2011

Design, development, and testing of a multi-agent autonomous surface fleet for environmental applications

Daniel Davis Smith
Louisiana State University and Agricultural and Mechanical College, dsmi112@tigers.lsu.edu

Follow this and additional works at: https://digitalcommons.lsu.edu/gradschool_theses
Part of the Engineering Commons

Recommended Citation
https://digitalcommons.lsu.edu/gradschool_theses/2571

This Thesis is brought to you for free and open access by the Graduate School at LSU Digital Commons. It has been accepted for inclusion in LSU Master's Theses by an authorized graduate school editor of LSU Digital Commons. For more information, please contact gradetd@lsu.edu.
DESIGN, DEVELOPMENT, AND TESTING OF A MULTI-AGENT AUTONOMOUS SURFACE FLEET FOR ENVIRONMENTAL APPLICATIONS

A Thesis

Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Master of Science in Biological and Agricultural Engineering

in

The Department of Biological and Agricultural Engineering

by
Daniel Davis Smith
B.S., Louisiana State University, 1992
August 2011
ACKNOWLEDGMENTS

First and foremost I want to thank God for the gift of life, my family and the opportunity to serve. I wish to thank my family for their continuing support in all areas of life, but especially in my pursuit of my education. I want to express my gratitude to my advisor, Dr. Steven G. Hall, for his guidance, wisdom, and friendship over many years and to Dr. Richard Bengtson and Dr. James Wandersee who also served on my Master’s committee. I also want to thank the student workers who have worked alongside me on this project: Mr. Brian M. Thompson and Miss Dolly Hsu. In addition, special thanks are due to Mr. Tom McClure, shop manager for the Department of Biological and Agricultural Engineering for his patient instruction and assistance. I wish to thank Mr. Jack Chiu, fellow student and BAE Staff member, for the loan of equipment and parts during my research. Finally, I want to express my thanks to the faculty and staff of the Department of Biological and Agricultural Engineering whose support and encouragement have been so important to my success in my academic endeavors.
TABLE OF CONTENTS

ACKNOWLEDGMENTS ..................................................................................................................... ii

LIST OF TABLES .......................................................................................................................... v

LIST OF FIGURES ......................................................................................................................... vi

ABSTRACT ...................................................................................................................................... viii

CHAPTER 1: INTRODUCTION ........................................................................................................ 1

CHAPTER 2: DESIGN AND CONSTRUCTION OF A FLEET OF AUTONOMOUS SURFACE VEHICLES .................................................................................................................. 4
  2.1: Introduction and Literature Review .................................................................................... 4
  2.2: Materials and Methods ........................................................................................................ 6
    2.2.1: Modularity ..................................................................................................................... 6
    2.2.2: Physical ASV Design and Construction ..................................................................... 7
    2.2.3: Control System Design and Construction ................................................................. 13
    2.2.4: Control System Software Design ............................................................................... 18
  2.3: Results .................................................................................................................................. 19
  2.4: Conclusions ........................................................................................................................ 22

CHAPTER 3: DESIGN AND TESTING OF A COMMUNICATIONS NETWORK FOR MULTIPLE AUTONOMOUS SURFACE VEHICLES ........................................................................... 23
  3.1: Introduction and Literature Review .................................................................................... 23
  3.2: Materials and Methods ........................................................................................................ 24
    3.2.1: Intra-ASV Communications ....................................................................................... 24
    3.2.2: Inter-ASV Communications ...................................................................................... 27
  3.3: Results .................................................................................................................................. 31
    3.3.1: Intra-ASV Results ....................................................................................................... 31
    3.3.2: Inter-ASV Results ....................................................................................................... 31
  3.4: Data Analysis ....................................................................................................................... 33
    3.4.1: Intra-ASV.................................................................................................................... 33
    3.4.2: Inter-ASV .................................................................................................................... 34
  3.5: Conclusions ........................................................................................................................ 35

CHAPTER 4: DATA ANALYSIS OF SPATIAL AND ENVIRONMENTAL DATA FROM MULTIPLE AUTONOMOUS SURFACE VEHICLES ............................................................................. 37
  4.1: Introduction and Literature Review .................................................................................... 37
  4.2: Materials and Methods ........................................................................................................ 37
    4.2.1: Environmental Data .................................................................................................... 38
    4.2.2: Spatial Data ................................................................................................................ 38
  4.3: Results .................................................................................................................................. 39
  4.4: Data Analysis ....................................................................................................................... 41
  4.5: Conclusions ........................................................................................................................ 41

CHAPTER 5: CONCLUSION ............................................................................................................. 43
LIST OF TABLES

Table 1 - Cost breakdown for ASV control systems ................................................................. 20
Table 2 - Intra-ASV Command List ....................................................................................... 26
Table 3 - XBee distance testing shoulder height raw data ....................................................... 31
LIST OF FIGURES

Figure 1 - Original ASV Design ................................................................. 2
Figure 2 - ASