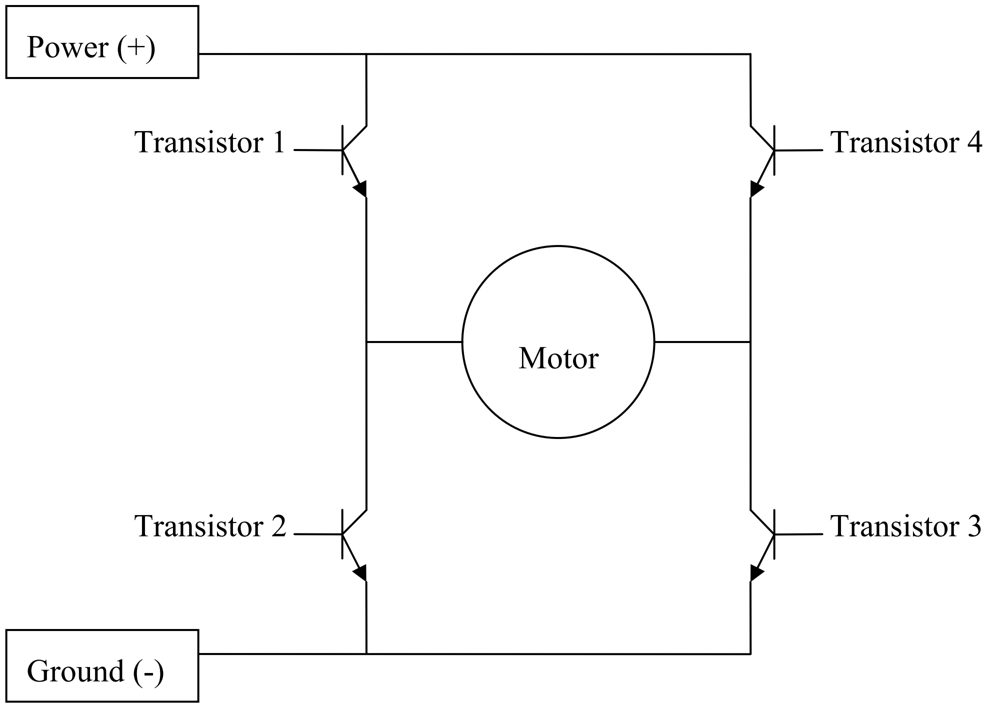


Figure 6: A typical H-bridge circuit

Variable speed is accomplished using Pulse-Width Modulation (PWM) provided by the Brainstem Moto 1.0 microcontroller described in the next section. PWM is a technique that accomplishes variable speed in motors by alternating the motor between full-on and full-off states at very high frequencies. The time which the motor is in a full-on state is called the *on-pulse*. The time duration for an on-pulse is called the *pulse-width*. Speed variation is achieved by lengthening the pulse-width for higher speeds, and shortening it for lower speeds. In PWM, the duration that the motor remains in a full-off state remains constant. This method proves to be more reliable and easier to implement than other approaches, such as raising and lowering the supplied voltage to the motor. The one problem with this approach is that the rapid transition between on and off states requires specialized transistors which can handle the high current necessary to run the motor, as well as the stress created by the rapid transitions between full-on and full-off states. These high-frequency transitions produce quite a bit of heat within the system, creating the need in many cases for a heat sink, which is generally an electrically isolated piece of metal that is attached to the circuit to conduct heat away from the board allowing it to cool faster. Figure 7 illustrates the complete wiring diagram for the Moto, Wirz 203, and two motor combination. Notice that two power supplies are require for the system to work, a logic supply and a supply to power the motors (in this case a 9.6v battery). The Wirz 203 also allows for directional and PWM control for each motor, providing inputs for each.

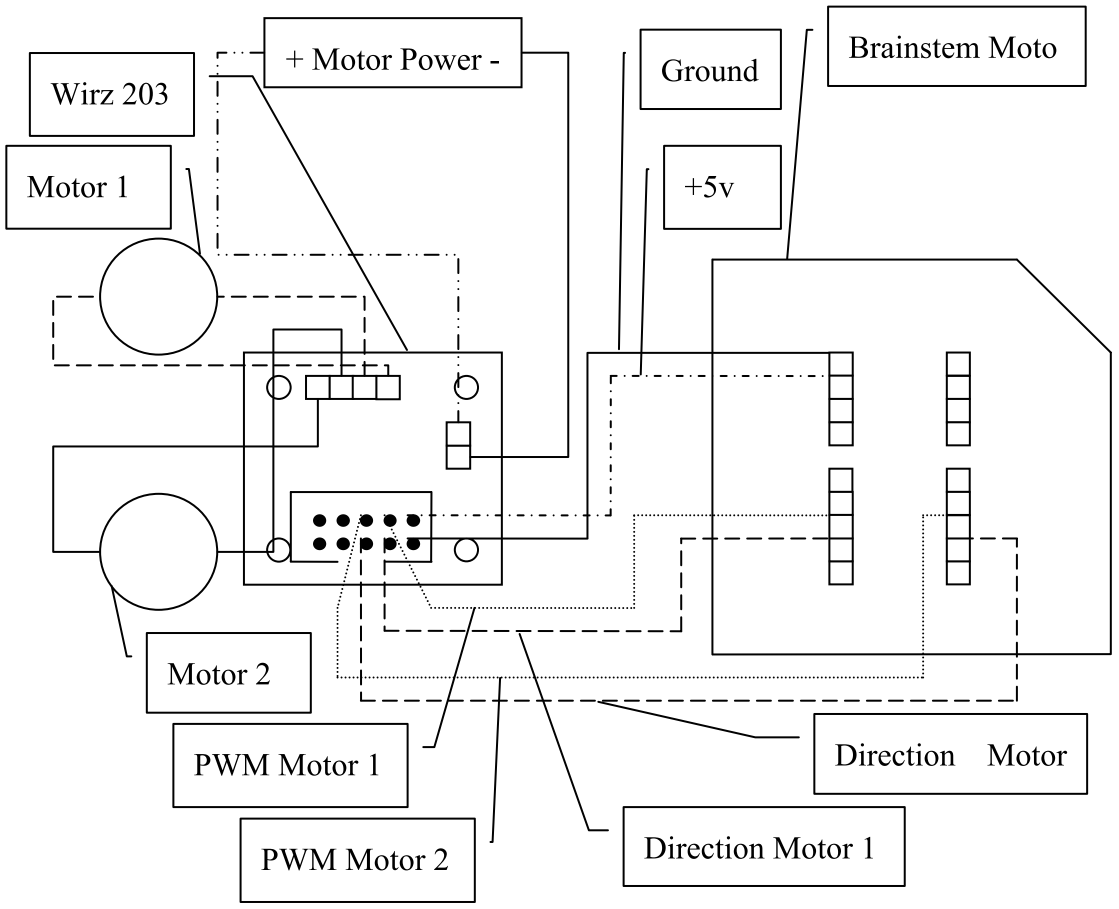


Figure 7: The Brainstem Moto interface to the Wirz 203 Motor Driver

The projectile launcher is controlled by a much simpler interface to the SN754410 (illustrated in Figure 8). The interface is simpler because neither speed control, nor directional control is necessary to control the firing mechanism. A projectile is launched simply by turning the power on and off to the motor in one direction. Each power-cycle causes only one projectile to be launched at a time.

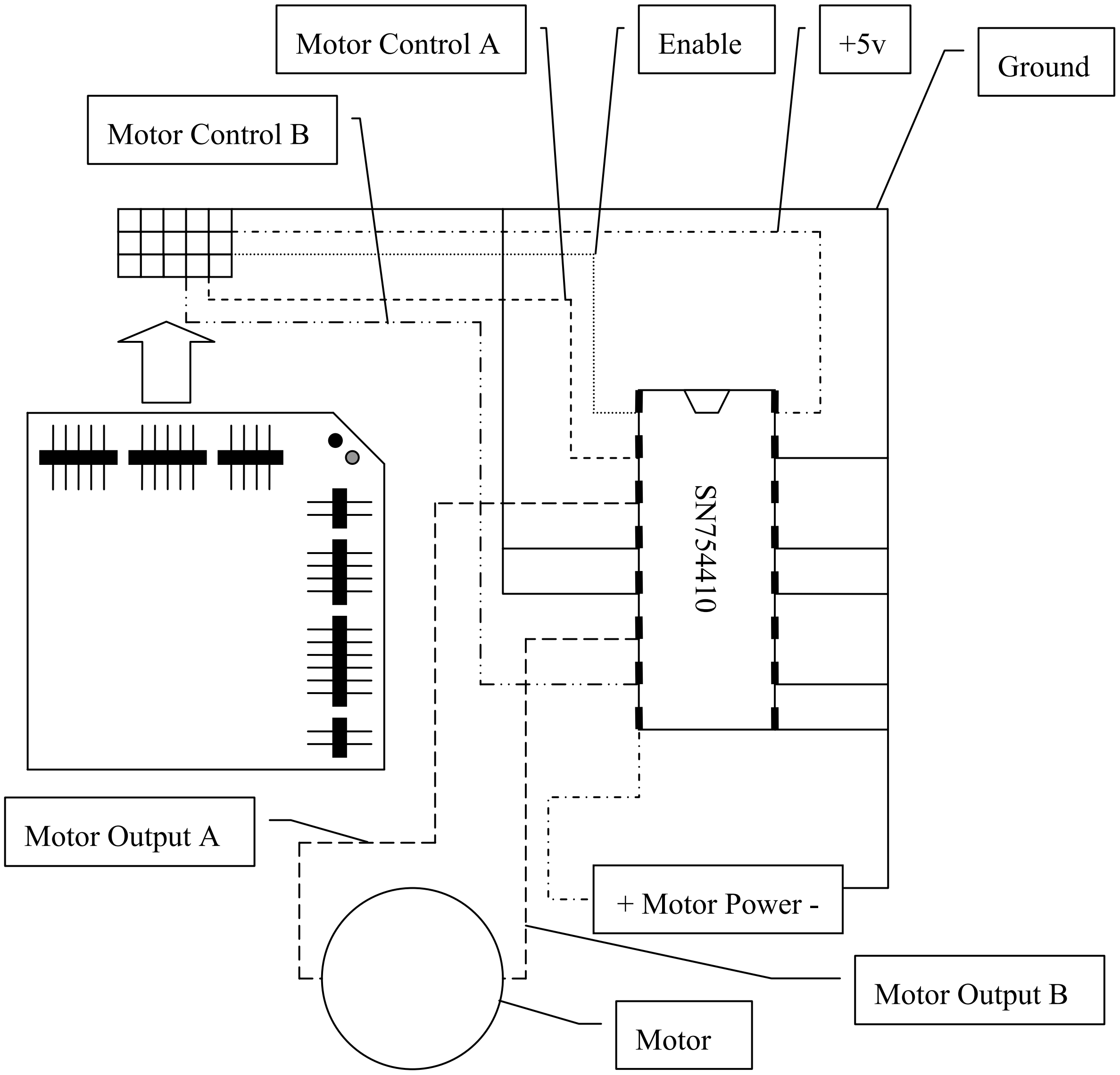


Figure 8: The Firing Mechanism control hardware

Each tank uses four Acroname SRF08 ultrasonic range sensors. The SRF08 is actually two sensors in one, with both the ultrasonic ranging component, as well as a light intensity sensor in the form of a photo resistor. It can detect objects up to approximately 20 feet away and can obtain incremental measurements in centimeters, inches, and feet. Range is determined by sending out an ultrasonic *ping* via one speaker, setting a timer, then waiting for the ping to return to the receiver, whereupon the timer is stopped. From there, a calculation is performed to translate the time that the sound traveled into distance. The SRF08 can be connected directly to the Inter Integrated Circuit (IIC) Bus on the Acroname Brainstem, where both range and light intensity readings can be accessed at specific addresses. Figure 9 shows the interface between the Brainstem and the SRF08. The IIC Bus is a bus with two communication lines (SCL and SDA) designed to allow small microcontrollers to access peripheral devices by address. A single Brainstem can support a large number of SRF08s by assigning each one a unique address and then "piggy-backing" them on the IIC Bus line.

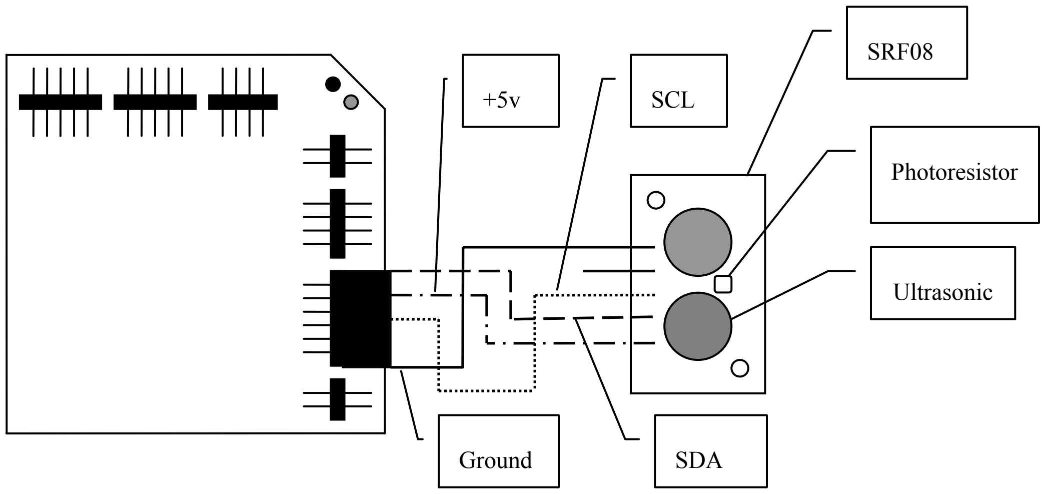


Figure 9: The Brainstem/SRF08 interface

In addition to the SRF08s, each tank is also outfitted with an Acroname CMPS03 magnetic compass sensor (interfaced with the IIC Bus as well. See Figure 10). This sensor allows the robot to determine its orientation and is advertised as being accurate to 1°, however in practice the sensor is really only accurate to about 5°. In addition, the CMPS03 is quite sensitive to electromagnetic interference. This means that there must not be any magnetic fields present when operating the sensor in order to get an accurate reading. In particular, it was found that the iPAQ controllers have a strong enough electromagnetic field to influence the sensor's readings within 2 to 3 inches of the device. Thus, it was necessary to place the compass away from all other electrical devices on the robot to get an accurate reading. Like the SRF08, the CMPS03 also interfaces directly to the Brainstem controller via the IIC Bus, and its readings are accessed by address.

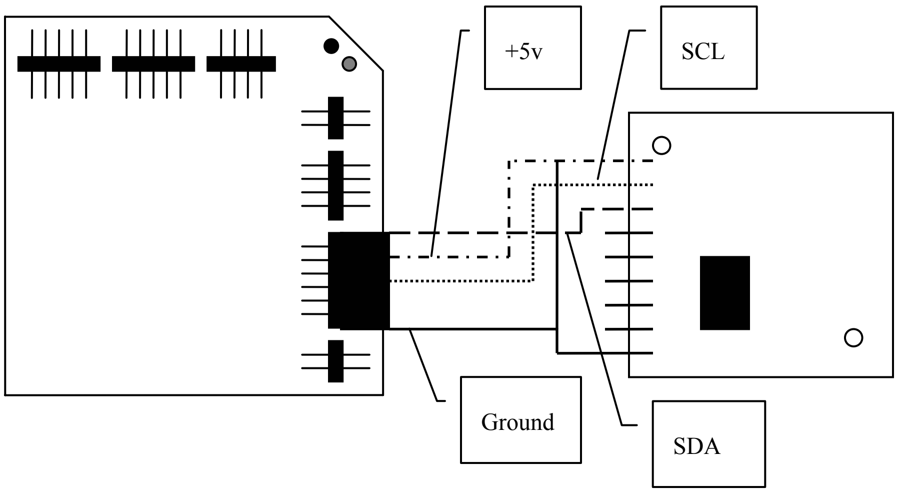


Figure 10: The Brainstem/CMPS03 interface

4.4 THE CONTROLLER NETWORK

The infrastructure for controlling the sensors and actuators for each robot consists of a network of three different controllers, a Brainstem GP 1.0 microcontroller, a Brainstem Moto 1.0 microcontroller, and a Compaq iPAQ 3970 running Windows CE (Pocket PC 2002) acting as the primary controller. The connection interface for this network is shown in Figure 11. This approach provides greater flexibility in that it allows for the greater computing power provided by the iPAQ, while still maintaining the ability to interface hardware into the system with ease using the Brainstem's interface ports. In addition, because there are multiple controllers within the system, it becomes possible to do multiple tasks at the same time using the different controllers. Another advantage is that it is quite simple to expand the capabilities of the system simply by adding more Brainstems to the network.

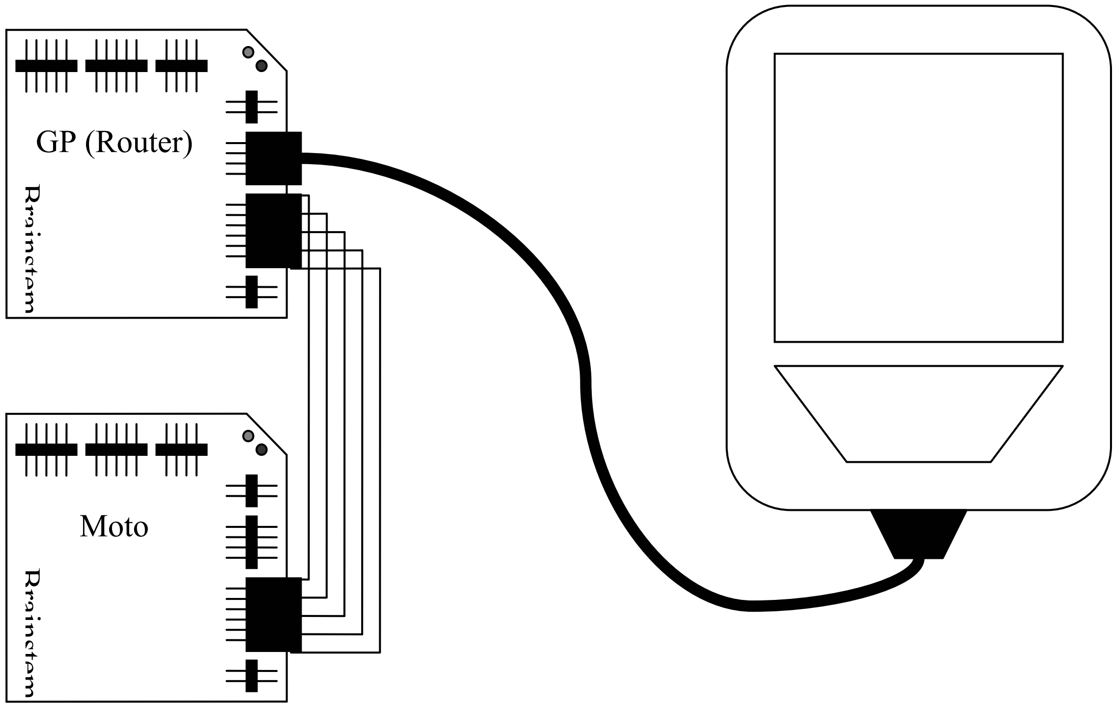


Figure 11: The iPAQ/Brainstem Controller Network

The Brainstem GP 1.0 is a microcontroller with a 40 MHz RISC processor, 1 MBit IIC Bus, five digital input/outputs, and four high resolution servo outputs. It is useful for controlling basic robotic functions. The GP is used in the present configuration as the main controller for all the sensors and as the controller for the projectile launcher. The Brainstem Moto is similar to the GP except that it has been modified to allow for high-frequency PWM. The present configuration uses the Moto as the controller for the primary drive motors. The two brainstems are networked together via the IIC Bus with the GP as the router. The *router* (GP) is connected to the iPAQ. All instructions to the Brainstem network are channeled through the router and delivered to the correct Brainstem by address.

The iPAQ 3970 acts as the primary controller for the system. It is characterized by high speed and a large memory on an extremely small footprint (about the size of an adult hand). In addition, the iPAQs used in the Collective are Bluetooth enabled, allowing the possibility of direct (peer-to-peer) multi-robot communication.

Instruction processing within each robot in the system proceeds in the following way. As the primary controller, the iPAQ makes all the decisions about which actions are necessary to take. Upon deciding on an action which requires access to the sensors or actuators, the iPAQ sends an addressed instruction packet to the router in the Brainstem network. The router then examines the address in the packet and forwards it to the correct Brainstem. Upon receipt, the packet is unpacked, executed, and upon completion, a message is sent back to the iPAQ indicating success or failure. Meanwhile, while the iPAQ is waiting for confirmation that the instruction succeeded, it is free to proceed with other tasks.

4.5 THE BLUETOOTH WIRELESS PROTOCOL

Communication between members of the Collective takes place using the Bluetooth wireless protocol. Invented in 1994 by L.M. Ericsson, Bluetooth was named after Harold Bluetooth I, the king of Denmark and Norway in the middle 900s. Intended to create a short-range radio link between electronic devices, Bluetooth is primarily used as a wireless desktop solution (e.g. wireless keyboards) because of its limited range of approximately 30 feet. Bluetooth is characterized by robustness, low complexity, low power, high data transmission speed, security, and low cost. Bluetooth has both advantages and limitations when compared with the more traditional 802.11*x* approach. In their design of a robotic 802.11*x* relay system, Nguyen et al. (2003) found that it was necessary to design a new 802.11*x* wireless modem, primarily because ordinary, proprietary modems were too large to be practical for robotics. However, the use of Bluetooth eliminates the problem of size restriction, since radios for this standard are small enough to fit into an iPAQ, which is the size of an adult hand.

Another aspect of the 802.11*x* approach is the need for infrastructure. In Gerkey et al. (2001) and Nguyen et al. (2003), the infrastructure took the form of a control server. While direct communication is possible, Nguyen et al. (2003) described how it is quite difficult. In general, it is at least necessary for a wireless router to be present. In contrast to this, Bluetooth makes it very straight forward to establish a direct serial link between two Bluetooth enabled devices, making it possible to avoid extra infrastructure related to the communication medium. Thus, it is possible to move the robot system from one location to another without the added effort of transferring other infrastructure such as routers. Another distinct advantage is a Bluetooth enabled robot's ability to communicate with any other Bluetooth device, including printers, and wireless phones, to name a few.

A major disadvantage of Bluetooth is its limited range of approximately 30 feet, as opposed to the 802.11*x* typical range of about 300 feet. While this was not an issue with the current project because of the small size of the task area, when working in larger arenas, Bluetooth type communication would require the robots to remain much closer together. However, it is quite possible to extend the range of the system by using robots to relay messages between one another, much in the same way as was described in Nguyen et al. (2003).

The Bluetooth Protocol stack in Windows CE (WinCE) 3.0 is from a company called Widcomm. A cost-effective way to go about Bluetooth development in WinCE 3.0 is to download the BTAccess libraries from High Point Software. These libraries allow a programmer direct access to the Bluetooth stack via C++ function calls. The demonstration version of this package is free to download. High Point also offers several Bluetooth applications that perform specific functions, such as SendFile.exe, a file transfer program. These applications also have demonstration versions and can easily be called from C++ by creating a process with the correct parameters. SendFile.exe is the Bluetooth product used in the present system for ease of implementation (see Figure 12 for a sample call to SendFile.exe). A third party solution is needed for Bluetooth development since software development capabilities did not become integrated within WinCE until version 4.0 (.NET), which presently does not appear to be available even on the newest versions of the iPAQ (as of March, 2004).

```
CString errmsg;
PROCESS_INFORMATION      pi;
DWORD rc = CreateProcess (_T("\\Windows\\Start Menu\\BTSendFile.exe"),
                                        _T("dev=00:02:C7:17:50:8F file=\\temp\\test.txt"),
                                        NULL, NULL, FALSE, 0, NULL, NULL, NULL, &pi);
errmsg = "Error: There as a problem launching BTSendFile.exe!";
if (rc == 0){
          AfxMessageBox(errmsg);
            return;  //Error launching program
            }

//Wait for it to complete
DWORD dwExitCode;
while (::GetExitCodeProcess (pi.hProcess,  &dwExitCode) )
{
          if (dwExitCode == STILL_ACTIVE)
                     Sleep(100);
          else if (dwExitCode == 0){
                    errmsg = "BTSendFile ended normally";
                    AfxMessageBox(errmsg);
                      break;
          }
          else if (dwExitCode == 1){
                    errmsg = "Error: Syntax Error!";
                    AfxMessageBox(errmsg);
                      break;
          }
          //use exit-codes to check for errors by inserting more cases here
          // 0 = BTSendFile ended normally
          // 1 = Error: Syntax Error!
          // 2 = Error: Unable to connect to the Bluetooth Stack!
          // 3 = Error: Device not found!
          // 4 = Error: File transfer service not available!
          // 5 = Nothing
          // 6 = Error: Problem during connection or while sending file!
          // 7 = Error: Timed out waiting for connection or file transfer to complete!
          // 8 = Error: Bluetooth radio was turned off!
          // 9 = Error: File not found!
}
//Close process
CloseHandle(pi.hThread);
CloseHandle(pi.hProcess);
```

Figure12: A sample call to BTSendFile.exe

CHAPTER 5

BEHAVIORS

So far, the physical structure of the robots, as well as the fundamental software components of the blackboard system have been described. These components can be viewed as the means to completing the tasks presented to the system. The actual intelligence to utilize these components, however, is present in individual behaviors like the ones presented in this chapter. These behaviors contain the intelligence that is task-specific, that is the behaviors are the only thing that need to be changed when transitioning the Collective from one task to another. Even the relatively simple behaviors, such as sensor management, are task-specific and could be left out of the system under certain circumstances (e.g. sensor management could be left out if the task did not require the system to know anything about the environment). Hence, this chapter describes the behaviors necessary to accomplish the Mapping Task, which include sensor management, actuator management, object recognition, and mapping.

Before describing the individual behaviors, the overall approach to the Mapping Task must be understood, making the division of this approach into behavioral subcomponents easier to understand. The general mapping algorithm is a wall-following algorithm. Thus, when performed by a single robot, the robot follows the outermost wall of the environment (an assumption is made that the robot is placed along the outside wall) scanning the inside of the environment (see Figure 13). In this way, given a few exceptions, the robot can obtain a complete map of the environment. There are two exceptions to this, a room-within-a-room scenario (see Figure 14), and limited sensor range (see Figure 15). The room-within-a-room exception is handled by mapping the entire perimeter of the environment, identifying the entrance to a possible room, going to the entrance, and then mapping the interior of that room by following the smaller room's perimeter. Thus, the algorithm becomes recursive. Limited sensor range can lead to unexplored areas within the center of the room. This exception is handled by exploring into the center of the room until all unknown squares have been accounted for. Using two robots moving in opposite directions on the perimeter (see Figure 16), the exploration time can be cut in half, making the use of multiple robots advantageous.

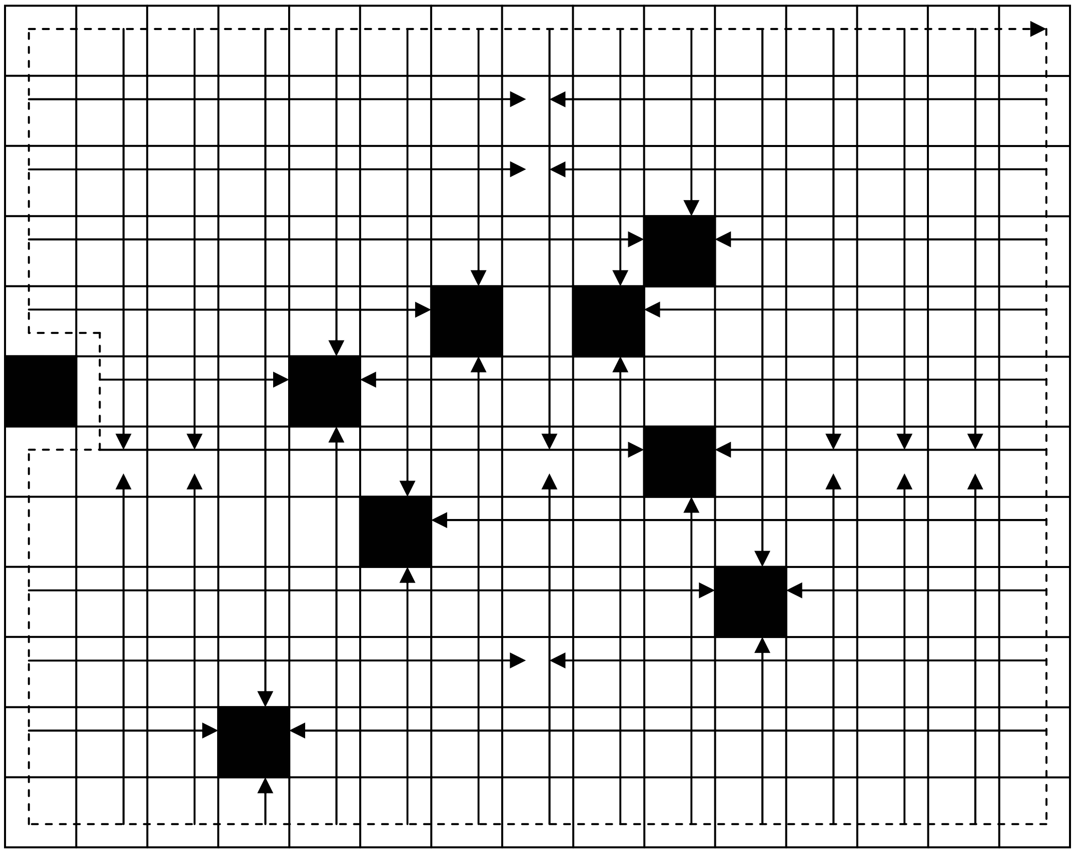


Figure 13: The perimeter mapping algorithm

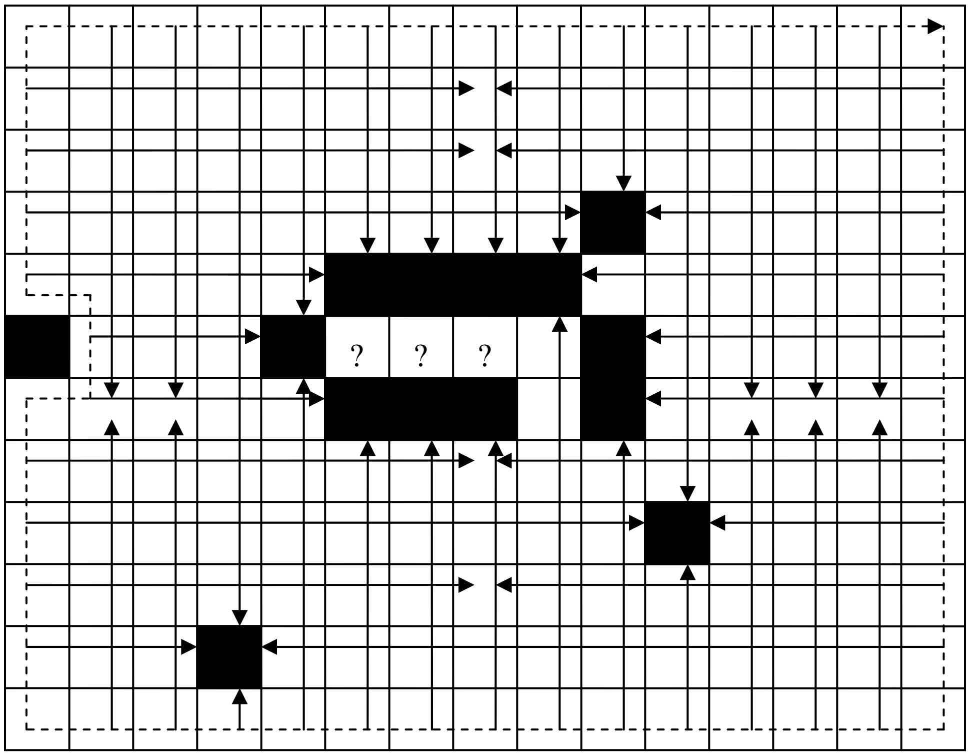


Figure 14: A room-within-a-room exception

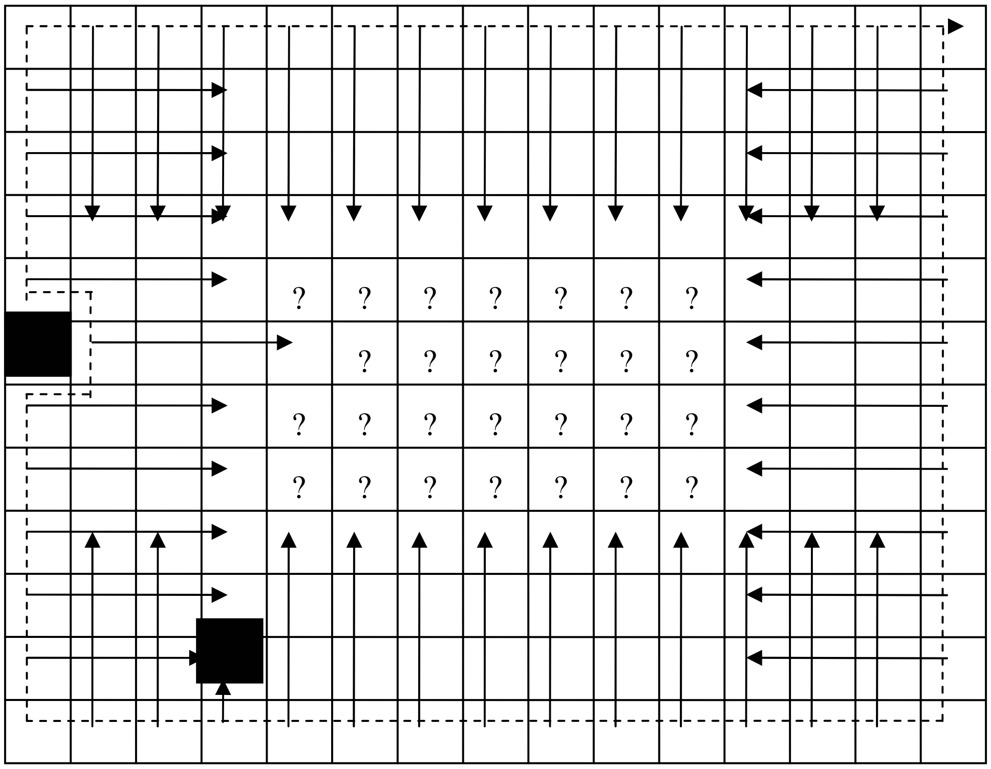


Figure 15: A limited range exception

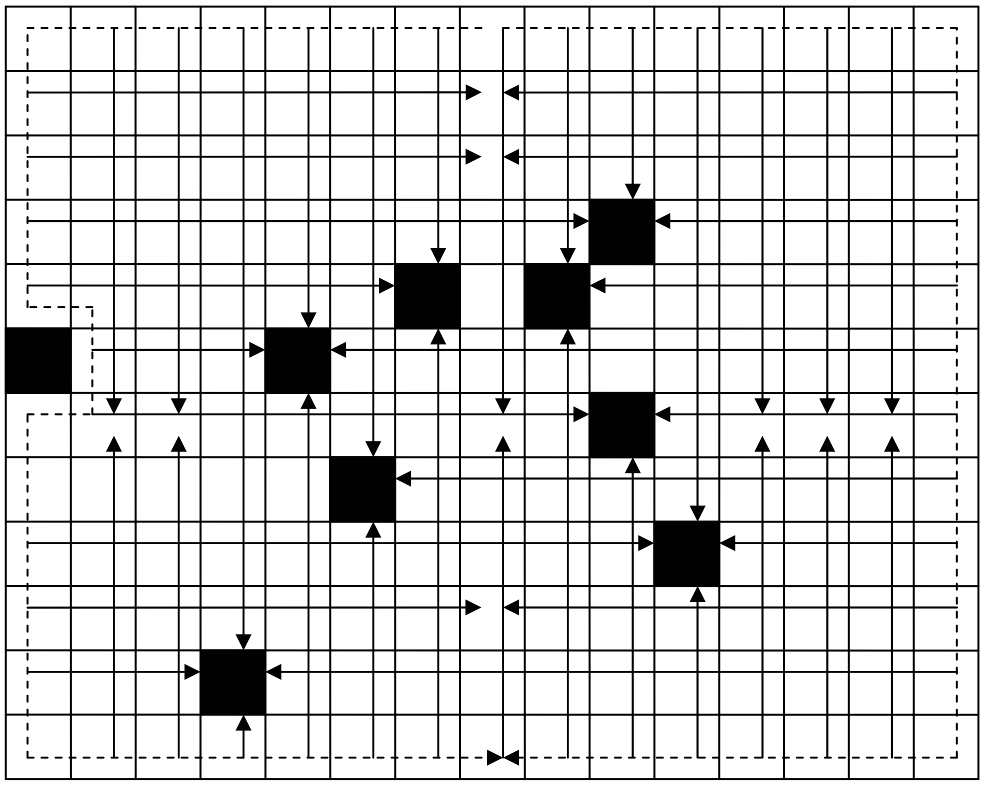


Figure 16: Using multiple robots to reduce search time

5.1 SENSOR MANAGEMENT

The sensor management behavior ascertains the environmental state of the robot. This behavior is comparable to the basic process in biological systems of perceiving the environment using nerves. This behavior has no attention rating, and thus is executed on every pass through the behaviors. It is the first behavior to be executed, so that the data it produces are available for the use of all the other behaviors. The behavior begins by polling each of the four SRF08 sensors and the CMPS03 sensor. These measurements are placed on the blackboard after being formatted. The distance measurements are formatted to inches, the compass readings to one of the four cardinal directions, and the light intensity readings are left in pure number format. The behavior does not try to ascertain objects within the environment; it merely places the measurements on the blackboard. Discerning what those measurements mean is left to other behaviors.

5.2 ACTUATOR MANAGEMENT

The actuator management behavior manages all of the actions performed by the robot. It is comparable to the biological process of sending signals to a muscle to start a contraction. In the present system, there are only three actions the robot can perform, move forward one cell, turn 90˚ to the right, or turn 90˚ to the left. The actions are performed without regard to the affect it may have on the robot, that is if there is an object in front of the robot and the command is "move forward", the robot will move forward regardless of whether this action may cause a collision (currently the system throws up an error condition in this case, preventing damage to the robot). Ensuring that a collision does not occur is left to the behavior which places the movement command on the blackboard. This behavior monitors sensor input independently of the sensor management routine in order to make sure its actions are correct (e.g. the compass is monitored when making a turn to make sure that the robot has really turned 90˚). The actuator management behavior also receives no attention rating, meaning it is used on every pass through the behaviors.

5.3 OBJECT RECOGNITION

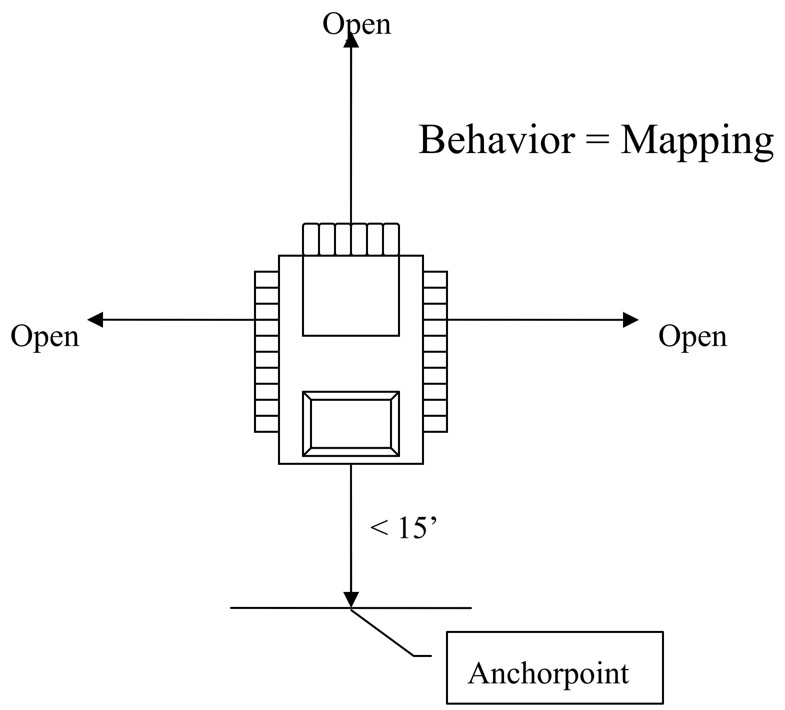
The object recognition behavior is responsible for ensuring the positional information of the robot is accurate. Each robot partitions the environment up into a two dimensional grid (x and y coordinates), where each cell can either be filled or empty. This grid represents the world model of the robot and is used to create the map once the robots have completed exploring the environment. Each cell within the grid is a sixteen inch square, so that the robot can fit within each cell. A robot can only move at right angles within the grid, that is it can only move along the x and y axis exclusively and cannot move along both axes at the same time. This simplifies the localization process by not requiring the system to deal with predicting which cell it is in after moving at an angle, a process which is error prone due to inaccuracies in the compass sensor in particular. A robot maintains its position within the grid using a reference point called an anchorpoint. An *anchorpoint* is an object along the axis that the robot is currently moving on that allows it to determine how far it has moved along that axis. A good description of an anchorpoint is a rope tied along an axis in the grid with knots tied in it every inch. The robot moves from knot to knot and is thus able to tell where it is in comparison to where it was.

The object recognition behavior receives attention only when it is necessary to establish a new anchorpoint. This results from two scenarios, either the robot has encountered an object along the axis which it is moving, or the robot has reached a distance from the anchorpoint such that to move any farther would put it out of the twenty foot range of the sonar range sensors. In either case, a change of direction is necessary, making the object recognition behavior responsible for all directional changes in the system.

Finding a new anchorpoint can proceed in two ways depending on the current situation of the robot. If there is an object in front of the robot, then the robot first checks for an anchorpoint 90˚ to the left and right (i.e. along the other axis). If there is one available, then the robot decides which direction it should turn based on which direction needs exploring. It then places a command on the blackboard to make a 90˚ turn in that direction, and the behavior terminates with the new anchorpoint established. On the other hand, if there is not an object directly in front of the robot, then the behavior first tries to establish an anchorpoint in front of the robot. It then tries the opposite axis. Finally, it will go back the way it came. Thus, the behavior avoids going back in the same direction it came as much as possible. When a decision must be made about which direction to take, the direction is chosen which has the closest unknown area on the map.

5.4 MAPPING

The mapping behavior processes the sensory inputs given by the sensor management behavior and uses them to produce a map of the environment on the blackboard. The behavior takes the sensor readings, which are measurements of how far an object is from the robot on a particular axis, and determines which cells in the grid it can fill in. For example, if the robot is facing north along the y-axis, and the right-hand sensor reads that there is an object one foot away, then the mapping behavior would label the cell that is one foot from the robot's current position to be filled and all the cells in between as empty. Once all of the sensor inputs have been processed in this fashion, the behavior places a "move forward" command on the blackboard for the actuator management behavior to execute. The mapping behavior only gets attention when there is not an object in the cell in front of the robot.



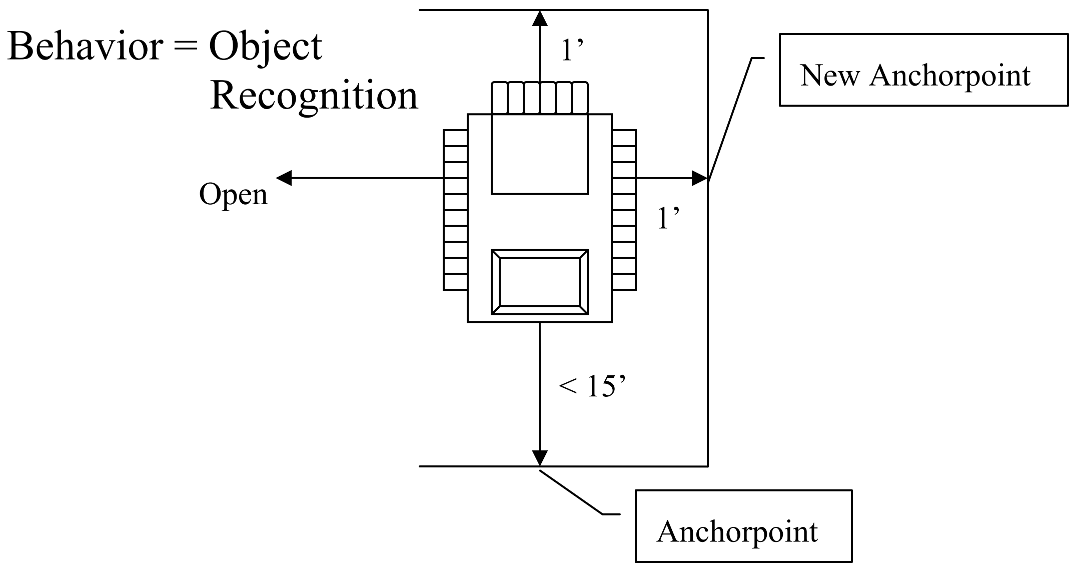


Figure 17: Two possible world states and the resulting behaviors

CHAPTER 6

EXPERIMENTAL RESULTS AND CONCLUSION

6.1 EXPERIMENTAL RESULTS

The Collective was tested using two robots within a tabletop testing environment that was 8 x 4 feet. Despite its small size, the system could be scaled up to much larger environments without very much alteration. Objects within the environment consisted of cardboard boxes. Figure 18 shows one configuration for the environment with a single object within it. Figure 19 shows both robots in their starting positions before the task begins. The robots started facing in opposite directions from one another. Each robot followed the wall in opposite directions while taking a map of the environment. Figure 20 gives before and after shots of the object recognition behavior being executed, causing the robot to turn. The task was completed when both robots were back next to each other and trying to enter the same cell. Figure 21 shows the resulting map of the environment after the Collective explored it. Notice that the map does not appear to describe the environment correctly, because the object appears to be next to the side wall. This is because the Collective maps the environment by labeling areas within the environment that one of the robots can occupy. In this particular environment, a robot cannot occupy the space between the object and the wall, since the space is not big enough. Thus, the robot labels that area as inaccessible. The code listing for the mapping task can be seen in Appendix A.

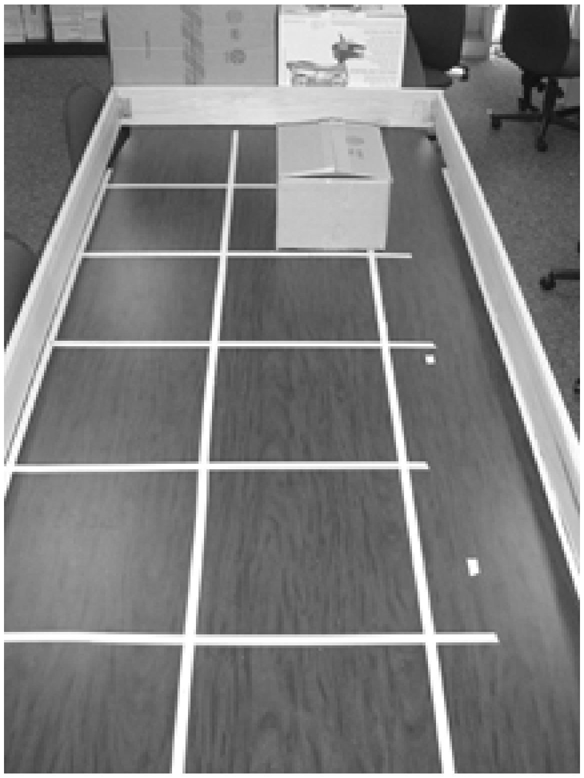


Figure 18: The testing environment

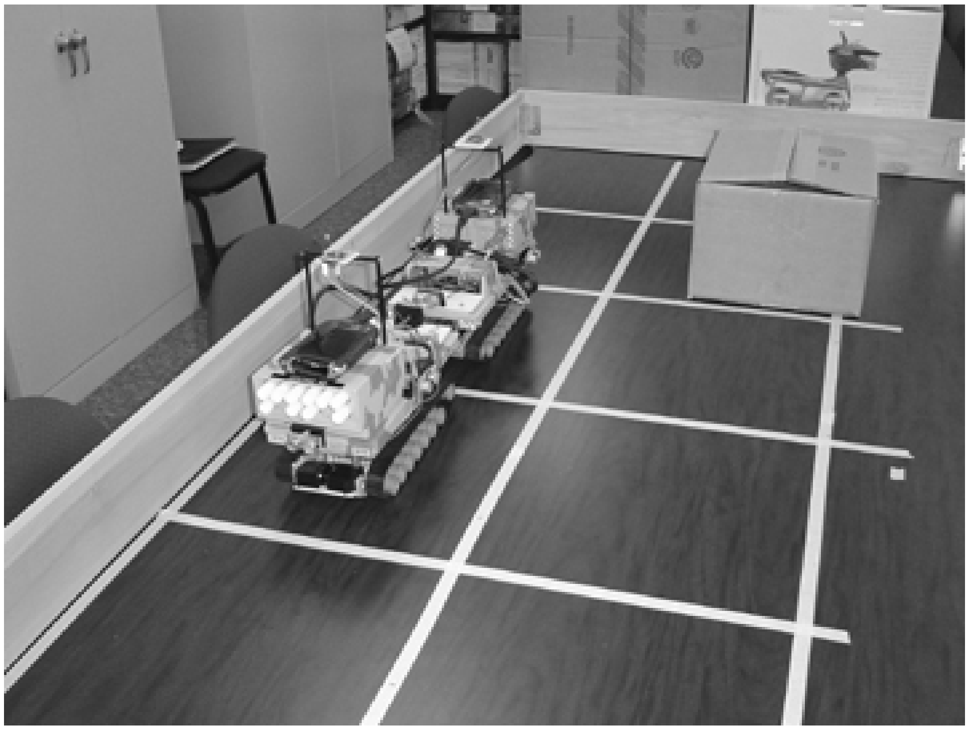


Figure 19: The Collective at the start of a mapping sequence

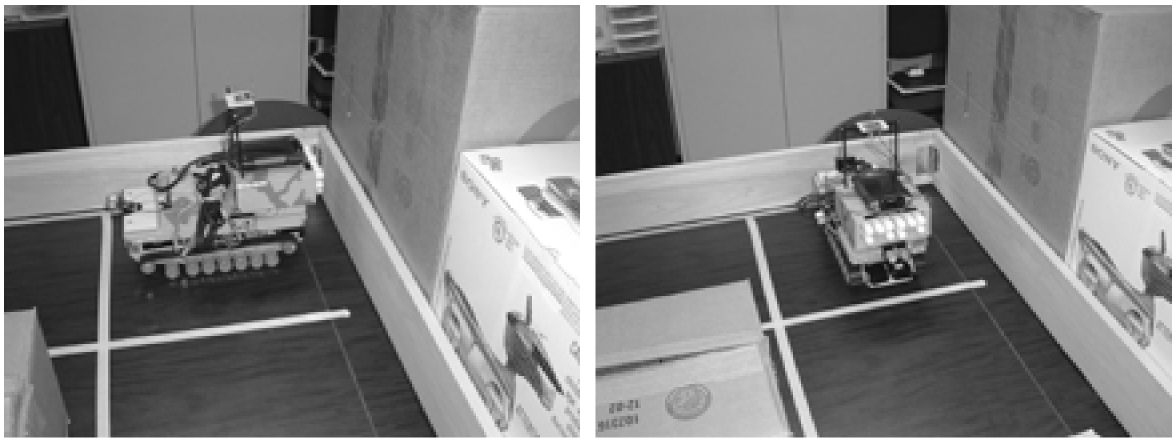


Figure 20: The object recognition behavior

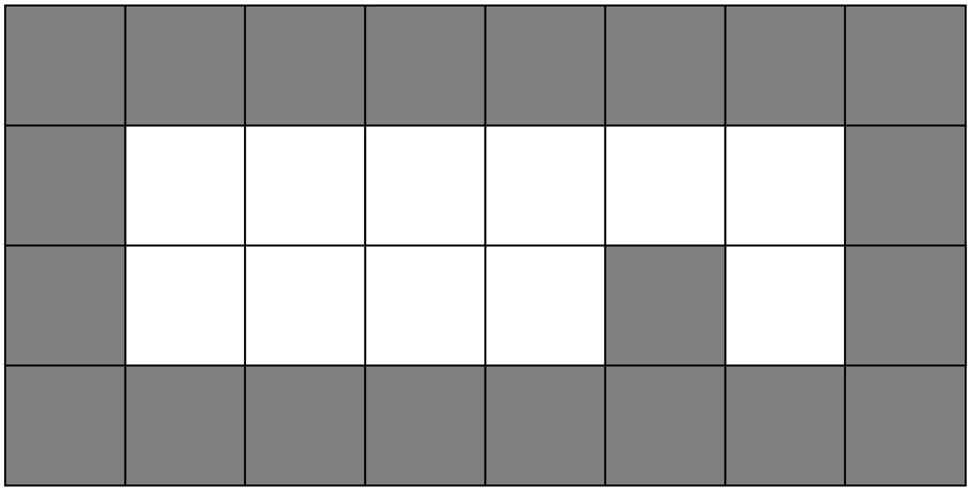


Figure 21: The final map produced by the Collective

6.2 PROBLEMS DURING DEVELOPMENT

Most of the problems associated with the development of the Collective came from the particular mediums chosen for its development. That is, the problems arose out of difficulties with the components used to design the system and not the design itself. A number of problems arose on the hardware side. One major problem was inaccuracies in the sensors, the SRF08's and the CMPS03. The major difficulty with the SRF08's was that despite their advertised range of twenty feet, their sensitivity led the actual range to be much smaller than that. It was found that when used on surfaces such as carpet, a large amount of interference occurred due to reflections off of the carpet itself. Thus, it was found that the system works best on smooth surfaces, creating the need for the tabletop testing environment. In addition, the inaccuracies in the CMPS03 also caused problems, because the robot relied on this sensor to make sure that it had turned 90° correctly. However, with some debugging this problem was overcome. Along with sensor problems, the motor scheme was not as effective as hoped. The Pulse Width Modulation provided by the Brainstem Moto did not work very well, leading to difficulty in executing the motions correctly.

In addition to hardware problems, there were software development problems as well. Initially, the design of the system called for directly accessing the Bluetooth device for the multi-robot communication aspect. However, this proved to be quite difficult due to the need for a special proprietary package to access the Bluetooth hardware. This package was difficult to use and eventually that aspect of the project was scrapped and BTSendFile.exe was used instead, making the entire development process much easier. There were also slight difficulties with the development environment, Microsoft Embedded Visual C++ (eVC++), which was a little difficult to configure.

6.3 FUTURE WORK

Despite these problems, the Behavior-Based Blackboard Architecture was easy to implement. Dividing up the task into simpler tasks made the development of the individual behaviors quite simple to manage and implement. In addition, once the key components of the system were in place, it was easy to design and debug each individual behavior by isolating it and examinining its actions in various situations.

The current project is considered to be a major cornerstone for a much larger project invloving experimentation with team-based robotics from a behavior-based perspective. Now that the core components of the blackboard have been developed, the system can be used to develop behaviors to accomplish a number of tasks related to coordinated robotics without having to worry about the basic control structure of the system. This flexibility makes the Collective an ideal platform for multi-robot system testing and development using behavior-based approaches.

REFERENCES

Arkin, R.C. 1998. *Behavior-Based Robotics*. Cambridge, MA: The MIT Press.

Barnhard, D.H., McClain, J.T., Wimpey, B.J., and Potter, W.D. 2004. Odin and Hodur: Using BlueTooth Communication for Coordinated Robotic Search. *Proceedings of the International Conference on Artificial Intelligence (ICAI'04)*, Las Vegas, Nevada, June 21st – 24th.

Brooks, R.A. 1986. A Robust Layered Control System for a Mobile Robot. *IEEE Journal of Robotics and Automation* 2(1): 14-23.

Brooks, R.A. 1990. Elephants Don't Play Chess. In *Designing Autonomous Agents*: 3-15, ed. P. Maes, Cambridge, MA: The MIT Press.

Brooks, R.A. 1991a. New Approaches to Robotics. *Science* 253:1227-1232

Brooks, R.A. 1991b. Intelligence without Representation. *Artificial Intelligence Journal* 47: 139-159.

Carver, N. and Lesser, V. 1992. The Evolution of Blackboard Control Architectures. *Expert Systems Applications, Special Issue on the Blackboard Paradigm and its Applications* 7(1): 1-30.

Fikes, R. and Nilsson, N. 1971. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence* 2: 189-208.

Gerkey, B.P., Vaughan, R.T., Stoy, K., Howard, A., Sukhatme, G.S., and Mataric, M.J. 2001. Most Valuable Player: A Robot Device Server for Distributed Control. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1226-1231, Wailea, Hawaii, October 29th – November 3rd.

Hayes-Roth, B. 1985. A Blackboard Architecture for Control. *Artificial Intelligence* 26(3): 251-321.

IEEE 2000. Part 5: Token Ring Access Method and Physical Layer Specifications – Corrigendum 1. *IEEE Standard for Information Technology – Telecommunications and Information Exchanges between Systems – Local and Metropolitan Area Networks*. New York, NY: IEEE.

Martin, K.D. 1996. A Blackboard System for Automatic Transcription of Simple Polyphonic Music. *MIT Media Laboratory Perceptual Computing Section Technical Report No. 385*.

McClain, J.T., Wimpey, B.J., Barnhard, D.H., and Potter, W.D. 2004. Distributed Robotic Target Acquisition Using BlueTooth Communication. *Proceedings of the 42nd Annual ACM Southeast Conference (ACMSE'04)*, 291-296, Huntsville, Alabama, April 2nd – 3rd.

Minsky, M.E. and Papert, S. 1974. *Artificial Intelligence*. Eugene, OR: Oregon State System of Higher Education.

Nguyen, H.G., Pezeshkian, M.R., Raymond, M., Gupta, A., and Spector, J.M. 2003. Autonomous Communication Relays for Tactical Robots. *11th Int. Conf. on Advanced Robotics*. Coimbra, Portugal, June 30th – July 3rd.

Nilsson, N. 1969. A Mobile Automaton: An Application of Artificial Intelligence Techniques. *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-69)*. Washington D.C., May.

Walter, W.G. 1953. *The Living Brain*. New York, NY: Norton.

Xu, H. and Van Brussel, H. 1997. A Behavior-Based Blackboard Architecture for Reactive and Efficient Task Execution of an Autonomous Robot. *Robotics and Autonomous Systems* 22: 115-132.

APPENDIX A

CODE LISTING FOR THE COLLECTIVE

The following code listing represents the key components of the Collective multi-robot system. Files Collective.h, Collective.cpp, CollectiveDlg.h, and CollectiveDlg.cpp were generated mostly by eVC++ with some slight modifications. Files board.h and board.cpp represent the main portions of the behavior-based blackboard system. Files aSRF08.h, aSRF08.c, aServo.h, aServo.c, aCMPS03.h, and aCMPS03.c were provided with the Brainstem development package from Acroname with slight modifications. For the entire package, visit www.Acroname.com.

Collective.h

```
// Collective.h : main header file for the COLLECTIVE application
//

#if !defined(AFX_COLLECTIVE_H__97550803_0C47_45DC_A846_750734B643A1__INCLUDED_)
#define AFX_COLLECTIVE_H__97550803_0C47_45DC_A846_750734B643A1__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#ifndef __AFXWIN_H__
	#error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"		// main symbols

/////////////////////////////////////////////////////////////////////////////
// CCollectiveApp:
// See Collective.cpp for the implementation of this class
//

class CCollectiveApp : public CWinApp
{
public:
	CCollectiveApp();

// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CCollectiveApp)
	public:
	virtual BOOL InitInstance();
	//}}AFX_VIRTUAL

// Implementation

	//{{AFX_MSG(CCollectiveApp)
		// NOTE - the ClassWizard will add and remove member functions here.
		//    DO NOT EDIT what you see in these blocks of generated code !
```

```
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};


/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft eMbedded Visual C++ will insert additional declarations immediately before the
previous line.

#endif // !defined(AFX_COLLECTIVE_H__97550803_0C47_45DC_A846_750734B643A1__INCLUDED_)
```

Collective.cpp

```
// Collective.cpp : Defines the class behaviors for the application.
//

#include "stdafx.h"
#include "Collective.h"
#include "CollectiveDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CCollectiveApp

BEGIN_MESSAGE_MAP(CCollectiveApp, CWinApp)
	//{{AFX_MSG_MAP(CCollectiveApp)
		// NOTE - the ClassWizard will add and remove mapping macros here.
		//    DO NOT EDIT what you see in these blocks of generated code!
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CCollectiveApp construction

CCollectiveApp::CCollectiveApp()
	: CWinApp()
{
	// TODO: add construction code here,
	// Place all significant initialization in InitInstance
}

/////////////////////////////////////////////////////////////////////////////
// The one and only CCollectiveApp object

CCollectiveApp theApp;

/////////////////////////////////////////////////////////////////////////////
// CCollectiveApp initialization

BOOL CCollectiveApp::InitInstance()
{
	if (!AfxSocketInit())
	{
		AfxMessageBox(IDP_SOCKETS_INIT_FAILED);
		return FALSE;
	}

	AfxEnableControlContainer();

	// Standard initialization
	// If you are not using these features and wish to reduce the size
	//  of your final executable, you should remove from the following
	//  the specific initialization routines you do not need.

	CCollectiveDlg dlg;
	m_pMainWnd = &dlg;
	int nResponse = dlg.DoModal();
	if (nResponse == IDOK)
	{
		// TODO: Place code here to handle when the dialog is
		//  dismissed with OK
	}
	else if (nResponse == IDCANCEL)
	{
		// TODO: Place code here to handle when the dialog is
		//  dismissed with Cancel
```

```
        }

        // Since the dialog has been closed, return FALSE so that we exit the
        //  application, rather than start the application's message pump.
        return FALSE;
}
```

CollectiveDlg.h

```
#if !defined(AFX_COLLECTIVEDLG_H__1C55FA76_99C5_476C_A872_D0E6CEFCD1B4__INCLUDED_)
#define AFX_COLLECTIVEDLG_H__1C55FA76_99C5_476C_A872_D0E6CEFCD1B4__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

/////////////////////////////////////////////////////////////////////////////
// CCollectiveDlg dialog

class CCollectiveDlg : public CDialog
{
// Construction
public:
	CCollectiveDlg(CWnd* pParent = NULL);	// standard constructor

// Dialog Data
	//{{AFX_DATA(CCollectiveDlg)
	enum { IDD = IDD_COLLECTIVE_DIALOG };
		// NOTE: the ClassWizard will add data members here
	//}}AFX_DATA

	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CCollectiveDlg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);	// DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
protected:
	HICON m_hIcon;

	// Generated message map functions
	//{{AFX_MSG(CCollectiveDlg)
	virtual BOOL OnInitDialog();
	afx_msg void OnGo();
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft eMbedded Visual C++ will insert additional declarations immediately before the
previous line.

#endif // !defined(AFX_COLLECTIVEDLG_H__1C55FA76_99C5_476C_A872_D0E6CEFCD1B4__INCLUDED_)
```

CollectiveDlg.cpp

```
#include "stdafx.h"
#include "board.h"
#include "Collective.h"
#include "CollectiveDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CCollectiveDlg dialog

CCollectiveDlg::CCollectiveDlg(CWnd* pParent /*=NULL*/)
	: CDialog(CCollectiveDlg::IDD, pParent)
{
	//{{AFX_DATA_INIT(CCollectiveDlg)
		// NOTE: the ClassWizard will add member initialization here
	//}}AFX_DATA_INIT
	// Note that LoadIcon does not require a subsequent DestroyIcon in Win32
	m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CCollectiveDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CCollectiveDlg)
		// NOTE: the ClassWizard will add DDX and DDV calls here
	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CCollectiveDlg, CDialog)
	//{{AFX_MSG_MAP(CCollectiveDlg)
	ON_BN_CLICKED(IDC_BUTTON1, OnGo)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CCollectiveDlg message handlers

BOOL CCollectiveDlg::OnInitDialog()
{
	CDialog::OnInitDialog();

	// Set the icon for this dialog.  The framework does this automatically
	//  when the application's main window is not a dialog
	SetIcon(m_hIcon, TRUE);			// Set big icon
	SetIcon(m_hIcon, FALSE);		// Set small icon

	CenterWindow(GetDesktopWindow());	// center to the hpc screen

	// TODO: Add extra initialization here

	return TRUE;  // return TRUE  unless you set the focus to a control
}
```

```
void CCollectiveDlg::OnGo()
{
	board *board_obj = new board;

	while(board_obj->status == NOT_DONE)
	{
		board_obj->sense();

		board_obj->map_world();

		board_obj->object_rec();

		board_obj->motion();

		if(ROBOT == ROBOT1)
		{
			board_obj->save_board("bb1.txt");
			board_obj->send_board();
			board_obj->load_board("\\My Documents\\bb2.txt");
			board_obj->delete_board("\\My Documents\\bb2.txt");
			board_obj->save_board("bb1.txt");
		}
		else if(ROBOT == ROBOT2)
		{
			board_obj->load_board("\\My Documents\\bb1.txt");
			board_obj->delete_board("\\My Documents\\bb1.txt");
			board_obj->save_board("bb2.txt");
			board_obj->send_board();
		}
	}

	delete board_obj;

	AfxMessageBox(_T("Done!"),NULL,NULL);
}
```

resource.h

```
//{{NO_DEPENDENCIES}}
// Microsoft Developer Studio generated include file.
// Used by Collective.rc
//
#define IDD_COLLECTIVE_DIALOG           102
#define IDP_SOCKETS_INIT_FAILED         103
#define IDR_MAINFRAME                   128
#define IDC_BUTTON1                     1000

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE        129
#define _APS_NEXT_COMMAND_VALUE         32771
#define _APS_NEXT_CONTROL_VALUE         1001
#define _APS_NEXT_SYMED_VALUE           101
#endif
#endif
```

sys_defines.h

```
//System defines for the Collective multi-robot system
//Created by Jonathan McClain
//Last Modified 4/19/04

#ifndef __SYS_DEFINES_H__
#define __SYS_DEFINES_H__

#include <stdio.h>
#include <string.h>

#define ROBOT1 1
#define ROBOT2 2
#define ROBOT ROBOT2
#define MAP_SIZE 100
#define aWINCE
#define aGP 2
#define CELL_WIDTH 16
#define RIGHT_MOTOR 0
#define LEFT_MOTOR 1
#define ROBOT1_RIGHT_SPEED 118
#define ROBOT1_LEFT_SPEED 140
#define ROBOT1_RIGHT_TURN_SPEED_RIGHT 150
#define ROBOT1_RIGHT_TURN_SPEED_LEFT 142
#define ROBOT1_LEFT_TURN_SPEED_RIGHT 118
#define ROBOT1_LEFT_TURN_SPEED_LEFT 116

#define ROBOT2_RIGHT_SPEED 1
#define ROBOT2_LEFT_SPEED 256
#define ROBOT2_RIGHT_TURN_SPEED_RIGHT 256
#define ROBOT2_RIGHT_TURN_SPEED_LEFT 256
#define ROBOT2_LEFT_TURN_SPEED_RIGHT 1
#define ROBOT2_LEFT_TURN_SPEED_LEFT 1

#define EMPTY 0
#define FULL 1
#define UNKNOWN -1

#define aSRF08_RIGHT 0xE0
#define aSRF08_BACK 0xE2
#define aSRF08_LEFT 0xE4
#define aSRF08_FRONT 0xE6
#define aPORTNAME "COM1:" /* note the trailing colon for WinCE */
#define aPORTSPEED 9600

#endif
```

board.h

```
//Class definition for the Collective behavior-based blackboard architecture
//Created by Jonathan McClain
//Last Modified 4/19/04

#ifndef __BOARD_H__
#define __BOARD_H__

#include "sys_defines.h"
#include "aIO.h"
#include "aPPRK.h"
#include "aStem.h"

//indices to motion commands
enum{NO_MOVE,FORWARD,TURN_LEFT,TURN_RIGHT,TURN_AROUND};

//indices to orientation
enum{NORTH,SOUTH,EAST,WEST};

//indices to anchorpoint
enum{FRONT,BACK};

//indices to status
enum{NOT_DONE,DONE};

struct robot_s
{
       int x;
       int y;
};

struct sonar_s
{
       short front;
       short back;
       short left;
       short right;
};

struct light_s
{
       int front;
       int back;
       int left;
       int right;
};

class board
{
   public:
              board(void);
              ~board(void);
              void motion(void);
              void sense(void);
              void map_world(void);
              void object_rec(void);
              int status;
              void add_cell(short x, short y, short cell);
              void save_board(const char *fname);
              void load_board(const char *fname);
              void send_board(void);
              void delete_board(const char *fname);
   private:
          robot_s self;
          robot_s partner;
          sonar_s sonar;
          light_s light;
          float heading;
          int orientation;
          int motion_command;
          int anchor;
          short map[MAP_SIZE][MAP_SIZE];
          short conversion_factor;
          aStemLib stemLibRef;
          aIOLib ioLibRef;
          void bs_init(void);
          void bs_release(void);
          void drive_forward_one_cell(void);
          void turn_right_90(void);
          void turn_left_90(void);
          void get_range_front(void);
          void get_range_back(void);
          void stop(void);
          void move_forward(void);
          void move_backward(void);
          void turn_right(void);
          void turn_left(void);
          void get_heading(void);
};

#endif
```

board.cpp

```
#include "board.h"
#include "sys_defines.h"
#include "aIO.h"
#include "aPPRK.h"
#include "aStem.h"
#include "windows.h"
#include "aSRF08.h"
#include "aCMPS03.h"
#include "aServo.h"
#include "Blackboard\map.h"
#include <afx.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <Winbase.h>
/*---------------------------------------------*/
board::board(void)
{
        int x = 0;
        int y = 0;
        //set initial coordinates
        //robot1 starts at 0,0 and robot2 starts at 0,-1
        if(ROBOT == ROBOT1)
        {
                self.x = 0;
                self.y = 0;
                partner.x = 0;
                partner.y = -1;
        }
        else if(ROBOT == ROBOT2)
        {
                self.x = 0;
                self.y = -1;
                partner.x = 0;
                partner.y = 0;
        }

        //set initial sensor readings
        sonar.back = 0;
        sonar.front = 0;
        sonar.left = 0;
        sonar.right = 0;

        light.back = 0;
        light.front = 0;
        light.left = 0;
        light.right = 0;

        heading = 0.0;

        //set initial orientation
        //robot1 faces north and robot2 faces south
        if(ROBOT == ROBOT1)
                orientation = NORTH;
        else
                orientation = SOUTH;

        //set initial motion command
        motion_command = NO_MOVE;

        //set initial anchorpoint
        anchor = FRONT;

        //initialize status
        status = NOT_DONE;

        //initialize the map
        for(x = 0; x < MAP_SIZE; x++)
```

```
	{
		for(y = 0; y < MAP_SIZE; y++)
		{
			map[x][y] = UNKNOWN;
		}
	}

	//initialize map conversion factor
	conversion_factor = MAP_SIZE / 2;

	//put initial positions in map
	add_cell(0,0,EMPTY);
	add_cell(0,-1,EMPTY);

	//initialize the brainstem network
	bs_init();
}

/*----------------------------------------------*/
board::~board(void)
{
	//release the brainstem network
	bs_release();
}

/*----------------------------------------------*/
void board::bs_init(void)
{
	ioLibRef = NULL;
	stemLibRef =  NULL;
	aStreamRef linkStream;
	aErr ioErr = aErrNone;
	aErr stemErr = aErrNone;
	aErr streamErr = aErrNone;
	aErr setStreamErr = aErrNone;
	/* Get the references to the io and stem libraries */
	aIO_GetLibRef(&ioLibRef, &ioErr);
	aStem_GetLibRef(&stemLibRef, &stemErr);

	/* Build a link stream to communicate serially with the stem */
    aStream_CreateSerial(ioLibRef, aPORTNAME, aPORTSPEED, &linkStream, &streamErr);

	/* Set this new stream as the stem's link stream. This stream
    * will automatically be destroyed when the stem is */
    aStem_SetStream(stemLibRef, linkStream, kStemModuleStream, &setStreamErr);
}

/*----------------------------------------------*/
void board::bs_release(void)
{
	/* release the BrainStem libraries now that we are done. */
    aStem_ReleaseLibRef(stemLibRef, NULL);
    aIO_ReleaseLibRef(ioLibRef, NULL);
}


/*----------------------------------------------*/
void board::sense()
{
	//initialize the SRF08s
	aSRF08_Init(stemLibRef,aGP);

	sonar.front = 0;
	sonar.back = 0;
	sonar.left = 0;
	sonar.right = 0;

	//get the ranges (multiple readings may be necessary)
	while(sonar.front == 0)
	{
		aSRF08_GetRange(stemLibRef, aGP, aSRF08_FRONT, aSRF08_INCH, &sonar.front, 1);
```

```
                Sleep(2000);
        }

        while(sonar.back == 0)
        {
                aSRF08_GetRange(stemLibRef, aGP, aSRF08_BACK, aSRF08_INCH, &sonar.back, 1);
                Sleep(2000);
        }

        while(sonar.left == 0)
        {
                aSRF08_GetRange(stemLibRef, aGP, aSRF08_LEFT, aSRF08_INCH, &sonar.left, 1);
                Sleep(2000);
        }

        while(sonar.right == 0)
        {
                aSRF08_GetRange(stemLibRef, aGP, aSRF08_RIGHT, aSRF08_INCH, &sonar.right, 1);
                Sleep(2000);
        }
}

/*------------------------------------------------*/
void board::motion(void)
{
        if(motion_command == FORWARD)
        {
                //if partner is in cell in front of us then we are done
                if((orientation == NORTH) && (partner.x == self.x) && (partner.y == (self.y + 1)))
                        status = DONE;
                if((orientation == SOUTH) && (partner.x == self.x) && (partner.y == (self.y - 1)))
                        status = DONE;
                if((orientation == EAST) && (partner.y== self.y) && (partner.x == (self.x + 1)))
                        status = DONE;
                if((orientation == WEST) && (partner.y== self.y) && (partner.x == (self.x - 1)))
                        status = DONE;
                else
                {
                        drive_forward_one_cell();
                        if(orientation == NORTH) //if facing north
                                self.y = self.y++; //increment y position
                        if(orientation == SOUTH) //if facing south
                                self.y = self.y--; //decrement y position
                        if(orientation == EAST) //if facing east
                                self.x = self.x++; //increment x position
                        if(orientation == WEST) //if facing west
                                self.x = self.x--; //decrement x position
                }
        }
        else if(motion_command == TURN_RIGHT)
        {
                turn_right_90();
                if(orientation == NORTH) //if facing north
                        orientation = EAST; //change to east
                else if(orientation == EAST) //if facing east
                        orientation = SOUTH; //change to south
                else if(orientation == SOUTH) //if facing south
                        orientation = WEST; //change to west
                else if(orientation == WEST) //if facing west
                        orientation = NORTH; //change to north
        }
        else if(motion_command == TURN_LEFT)
        {
                turn_left_90();
                if(orientation == NORTH) //if facing north
                        orientation = WEST; //change to west
                else if(orientation == WEST) //if facing west
                        orientation = SOUTH; //change to south
                else if(orientation == SOUTH) // if facing south
                        orientation = EAST; //change to east
                else if(orientation == EAST) //if facing east
```

```
			orientation = NORTH; //change to north
	}
	motion_command = NO_MOVE;
}

/*--------------------------------------------*/
void board::map_world(void)
{
	int i = 0;
	short front_object = 0;
	short back_object = 0;
	short right_object = 0;
	short left_object = 0;
	char str[64];
	CString strUnicode;

	if(orientation == NORTH) //facing up on y-axis
	{
		front_object = self.y + ((sonar.front / CELL_WIDTH) + 1);
		back_object = self.y - ((sonar.back / CELL_WIDTH) + 1);
		right_object = self.x + (((sonar.right - 3) / CELL_WIDTH) + 1);
		left_object = self.x - (((sonar.left - 3) / CELL_WIDTH) + 1);
		//add detected objects as long as the other robot is not on the same axis
		if(partner.x != self.x) // if not on the same y-axis
		{
			add_cell(self.x,front_object,FULL);

			for(i = self.y; i < front_object; i++) //add empty spaces
				add_cell(self.x,i,EMPTY);

			add_cell(self.x,back_object,FULL);

			for(i = self.y; i > back_object; i--) //add empty spaces
				add_cell(self.x,i,EMPTY);
		}
		else if(partner.y > self.y) //else if the partner is in front
		{
			add_cell(self.x,back_object,FULL);

			for(i = self.y; i > back_object; i--) //add empty spaces
				add_cell(self.x,i,EMPTY);
		}
		else //partner is behind
		{
			add_cell(self.x,front_object,FULL);

			for(i = self.y; i < front_object; i++) //add empty spaces
				add_cell(self.x,i,EMPTY);
		}

		if(partner.y != self.y) //if not on the same x-axis
		{
			add_cell(right_object,self.y,FULL);

			for(i = self.x; i < right_object; i++) //add empty spaces
				add_cell(i,self.y,EMPTY);

			add_cell(left_object,self.y,FULL);

			for(i = self.x; i > left_object; i--) //add empty spaces
				add_cell(i,self.y,EMPTY);
		}
		else if(partner.x > self.x) //else if partner is on right
		{
			add_cell(left_object,self.y,FULL);

			for(i = self.x; i > left_object; i--) //add empty spaces
				add_cell(i,self.y,EMPTY);
		}
		else //partner is on left
		{
```

```
                add_cell(right_object,self.y,FULL);

                for(i = self.x; i < right_object; i++) //add empty spaces
                        add_cell(i,self.y,EMPTY);
        }
}
else if(orientation == SOUTH) //facing down on y-axis
{
        back_object = self.y + ((sonar.back / CELL_WIDTH) + 1);
        front_object = self.y - ((sonar.front / CELL_WIDTH) + 1);
        left_object = self.x + (((sonar.left - 3) / CELL_WIDTH) + 1);
        right_object = self.x - (((sonar.right - 3) / CELL_WIDTH) + 1);

        //add detected objects
        if(self.x != partner.x) //if not on the same y-axis
        {
                add_cell(self.x,back_object,1);

                for(i = self.y; i < back_object; i++) //add empty spaces
                        add_cell(self.x,i,0);

                add_cell(self.x,front_object,1);

                for(i = self.y; i > front_object; i--)
                        add_cell(self.x,i,0); //add empty spaces
        }
        else if(partner.y > self.y) //if partner is behind
        {
                add_cell(self.x,front_object,1);

                for(i = self.y; i > front_object; i--)
                        add_cell(self.x,i,0); //add empty spaces
        }
        else //partner is in front
        {
                add_cell(self.x,back_object,1);

                for(i = self.y; i < back_object; i++) //add empty spaces
                        add_cell(self.x,i,0);
        }

        if(partner.y != self.y) //if not on the same x-axis
        {
                add_cell(left_object,self.y,1);

                for(i = self.x; i < left_object; i++) //add empty spaces
                        add_cell(i,self.y,0);

                add_cell(right_object,self.y,1);

                for(i = self.x; i > right_object; i--) //add empty spaces
                        add_cell(i,self.y,0);
        }
        else if(partner.x > self.x) //else if the partner is on the left
        {
                add_cell(right_object,self.y,1);

                for(i = self.x; i > right_object; i--) //add empty spaces
                        add_cell(i,self.y,0);
        }
        else //partner is on right
        {
                add_cell(left_object,self.y,1);

                for(i = self.x; i < left_object; i++) //add empty spaces
                        add_cell(i,self.y,0);
        }
}
else if(orientation == EAST)
{
        left_object = self.y + (((sonar.left - 3) / CELL_WIDTH) + 1);
        right_object = self.y - (((sonar.right - 3) / CELL_WIDTH) + 1);
        front_object = self.x + ((sonar.front / CELL_WIDTH) + 1);
        back_object = self.x - ((sonar.back / CELL_WIDTH) + 1);
        //add detected objects

        if(partner.x != self.x) // if not on the same y axis
        {
                add_cell(self.x,left_object,1);

                for(i = self.y; i < left_object; i++) //add empty spaces
                        add_cell(self.x,i,0);

                add_cell(self.x,right_object,1);

                for(i = self.y; i > right_object; i--) //add empty spaces
                        add_cell(self.x,i,0);
        }
        else if(partner.y < self.y) //if partner is on right
        {
                add_cell(self.x,left_object,1);

                for(i = self.y; i < left_object; i++) //add empty spaces
                        add_cell(self.x,i,0);
        }
        else //partner is on left
        {
                add_cell(self.x,right_object,1);

                for(i = self.y; i > right_object; i--) //add empty spaces
                        add_cell(self.x,i,0);
        }

        if(partner.y != self.y) //if not on the same x-axis
        {
                add_cell(front_object,self.y,1);

                for(i = self.x; i < front_object; i++) //add empty spaces
                        add_cell(i,self.y,0);

                add_cell(back_object,self.y,1);

                for(i = self.x; i > back_object; i--) //add empty spaces
                        add_cell(i,self.y,0);
        }
        else if(partner.x < self.x) //if partner is behind
        {
                add_cell(front_object,self.y,1);

                for(i = self.x; i < front_object; i++) //add empty spaces
                        add_cell(i,self.y,0);
        }
        else //partner is in front
        {
                add_cell(back_object,self.y,1);

                for(i = self.x; i > back_object; i--) //add empty spaces
                        add_cell(i,self.y,0);
        }
}
else if(orientation == WEST)
{
        right_object = self.y + (((sonar.right - 3) / CELL_WIDTH) + 1);
        left_object = self.y - (((sonar.left - 3) / CELL_WIDTH) + 1);
        back_object = self.x + ((sonar.back / CELL_WIDTH) + 1);
        front_object = self.x - ((sonar.front / CELL_WIDTH) + 1);
        //add detected objects

        if(partner.x != self.x) //if not on the same y-axis
        {
                add_cell(self.x,right_object,1);
```

```
                        for(i = self.y; i < right_object; i++) //add empty spaces
                                add_cell(self.x,i,0);

                        add_cell(self.x,left_object,1);

                        for(i = self.y; i > left_object; i--) //add empty spaces
                                add_cell(self.x,i,0);
                }
                else if(partner.y > self.y) //if partner is on right
                {
                        add_cell(self.x,left_object,1);

                        for(i = self.y; i > left_object; i--) //add empty spaces
                                add_cell(self.x,i,0);
                }
                else //partner is on left
                {
                        add_cell(self.x,right_object,1);

                        for(i = self.y; i < right_object; i++) //add empty spaces
                                add_cell(self.x,i,0);
                }

                if(partner.y != self.y) //if not on the same x-axis
                {
                        add_cell(back_object,self.y,1);

                        for(i = self.x; i < back_object; i++) //add empty spaces
                                add_cell(i,self.y,0);

                        add_cell(front_object,self.y,1);

                        for(i = self.x; i > front_object; i--) //add empty spaces
                                add_cell(i,self.y,0);
                }
                else if(partner.x > self.x) //if partner is behind
                {
                        add_cell(front_object,self.y,1);

                        for(i = self.x; i > front_object; i--) //add empty spaces
                                add_cell(i,self.y,0);
                }
                else //partner is in front
                {
                        add_cell(back_object,self.y,1);

                        for(i = self.x; i < back_object; i++) //add empty spaces
                                add_cell(i,self.y,0);
                }
        }
        if((sonar.front > CELL_WIDTH) && (motion_command != TURN_LEFT) &&
        (motion_command != TURN_RIGHT))
                motion_command = FORWARD;
}

/*---------------------------------------------*/
void board::object_rec(void)
{
        //do not use partner as an anchor
        //if facing north and partner is on same y-axis and partner is in front
        if((orientation == NORTH) && (partner.x == self.x) && (partner.y > self.y))
                anchor = BACK;
        //if facing north and partner is on same y-axis and partner is behind
        else if((orientation == NORTH) && (partner.x == self.x) && (partner.y < self.y))
                anchor = FRONT;
        //if facing south and partner is on same y-axis and partner is behind
        else if((orientation == SOUTH) && (partner.x == self.x) && (partner.y > self.y))
                anchor = FRONT;
        //if facing south and partner is on same y-axis and partner is in front
        else if((orientation == SOUTH) && (partner.x == self.x) && (partner.y < self.y))
                anchor = BACK;
        //if facing east and partner is on same x-axis and partner is in front
        else if((orientation == EAST) && (partner.y == self.y) && (partner.x > self.x))
                anchor = BACK;
        //if facing east and partner is on same x-axis and partner is behind
        else if((orientation == EAST) && (partner.y == self.y) && (partner.x < self.x))
                anchor = FRONT;
        //if facing west and partner is on same x-axis and partner is behind
        else if((orientation == WEST) && (partner.y == self.y) && (partner.x > self.x))
                anchor = FRONT;
        //if facing west and partner is on same x-axis and partner is in front
        else if((orientation == WEST) && (partner.y == self.y) && (partner.x < self.x))
                anchor = BACK;
        //else choose the shortest point and make it the anchor
        else if(sonar.front <= sonar.back)
                anchor = FRONT;
        else
                anchor = BACK;
        if(ROBOT == ROBOT1)
        {
                if(sonar.left > CELL_WIDTH) //if we can go left
                        motion_command = TURN_LEFT; //then turn left
                else if(sonar.front <= CELL_WIDTH) //else if can't go straight
                        motion_command = TURN_RIGHT; //turn right
        }
        else
        {
                if(sonar.right > CELL_WIDTH) //if we can go right
                        motion_command = TURN_RIGHT; //then turn right
                else if(sonar.front <= CELL_WIDTH) //else if can't go straight
                        motion_command = TURN_LEFT; //turn left
        }
}

/*----------------------------------------------*/
void board::add_cell(short x, short y, short cell)
{
        int conv_x = x + conversion_factor;
        int conv_y = y + conversion_factor;
        map[conv_x][conv_y] = cell;
}

/*----------------------------------------------*/
void board::save_board(const char *fname)
{
        FILE *bb_file;
        int x = 0;
        int y = 0;
        int conv_x = 0;
        int conv_y = 0;

    if ((bb_file = fopen(fname,"w+")) == NULL)
                MessageBox(NULL,_T("File did not open!"),NULL,NULL);

        fprintf(bb_file,"%d %d\n",self.x,self.y); //add current position

        for(x = 0; x < MAP_SIZE; x++)
        {
                for(y = 0; y < MAP_SIZE; y++)
                {
                        if(map[x][y] != UNKNOWN)
                        {
                                conv_x = x - conversion_factor;
                                conv_y = y - conversion_factor;
                                fprintf(bb_file,"%d %d %d\n",conv_x,conv_y,map[x][y]);
                        }
                }
        }
        fprintf(bb_file,"\n");
        fclose(bb_file);
}
```

```
/*----------------------------------------------*/
void board::load_board(const char *fname)
{
        FILE *bb_file;
        short x = 0;
        short y = 0;
        short cell = 0;
        char str[64];
        CString strUnicode;

        while((bb_file = fopen(fname,"r")) == NULL)
                MessageBox(NULL,_T("failed to open file...trying again"),NULL,NULL);

        fscanf(bb_file,"%d %d\n",&partner.x,&partner.y);
        add_cell(partner.x,partner.y,EMPTY);
        while(!feof(bb_file))
        {
                fscanf(bb_file,"%hd %hd %hd\n",&x,&y,&cell);
                add_cell(x,y,cell);
        }
        fclose(bb_file);
}

/*----------------------------------------------*/
void board::send_board(void)
{
        CString errmsg;
        PROCESS_INFORMATION      pi;
        DWORD rc;
        if(ROBOT == ROBOT1)
                rc = CreateProcess (_T("\\Windows\\Start Menu\\BTSendFile.exe"),
                                    _T("dev=00:02:C7:17:54:88 file=\\bb1.txt"),
                                    NULL, NULL, FALSE, 0, NULL, NULL, NULL, &pi);
        else
                rc = CreateProcess (_T("\\Windows\\Start Menu\\BTSendFile.exe"),
                                    _T("dev=00:02:C7:17:50:97 file=\\bb2.txt"),
                                    NULL, NULL, FALSE, 0, NULL, NULL, NULL, &pi);
        errmsg = "Error: There as a problem launching BTSendFile.exe!";
        if (rc == 0){
                MessageBox(NULL,errmsg,NULL,NULL);
                  return;  //Error launching program
                  }

        //Wait for it to complete
        DWORD dwExitCode;
        while (::GetExitCodeProcess (pi.hProcess,  &dwExitCode) )
        {
                if (dwExitCode == STILL_ACTIVE)
                          Sleep(100);
                else if (dwExitCode == 0){
                        errmsg = "BTSendFile ended normally";
                        MessageBox(NULL,errmsg,NULL,NULL);
                          break;
                }
                else if (dwExitCode == 1){
                        errmsg = "Error: Syntax Error!";
                        MessageBox(NULL,errmsg,NULL,NULL);
                          break;
                }
                else if (dwExitCode == 2){
                        errmsg = "Error: Unable to connect to the Bluetooth Stack!";
                        MessageBox(NULL,errmsg,NULL,NULL);
                          break;
                }
                else if (dwExitCode == 3){
                        errmsg = "Error: Device not found!";
                        MessageBox(NULL,errmsg,NULL,NULL);
                          break;
                }
                else if (dwExitCode == 4){
                        errmsg = "Error: File transfer service not available!";
                        MessageBox(NULL,errmsg,NULL,NULL);
                          break;
                }
                else if (dwExitCode == 6){
                        errmsg = "Error: Problem during connection or while sending file!";
                        MessageBox(NULL,errmsg,NULL,NULL);
                          break;
                }
                else if (dwExitCode == 7){
                        errmsg = "Error: Timed out waiting for connection or file transfer to
                        complete!";
                        MessageBox(NULL,errmsg,NULL,NULL);
                          break;
                }
                else if (dwExitCode == 8){
                        errmsg = "Error: Bluetooth radio was turned off!";
                        MessageBox(NULL,errmsg,NULL,NULL);
                          break;
                }
                else if (dwExitCode == 9){
                        errmsg = "Error: File not found!";
                        MessageBox(NULL,errmsg,NULL,NULL);
                          break;
                }
        }

        //Close process
        CloseHandle(pi.hThread);
        CloseHandle(pi.hProcess);
}

/*----------------------------------------------*/
void board::delete_board(const char *fname)
{
        CString strUnicode;
        strUnicode = fname;
        DeleteFile(strUnicode);
}

/*----------------------------------------------*/
void board::drive_forward_one_cell(void)
{
        short goal = 0;
        short temp = 0;

        aSRF08_Init(stemLibRef, aGP);

        while(sonar.front == 0) //if we misread
                get_range_front(); //read again
        while(sonar.back == 0) //if we misread
                get_range_back(); //read again
        if(sonar.front <= CELL_WIDTH)
        {
                MessageBox(NULL,_T("ERROR: TRYING TO MOVE INTO A FILLED CELL"),NULL,NULL);
                status = DONE;
        }
        else if(anchor == FRONT)
        {
                goal = sonar.front - CELL_WIDTH;

                while(sonar.front > goal)
                {
                        move_forward();
                        Sleep(500);
                        stop();
                        temp = sonar.front; //store previous
                        get_range_front();
                        if(sonar.front == 0) //misread
                                sonar.front = temp; //so revert to previous reading
                }
        }
```

```
	else if(anchor == BACK)
	{
		goal = sonar.back + CELL_WIDTH;

		while(sonar.back < goal)
		{
			move_forward();
			Sleep(500);
			stop();
			temp = sonar.back; //store previous
			get_range_back();
			if(sonar.back == 0) //misread
				sonar.back = temp; //so revert to previous reading
		}
	}
}

/*---------------------------------------------*/
void board::turn_right_90(void)
{
	float goal = 0.0;
	float low_spread = 0.0;
	float high_spread = 0.0;
	char str[64];
	CString strUnicode;
	while(heading == 0.0)
	{
		get_heading();
		Sleep(1000);
	}

	if((heading + 70) > 360)
		goal = 70 - (360 - heading);
	else
		goal = heading + 70;

	if((goal - 3) < 0)
		low_spread = 360 + (goal - 3);
	else
		low_spread = goal - 3;

	if((goal + 3) > 360)
		high_spread = 3 - (360 - goal);
	else
		high_spread = goal + 3;

	while(!((heading >= low_spread) && (heading <= high_spread)))
	{
		turn_right();
		Sleep(250);
		stop();
		get_heading();
	}

}

/*---------------------------------------------*/
void board::turn_left_90(void)
{
	float goal = 0.0;
	float low_spread = 0.0;
	float high_spread = 0.0;
	char str[64];
	CString strUnicode;
	while(heading == 0.0)
	{
		get_heading();
		Sleep(1000);
	}

	if((heading - 70) < 0)
```

```
                goal = 360 + (heading - 70);
        else
                goal = heading - 70;

        if((goal - 3) < 0)
                low_spread = 360 + (goal - 3);
        else
                low_spread = goal - 3;

        if((goal + 3) > 360)
                high_spread = 3 - (360 - goal);
        else
                high_spread = goal + 3;

        while(!((heading >= low_spread) && (heading <= high_spread)))
        {
                turn_left();
                Sleep(250);
                stop();
                get_heading();
        }

}

/*---------------------------------------------*/
void board::turn_right(void)
{
        //robot motor speeds differ because of hardware differences
        if(ROBOT == ROBOT1)
        {
                aServo_SetPositionAbs(stemLibRef, aGP, RIGHT_MOTOR,
ROBOT1_RIGHT_TURN_SPEED_RIGHT);
                aServo_SetPositionAbs(stemLibRef, aGP, LEFT_MOTOR, ROBOT1_RIGHT_TURN_SPEED_LEFT);
        }
        else
        {
                aServo_SetPositionAbs(stemLibRef, aGP, RIGHT_MOTOR,
ROBOT2_RIGHT_TURN_SPEED_RIGHT);
                aServo_SetPositionAbs(stemLibRef, aGP, LEFT_MOTOR, ROBOT2_RIGHT_TURN_SPEED_LEFT);
        }
}

/*---------------------------------------------*/
void board::turn_left(void)
{
        //robot motor speeds differ because of hardware differences
        if(ROBOT == ROBOT1)
        {
                aServo_SetPositionAbs(stemLibRef, aGP, RIGHT_MOTOR, ROBOT1_LEFT_TURN_SPEED_RIGHT);
                aServo_SetPositionAbs(stemLibRef, aGP, LEFT_MOTOR, ROBOT1_LEFT_TURN_SPEED_LEFT);
        }
        else
        {
                aServo_SetPositionAbs(stemLibRef, aGP, RIGHT_MOTOR, ROBOT2_LEFT_TURN_SPEED_RIGHT);
                aServo_SetPositionAbs(stemLibRef, aGP, LEFT_MOTOR, ROBOT2_LEFT_TURN_SPEED_LEFT);
        }
}

/*---------------------------------------------*/
void board::get_heading(void)
{
        heading = 0.0;
        aCMPS03_GetHeading(stemLibRef, aGP, aCMPS03_DEGREES, &heading);
        Sleep(750);
}

/*---------------------------------------------*/
void board::get_range_front(void)
{
        sonar.front = 0;
        aSRF08_GetRange(stemLibRef, aGP, aSRF08_FRONT, aSRF08_INCH, &sonar.front, 1);
        Sleep(750);
}

/*---------------------------------------------*/
void board::get_range_back(void)
{
        sonar.back = 0;
        aSRF08_GetRange(stemLibRef, aGP, aSRF08_BACK, aSRF08_INCH, &sonar.back, 1);
        Sleep(750);
}

/*---------------------------------------------*/
void board::stop(void)
{
        aServo_SetPositionAbs(stemLibRef, aGP, RIGHT_MOTOR, 128);
        aServo_SetPositionAbs(stemLibRef, aGP, LEFT_MOTOR, 128);
}

/*---------------------------------------------*/
void board::move_forward(void)
{
        //robot motor speeds differ because of hardware differences
        if(ROBOT == ROBOT1)
        {
                aServo_SetPositionAbs(stemLibRef, aGP, RIGHT_MOTOR, ROBOT1_RIGHT_SPEED);
                aServo_SetPositionAbs(stemLibRef, aGP, LEFT_MOTOR, ROBOT1_LEFT_SPEED);
        }
        else
        {
                aServo_SetPositionAbs(stemLibRef, aGP, RIGHT_MOTOR, ROBOT2_RIGHT_SPEED);
                aServo_SetPositionAbs(stemLibRef, aGP, LEFT_MOTOR, ROBOT2_LEFT_SPEED);
        }
}

/*---------------------------------------------*/
void board::move_backward(void)
{
        //robot motor speeds differ because of hardware differences
        if(ROBOT == ROBOT1)
        {
                aServo_SetPositionAbs(stemLibRef, aGP, RIGHT_MOTOR, ROBOT1_LEFT_SPEED);
                aServo_SetPositionAbs(stemLibRef, aGP, LEFT_MOTOR, ROBOT1_RIGHT_SPEED);
        }
        else
        {
                aServo_SetPositionAbs(stemLibRef, aGP, RIGHT_MOTOR, ROBOT2_LEFT_SPEED);
                aServo_SetPositionAbs(stemLibRef, aGP, LEFT_MOTOR, ROBOT2_RIGHT_SPEED);
        }
}
```

aServo.h

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                                 */
/* file: aServo.h                                                  */
/*                                                                 */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                                 */
/* description: Implementation of servo control routines for the   */
/*             BrainStem modules.                                  */
/*                                                                 */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                                 */
/* Copyright © 1997-2004. Acroname Inc.                            */
/*                                                                 */
/* This software is the property of Acroname Inc.  Any             */
/* distribution, sale, transmission, or re-use of this code is     */
/* strictly forbidden except with permission from Acroname Inc.    */
/*                                                                 */
/* To the full extent allowed by law, Acroname Inc. also excludes  */
/* for itself and its suppliers any liability, wheither based in   */
/* contract or tort (including negligence), for direct,            */
/* incidental, consequential, indirect, special, or punitive       */
/* damages of any kind, or for loss of revenue or profits, loss of */
/* business, loss of information or data, or other financial loss  */
/* arising out of or in connection with this software, even if     */
/* Acroname Inc. has been advised of the possibility of such       */
/* damages.                                                        */
/*                                                                 */
/* Acroname Inc.                                                   */
/* www.acroname.com                                                */
/* 720-564-0373                                                    */
/*                                                                 */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#ifndef _aServo_H_
#define _aServo_H_

#include "aStem.h"
#include "aStemCore.h"


/* defines for setting servo configuration */
/* these may be ORed together to configure a servo output */

#define aSERVO_DSTA             32
#define aSERVO_INV              64
#define aSERVO_ENA              128
#define aSERVO_SPEEDMASK        0x0F


#ifdef __cplusplus
extern "C" {
#endif /* __cplusplus */

aErr aServo_GetPosition(aStemLib stemLib,
                      const unsigned char module,
                      const unsigned char nServoIndex,
                      unsigned char *pServoPosition);

aErr aServo_GetConfig(aStemLib stemLib,
                     const unsigned char module,
                     const unsigned char nServoIndex,
                     unsigned char *pConfigByte);

aErr aServo_GetLimits(aStemLib stemLib,
                     const unsigned char module,
                     const unsigned char nServoIndex,
                     unsigned char *pLimitBytes);
aErr aServo_SetPositionAbs(aStemLib stemLib,
                          const unsigned char module,
                          const unsigned char nServoIndex,
                          const unsigned char nPosition);

aErr aServo_SetPositionRel(aStemLib stemLib,
                          const unsigned char module,
                          const unsigned char nServoIndex,
                          const unsigned char nChange);

aErr aServo_SetConfig(aStemLib stemLib,
                     const unsigned char module,
                     const unsigned char nServoIndex,
                     const unsigned char configByte);

aErr aServo_SetLimits(aStemLib stemLib,
                     const unsigned char module,
                     const unsigned char nServoIndex,
                     const unsigned char *pLimitBytes);

aErr aServo_CommitSettings(aStemLib stemLib,
                          const unsigned char module);

#ifdef __cplusplus
}
#endif

#endif /* _aServo_H_ */
```

aServo.c

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                                   */
/* file: aServo.c                                                     */
/*                                                                   */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                                   */
/* description: Implementation of servo control routines for the    */
/*              BrainStem modules.                                   */
/*                                                                   */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                                   */
/* Copyright © 1997-2004. Acroname Inc.                              */
/*                                                                   */
/* This software is the property of Acroname Inc.  Any               */
/* distribution, sale, transmission, or re-use of this code is       */
/* strictly forbidden except with permission from Acroname Inc.      */
/*                                                                   */
/* To the full extent allowed by law, Acroname Inc. also excludes   */
/* for itself and its suppliers any liability, wheither based in    */
/* contract or tort (including negligence), for direct,              */
/* incidental, consequential, indirect, special, or punitive         */
/* damages of any kind, or for loss of revenue or profits, loss of */
/* business, loss of information or data, or other financial loss   */
/* arising out of or in connection with this software, even if       */
/* Acroname Inc. has been advised of the possibility of such         */
/* damages.                                                          */
/*                                                                   */
/* Acroname Inc.                                                     */
/* www.acroname.com                                                  */
/* 720-564-0373                                                      */
/*                                                                   */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#include "aCmd.tea"

#include "sys_defines.h"
#include "aUtil.h"
#include "aServo.h"

#define aSERVOREPLYTIMEOUTMS  100


/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 * aServo_GetPosition
 */

aErr aServo_GetPosition(aStemLib stemLib,
                        const unsigned char module,
                        const unsigned char nServoIndex,
                        unsigned char *pServoPosition)
{
  aErr servoErr = aErrNone;
  char data[aSTEMMAXPACKETBYTES];
  unsigned char address;
  unsigned char length;
  aPacketRef packet;

  aAssert(stemLib);
  aAssert(pServoPosition);

  data[0] = cmdSRV_ABS;
  data[1] = (char)nServoIndex;

  /* build and send the packet */
  aPacket_Create(stemLib, module, 2, data, &packet, &servoErr);
  if (servoErr == aErrNone)
    aStem_SendPacket(stemLib, packet, &servoErr);
  /* look for the reply */
  if (servoErr == aErrNone)
    aStem_GetPacket(stemLib, aStemCore_CmdFilter,
                     (void*)cmdSRV_ABS,
                    aSERVOREPLYTIMEOUTMS,
                    &packet,
                    &servoErr);

  /* get the packet data */
  if (servoErr == aErrNone)
    aPacket_GetData(stemLib, packet, &address, &length, data, &servoErr);

  if (servoErr == aErrNone) {
    aAssert(data[0] == cmdSRV_ABS);
    aAssert(data[1] == nServoIndex);
    *pServoPosition = (unsigned char)data[2];
    aPacket_Destroy(stemLib, packet, &servoErr);
  }

  return servoErr;

} /* aServo_GetPosition */


/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 * aServo_GetServoConfig
 */

aErr aServo_GetConfig(aStemLib stemLib,
                              const unsigned char module,
                              const unsigned char nServoIndex,
                              unsigned char *pConfigByte)
{
  aErr servoErr = aErrNone;
  char data[aSTEMMAXPACKETBYTES];
  unsigned char address;
  unsigned char length;
  aPacketRef packet;

  aAssert(stemLib);
  aAssert(pConfigByte);

  data[0] = cmdSRV_CFG;
  data[1] = (char)nServoIndex;

  /* build and send the packet */
  aPacket_Create(stemLib, module, 2, data, &packet, &servoErr);
  if (servoErr == aErrNone)
    aStem_SendPacket(stemLib, packet, &servoErr);

  /* look for the reply */
  if (servoErr == aErrNone)
    aStem_GetPacket(stemLib, aStemCore_CmdFilter, (void*)cmdSRV_CFG,
                    aSERVOREPLYTIMEOUTMS, &packet, &servoErr);

  /* get the packet data */
  if (servoErr == aErrNone)
    aPacket_GetData(stemLib, packet, &address, &length, data, &servoErr);

  if (servoErr == aErrNone) {
    aAssert(data[0] == cmdSRV_CFG);
    aAssert(data[1] == nServoIndex);
    *pConfigByte = (unsigned char)data[2];
    aPacket_Destroy(stemLib, packet, &servoErr);
  }

  return servoErr;

} /* aServo_GetServoConfig */
```

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 * aServo_GetLimits
 */

aErr aServo_GetLimits(aStemLib stemLib,
                              const unsigned char module,
                              const unsigned char nServoIndex,
                              unsigned char *pLimitBytes)
{

  aErr servoErr = aErrNone;
  char data[aSTEMMAXPACKETBYTES];
  unsigned char address;
  unsigned char length;
  aPacketRef packet;

  aAssert(stemLib);
  aAssert(pLimitBytes);

  data[0] = cmdSRV_LMT;
  data[1] = (char)nServoIndex;

  /* build and send the packet */
  aPacket_Create(stemLib, module, 2, data, &packet, &servoErr);
  if (servoErr == aErrNone)
    aStem_SendPacket(stemLib, packet, &servoErr);

  /* look for the reply */
  if (servoErr == aErrNone)
    aStem_GetPacket(stemLib,
                    aStemCore_CmdFilter,
                    (void*)cmdSRV_LMT,
                   aSERVOREPLYTIMEOUTMS,
                   &packet,
                   &servoErr);

  /* get the packet data */
  if (servoErr == aErrNone)
    aPacket_GetData(stemLib, packet, &address, &length, data, &servoErr);

  if (servoErr == aErrNone) {
    aAssert(data[0] == cmdSRV_LMT);
    aAssert(data[1] == nServoIndex);
    aMemCopy(pLimitBytes, &data[2], 2);
    aPacket_Destroy(stemLib, packet, &servoErr);
  }

  return servoErr;

} /* aServo_GetServoLimits */


/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 * aServo_SetPositionAbs
 */

aErr aServo_SetPositionAbs(aStemLib stemLib,
                                   const unsigned char module,
                                   const unsigned char nServoIndex,
                                   const unsigned char nPosition)
{
  aErr servoErr = aErrNone;
  char data[aSTEMMAXPACKETBYTES];
  aPacketRef packet;

  aAssert(stemLib);

  data[0] = cmdSRV_ABS;
  data[1] = (char)nServoIndex;
  data[2] = (char)nPosition;
  aPacket_Create(stemLib, module, 3, data, &packet, &servoErr);

  if (servoErr == aErrNone)
    aStem_SendPacket(stemLib, packet, &servoErr);

  return servoErr;

} /* aServo_SetServoPositionAbs */



/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 * aServo_SetServoPositionRel
 */

aErr aServo_SetPositionRel(aStemLib stemLib,
                                   const unsigned char module,
                                   const unsigned char nServoIndex,
                                   const unsigned char nChange)
{
  aErr servoErr = aErrNone;
  char data[aSTEMMAXPACKETBYTES];
  aPacketRef packet;

  aAssert(stemLib);

  data[0] = cmdSRV_ABS;
  data[1] = (char)nServoIndex;
  data[2] = (char)nChange;

  aPacket_Create(stemLib, module, 3, data, &packet, &servoErr);

  if (servoErr == aErrNone)
    aStem_SendPacket(stemLib, packet, &servoErr);

  return servoErr;

} /* aServo_SetServoPositionRel */


/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 * aServo_SetConfig
 */

aErr aServo_SetConfig(aStemLib stemLib,
                              const unsigned char module,
                              const unsigned char nServoIndex,
                              const unsigned char configByte)
{
  aErr servoErr = aErrNone;
  char data[aSTEMMAXPACKETBYTES];
  aPacketRef packet;

  aAssert(stemLib);

  data[0] = cmdSRV_CFG;
  data[1] = (char)nServoIndex;
  data[2] = (char)configByte;

  aPacket_Create(stemLib, module, 3, data, &packet, &servoErr);

  if (servoErr == aErrNone)
    aStem_SendPacket(stemLib, packet, &servoErr);

  return servoErr;

} /* aServo_SetServoConfig */


/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

```
 * aServo_SetServoLimits
 */

aErr aServo_SetLimits(aStemLib stemLib,
                              const unsigned char module,
                              const unsigned char nServoIndex,
                              const unsigned char *pLimitBytes)
{
  aErr servoErr = aErrNone;
  char data[aSTEMMAXPACKETBYTES];
  aPacketRef packet;

  aAssert(stemLib);
  aAssert(pLimitBytes);

  data[0] = cmdSRV_LMT;
  data[1] = (char)nServoIndex;
  aMemCopy(&data[2], pLimitBytes, 2);

  /* build and send the packet */
  aPacket_Create(stemLib, module, 4, data, &packet, &servoErr);
  if (servoErr == aErrNone)
    aStem_SendPacket(stemLib, packet, &servoErr);

  return servoErr;

} /* aServo_SetLimits */


/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 * aServo_CommitSettings
 */

aErr aServo_CommitSettings(aStemLib stemLib,
                         const unsigned char module)
{

  aErr servoErr = aErrNone;
  char data[aSTEMMAXPACKETBYTES];
  aPacketRef packet;

  aAssert(stemLib);

  data[0] = cmdSRV_SAV;

  /* build and send the packet */
  aPacket_Create(stemLib, module, 1, data, &packet, &servoErr);
  if (servoErr == aErrNone)
    aStem_SendPacket(stemLib, packet, &servoErr);

  return servoErr;

} /* aServo_CommitSettings */
```

aSRF08.h

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                                  */
/* file: aSRF08.c                                                   */
/*                                                                  */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                                  */
/* description: Definition of cross-platform SRF08 Ranger module   */
/*              interface routines.                                 */
/*                                                                  */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                                  */
/*        docs: Documentation can be found in the BrainStem        */
/*             reference under the major topic "C", category       */
/*              aSRF08                                              */
/*                                                                  */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                                  */
/* Copyright © 1997-2004. Acroname Inc.                             */
/*                                                                  */
/* This software is the property of Acroname Inc.  Any              */
/* distribution, sale, transmission, or re-use of this code is     */
/* strictly forbidden except with permission from Acroname Inc.    */
/*                                                                  */
/* To the full extent allowed by law, Acroname Inc. also excludes  */
/* for itself and its suppliers any liability, wheither based in   */
/* contract or tort (including negligence), for direct,            */
/* incidental, consequential, indirect, special, or punitive       */
/* damages of any kind, or for loss of revenue or profits, loss of */
/* business, loss of information or data, or other financial loss  */
/* arising out of or in connection with this software, even if     */
/* Acroname Inc. has been advised of the possibility of such       */
/* damages.                                                         */
/*                                                                  */
/* Acroname Inc.                                                    */
/* www.acroname.com                                                 */
/* 720-564-0373                                                     */
/*                                                                  */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#ifndef _aSRF08_H_
#define _aSRF08_H_

#include "aIO.h"
#include "aStem.h"
#include "aSRF08Defs.tea"


#ifdef __cplusplus
extern "C" {
#endif
void aSRF08_Init(aStemLib stemLib,const unsigned char router);
aErr aSRF08_GetRange(aStemLib stemLib,
                    const unsigned char router,
                    const unsigned char srf08address,
                    const unsigned char units,
                    short* pVals,
                    const unsigned char nVals);

aErr aSRF08_GetLight(aStemLib stemLib,
                    const unsigned char router,
                    const unsigned char srf08address,
                    unsigned char* pVal);
#ifdef __cplusplus
}
#endif

#endif /* _aSRF08_H_ */
```

aSRF08.c

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                                   */
/* file: aSRF08.h                                                    */
/*                                                                   */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                                   */
/* description: Implementation of cross-platform SRF08 Ranger        */
/*              module interface routines.                           */
/*                                                                   */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                                   */
/*        docs: Documentation can be found in the BrainStem          */
/*             reference under the major topic "C", category         */
/*              aSRF08                                               */
/*                                                                   */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                                   */
/* Copyright © 1997-2004. Acroname Inc.                              */
/*                                                                   */
/* This software is the property of Acroname Inc.  Any               */
/* distribution, sale, transmission, or re-use of this code is       */
/* strictly forbidden except with permission from Acroname Inc.      */
/*                                                                   */
/* To the full extent allowed by law, Acroname Inc. also excludes    */
/* for itself and its suppliers any liability, wheither based in     */
/* contract or tort (including negligence), for direct,              */
/* incidental, consequential, indirect, special, or punitive         */
/* damages of any kind, or for loss of revenue or profits, loss of   */
/* business, loss of information or data, or other financial loss    */
/* arising out of or in connection with this software, even if       */
/* Acroname Inc. has been advised of the possibility of such         */
/* damages.                                                          */
/*                                                                   */
/* Acroname Inc.                                                     */
/* www.acroname.com                                                  */
/* 720-564-0373                                                      */
/*                                                                   */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#include "sys_defines.h"
#include "aCmd.tea"
#include "aStemMsg.h"
#include "aUtil.h"
#include "aSRF08.h"


/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 * local defines
 */

#define aSRF08_GETDATATIMEOUTMS              100



/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 * local prototypes
 */

static aBool sSRF08IICErrFilter(const unsigned char module,
                              const unsigned char dataLength,
                              const char* data,
                              void* ref);
static aBool sSRF08IICDataFilter(const unsigned char module,
                               const unsigned char dataLength,
                               const char* data,
                               void* ref);
```

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 * sSRF08IICErrFilter
 */

aBool sSRF08IICErrFilter(const unsigned char module,
                       const unsigned char dataLength,
                       const char* data,
                       void* ref)
{
  if ((dataLength > 1)
      && (data[0] == cmdMSG)
      && (data[1] == iicNoAck))
    return aTrue;

  return aFalse;

} /* sSRF08IICErrFilter */



/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 * sSRF08IICDataFilter
 *
 * This uses the hard-coded device ID of 52 which is valid for
 * the GP and Moto boards.  It should probably be more general.
 */

aBool sSRF08IICDataFilter(const unsigned char module,
                       const unsigned char dataLength,
                       const char* data,
                       void* ref)
{
  aAssert(pAddr);

  if ((dataLength > 1)
      && (data[0] == cmdDEV_VAL)
      && (data[1] == 52))
    return aTrue;

  return aFalse;

} /* sSRF08IICErrFilter */



/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 * aSRF08_Init
 *
 * This initializes the SRF08 by setting the Brainstem IIC baud
 * rate to 400.
 */

void aSRF08_Init(aStemLib stemLib, const unsigned char module)
{
  aErr InitErr = aErrNone;
  char data[aSTEMMAXPACKETBYTES];
  aPacketRef packet;

  aAssert(stemLib);

  data[0] = cmdVAL_SET;
  data[1] = 3;
  data[2] = 1;

  aPacket_Create(stemLib, module, 3, data, &packet, &InitErr);

  if (InitErr == aErrNone)
    aStem_SendPacket(stemLib, packet, &InitErr);

}
```

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 * aSRF08_GetRange
 *
 * This routine initiates a sonar ranging from the addressed
 * SRF08 and returns the requested number of range values
 * in the specified units.
 *
 * This routine requires the IIC baud rate to be set to 400kHz
 * as the SRF08 cannot handle 1MBit communciation.
 */

aErr aSRF08_GetRange(aStemLib stemLib,
                    const unsigned char router,
                    const unsigned char srf08address,
                    const unsigned char units,
                    short* pVals,
                    const unsigned char nVals)
{
  aErr srf08Err = aErrNone;
  unsigned char module;
  unsigned char length;
  char data[aSTEMMAXPACKETBYTES];
  aPacketRef packet;

  /* check all the parameters and get set up */
  if (!stemLib || !pVals || !router)
    srf08Err = aErrParam;
  if ((srf08Err == aErrNone)
      && ((units != aSRF08_INCH)
          && (units != aSRF08_CM)
          && (units != aSRF08_MS)))
    srf08Err = aErrParam;
  if ((srf08Err == aErrNone)
      && ((srf08address % 1)
          || (srf08address < 0xE0)
          || (srf08address == 0xFF)))
    srf08Err = aErrParam;
  if ((srf08Err == aErrNone)
      && ((nVals == 0) || (nVals > aSRF08_NMAXREADINGS)))
    srf08Err = aErrRange;

  /* set the memory pointer in the SRF08 */
  if (srf08Err == aErrNone) {
    data[0] = 0;
    data[1] = (char)units;
    if (!aPacket_Create(stemLib, srf08address,
                       2, data, &packet, &srf08Err))
      aStem_SendPacket(stemLib, packet, &srf08Err);
  }

  /* now, wait for the reading and look for an IIC ACK error which
   * would indicate no SRF08 connected at that address */
  if (srf08Err == aErrNone) {
    if (!aStem_GetPacket(stemLib,
                        sSRF08IICErrFilter,
                        NULL,
                        aSRF08_READDELAYMS,
                        &packet,
                        &srf08Err)) {
      aPacket_Destroy(stemLib, packet, NULL);
      srf08Err = aErrNotFound;
    } else {
      if (srf08Err == aErrTimeout)
        srf08Err = aErrNone;
    }
  }

  /* now, if there are no errors, we can assume there is:
   *  a) no brainstem attached at all
   *  b) a functioning SRF08 at the requested address
   *  c) some other IIC device at that address
   */

  /* set the SRF08 register pointer to the range data */
  if (srf08Err == aErrNone) {
    data[0] = 2;
    if (!aPacket_Create(stemLib, srf08address,
                        1, data, &packet, &srf08Err))
      aStem_SendPacket(stemLib, packet, &srf08Err);
  }

  /* read out the data up to 3 shorts at a time */
  if (srf08Err == aErrNone) {
    int i;
    short* p = pVals;
    int inc;
    for (i = nVals; (i > 0) && (srf08Err == aErrNone); i -= inc) {

      /* compute number of shorts for this pass */
      if (i > 3)
        inc = 3;
      else
        inc = i;

      /* request the next IIC read */
      data[0] = cmdIIC_RD;
      data[1] = (char)bit_IIC_RD_HOST;
      data[2] = (char)(srf08address + 1);
      data[3] = (char)(inc * 2);
      if (!aPacket_Create(stemLib, router,
                          4, data, &packet, &srf08Err))
        aStem_SendPacket(stemLib, packet, &srf08Err);

      /* now, seek the reply packet with the shorts */
      if (srf08Err == aErrNone) {
        if (!aStem_GetPacket(stemLib,
                             sSRF08IICDataFilter,
                             NULL,
                             aSRF08_GETDATATIMEOUTMS,
                             &packet,
                             &srf08Err)) {
          if (!aPacket_GetData(stemLib, packet, &module,
                               &length, data, &srf08Err)) {
            if (length != (inc * 2 + 2))
              srf08Err = aErrIO;
            else {
              int j;
              char* v = &data[2];
              for (j = 0; j < inc; j++) {
                *p = aUtil_RetrieveShort(v);
                v += 2;
                p++;
              }
            }
          }
        } else {
          /* if we timed out, there is not likely a SRF08 present */
          if (srf08Err == aErrTimeout)
            srf08Err = aErrNotFound;

        }
      }
    }
  }

  return srf08Err;

} /* aSRF08_GetRange */
```

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 * aSRF08_GetLight
 *
 * This routine initiates a sonar ranging from the addressed
 * SRF08 and returns light sensor value.
 *
 * This routine requires the IIC baud rate to be set to 400kHz
 * as the SRF08 cannot handle 1MBit communciation.
 */

aErr aSRF08_GetLight(aStemLib stemLib,
                    const unsigned char router,
                    const unsigned char srf08address,
                    unsigned char* pVal)
{
  aErr srf08Err = aErrNone;
  unsigned char module;
  unsigned char length;
  char data[aSTEMMAXPACKETBYTES];
  aPacketRef packet;

  /* check all the parameters and get set up */
  if (!stemLib || !pVal || !router)
    srf08Err = aErrParam;
  if ((srf08Err == aErrNone)
      && ((srf08address % 1)
          || (srf08address < 0xE0)
          || (srf08address == 0xFF)))
    srf08Err = aErrParam;

  /* set the memory pointer in the SRF08 */
  if (srf08Err == aErrNone) {
    data[0] = 0;
    data[1] = (char)aSRF08_MS;
    if (!aPacket_Create(stemLib, srf08address,
                       2, data, &packet, &srf08Err))
      aStem_SendPacket(stemLib, packet, &srf08Err);
  }

  /* now, wait for the reading and look for an IIC ACK error which
   * would indicate no SRF08 connected at that address */
  if (srf08Err == aErrNone) {
    if (!aStem_GetPacket(stemLib,
                        sSRF08IICErrFilter,
                        NULL,
                        aSRF08_READDELAYMS,
                        &packet,
                        &srf08Err)) {
      aPacket_Destroy(stemLib, packet, NULL);
      srf08Err = aErrNotFound;
    } else {
      if (srf08Err == aErrTimeout)
        srf08Err = aErrNone;
    }
  }

  /* now, if there are no errors, we can assume there is:
   *  a) no brainstem attached at all
   *  b) a functioning SRF08 at the requested address
   *  c) some other IIC device at that address
   */

  /* set the SRF08 register pointer to the light sensor data */
  if (srf08Err == aErrNone) {
    data[0] = 1;
    if (!aPacket_Create(stemLib, srf08address,
                       1, data, &packet, &srf08Err))
      aStem_SendPacket(stemLib, packet, &srf08Err);
  }
  /* request a read of the data */
  if (srf08Err == aErrNone) {
    data[0] = cmdIIC_RD;
    data[1] = (char)bit_IIC_RD_HOST;
    data[2] = (char)(srf08address + 1);
    data[3] = 1;
    if (!aPacket_Create(stemLib, router,
                        4, data, &packet, &srf08Err))
      aStem_SendPacket(stemLib, packet, &srf08Err);
  }

  /* now, seek the reply packet with the light sensor value */
  if (srf08Err == aErrNone) {
    if (!aStem_GetPacket(stemLib,
                         sSRF08IICDataFilter,
                         NULL,
                         aSRF08_GETDATATIMEOUTMS,
                         &packet,
                         &srf08Err)) {
      if (!aPacket_GetData(stemLib, packet, &module,
                           &length, data, &srf08Err)) {
        if (length != 3)
          srf08Err = aErrIO;
        else
          *pVal = (unsigned char)data[2];
      } else {
        /* if we timed out, there is not likely a SRF08 present */
        if (srf08Err == aErrTimeout)
          srf08Err = aErrNotFound;

      }
    }
  }

  return srf08Err;

} /* aSRF08_GetLight */
```

aCMPS03.h

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                                 */
/* file: aCMPS03.h                                                 */
/*                                                                 */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                                 */
/* description: Definition of cross-platform CMPS03 Compass module */
/*              interface routines.                                */
/*                                                                 */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                                 */
/*        docs: Documentation can be found in the BrainStem        */
/*             reference under the major topic "C", category       */
/*              aCMPS03                                            */
/*                                                                 */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                                 */
/* Copyright © 1997-2004. Acroname Inc.                            */
/*                                                                 */
/* This software is the property of Acroname Inc.  Any             */
/* distribution, sale, transmission, or re-use of this code is     */
/* strictly forbidden except with permission from Acroname Inc.    */
/*                                                                 */
/* To the full extent allowed by law, Acroname Inc. also excludes  */
/* for itself and its suppliers any liability, wheither based in   */
/* contract or tort (including negligence), for direct,            */
/* incidental, consequential, indirect, special, or punitive       */
/* damages of any kind, or for loss of revenue or profits, loss of */
/* business, loss of information or data, or other financial loss  */
/* arising out of or in connection with this software, even if     */
/* Acroname Inc. has been advised of the possibility of such       */
/* damages.                                                        */
/*                                                                 */
/* Acroname Inc.                                                   */
/* www.acroname.com                                                */
/* 720-564-0373                                                    */
/*                                                                 */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#ifndef _aCMPS03_H_
#define _aCMPS03_H_

#include "aStem.h"


#define aCMPS03_RADIANS               ((unsigned char)0)
#define aCMPS03_DEGREES               ((unsigned char)1)
#define aCMPS03_ROTATIONS     ((unsigned char)2)

#ifdef __cplusplus
extern "C" {
#endif

aErr aCMPS03_GetHeading(aStemLib stemLib,
                       const unsigned char router,
                       const unsigned char units,
                       float* pVal);

#ifdef __cplusplus
}
#endif

#endif /* _aCMPS03_H_ */
```

aCMPS03.c

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                                   */
/* file: aCMPS03.c                                                   */
/*                                                                   */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                                   */
/* description: Implementation of cross-platform CMPS03 Compass     */
/*              module interface routines.                           */
/*                                                                   */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                                   */
/*        docs: Documentation can be found in the BrainStem         */
/*             reference under the major topic "C", category        */
/*              aCMPS03                                              */
/*                                                                   */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                                   */
/* Copyright © 1997-2004. Acroname Inc.                              */
/*                                                                   */
/* This software is the property of Acroname Inc.  Any              */
/* distribution, sale, transmission, or re-use of this code is      */
/* strictly forbidden except with permission from Acroname Inc.     */
/*                                                                   */
/* To the full extent allowed by law, Acroname Inc. also excludes  */
/* for itself and its suppliers any liability, wheither based in   */
/* contract or tort (including negligence), for direct,             */
/* incidental, consequential, indirect, special, or punitive       */
/* damages of any kind, or for loss of revenue or profits, loss of */
/* business, loss of information or data, or other financial loss  */
/* arising out of or in connection with this software, even if     */
/* Acroname Inc. has been advised of the possibility of such       */
/* damages.                                                          */
/*                                                                   */
/* Acroname Inc.                                                     */
/* www.acroname.com                                                  */
/* 720-564-0373                                                      */
/*                                                                   */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#include "sys_defines.h"
#include "aCmd.tea"
#include "aStemMsg.h"
#include "aUtil.h"
#include "aCMPS03.h"


/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 * local defines
 */

#define aCMPS03_ADDRESS                      0xC0
#define aCMPS03_GETDATATIMEOUTMS     100



/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 * local prototypes
 */

static aBool sCMPS03IICErrFilter(const unsigned char module,
                              const unsigned char dataLength,
                              const char* data,
                              void* ref);
static aBool sCMPS03IICDataFilter(const unsigned char module,
                               const unsigned char dataLength,
                               const char* data,
                               void* ref);
```

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 * sCMPS03IICErrFilter
 */

aBool sCMPS03IICErrFilter(const unsigned char module,
                       const unsigned char dataLength,
                       const char* data,
                       void* ref)
{
  if ((dataLength > 1)
      && (data[0] == cmdMSG)
      && (data[1] == iicNoAck))
    return aTrue;

  return aFalse;

} /* sCMPS03IICErrFilter */



/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 * sCMPS03IICDataFilter
 *
 * This uses the hard-coded device ID of 52 which is valid for
 * the GP and Moto boards.  It should probably be more general.
 */

aBool sCMPS03IICDataFilter(const unsigned char module,
                       const unsigned char dataLength,
                       const char* data,
                       void* ref)
{
  aAssert(pAddr);

  if ((dataLength > 1)
      && (data[0] == cmdDEV_VAL)
      && (data[1] == 52))
    return aTrue;

  return aFalse;

} /* sCMPS03IICDataFilter */



/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 * aCMPS03_GetHeading
 *
 * This routine retrieves the compass heading and returns the
 * heading in the units requested.
 *
 * This routine requires the IIC baud rate to be set to 100kHz
 * as the CMPS03 cannot handle 1MBit or 400kHz communciation.
 */

aErr aCMPS03_GetHeading(aStemLib stemLib,
                       const unsigned char router,
                       const unsigned char units,
                       float* pVal)
{
  aErr cmpsErr = aErrNone;
  unsigned char module;
  unsigned char length;
  char data[aSTEMMAXPACKETBYTES];
  aPacketRef packet;

  /* check all the parameters and get set up */
  if (!stemLib || !pVal || !router)
    cmpsErr = aErrParam;
  if ((cmpsErr == aErrNone)
      && ((units != aCMPS03_RADIANS)
          && (units != aCMPS03_DEGREES)
          && (units != aCMPS03_ROTATIONS)))
    cmpsErr = aErrParam;

  /* set the memory pointer in the CMPS03 */
  if (cmpsErr == aErrNone) {
    data[0] = 2;
    if (!aPacket_Create(stemLib, aCMPS03_ADDRESS,
                       1, data, &packet, &cmpsErr))
      aStem_SendPacket(stemLib, packet, &cmpsErr);
  }

  /* now, wait for a bit and look for an IIC ACK error which
   * would indicate no CMPS03 connected at that address */
  if (cmpsErr == aErrNone) {
    if (!aStem_GetPacket(stemLib,
                        sCMPS03IICErrFilter,
                        NULL,
                        50,
                        &packet,
                        &cmpsErr)) {
      aPacket_Destroy(stemLib, packet, NULL);
      cmpsErr = aErrNotFound;
    } else {
      if (cmpsErr == aErrTimeout)
        cmpsErr = aErrNone;
    }
  }

  /* now, if there are no errors, we can assume there is:
   *  a) no brainstem attached at all
   *  b) a functioning CMPS03 at the requested address
   *  c) some other IIC device at that address
   */

  /* request a read of the data */
  if (cmpsErr == aErrNone) {
    data[0] = cmdIIC_RD;
    data[1] = (char)bit_IIC_RD_HOST;
    data[2] = (char)(aCMPS03_ADDRESS + 1);
    data[3] = 2;
    if (!aPacket_Create(stemLib, router,
                       4, data, &packet, &cmpsErr))
      aStem_SendPacket(stemLib, packet, &cmpsErr);
  }

  /* now, seek the reply packet with the light sensor value */
  if (cmpsErr == aErrNone) {
    if (!aStem_GetPacket(stemLib,
                        sCMPS03IICDataFilter,
                        NULL,
                        aCMPS03_GETDATATIMEOUTMS,
                        &packet,
                        &cmpsErr)) {
      if (!aPacket_GetData(stemLib, packet, &module,
                            &length, data, &cmpsErr)) {
        if (length != 4)
          cmpsErr = aErrIO;
        else {
          short raw = aUtil_RetrieveShort(&data[2]);
          switch (units) {
          case aCMPS03_RADIANS:
            *pVal = (float)(((float)raw / 3600.0f) * 2.0f * aPI);
            break;
          case aCMPS03_DEGREES:
            *pVal = (float)((float)raw / 10.0f);
            break;
          case aCMPS03_ROTATIONS:
            *pVal = (float)((float)raw /3600.0f);
            break;
          } /* switch */
        }
      } else {
        /* if we timed out, there is not likely a CMPS03 present */
        if (cmpsErr == aErrTimeout)
          cmpsErr = aErrNotFound;

      }
    }
  }

  return cmpsErr;

} /* aCMPS03_GetRange */
```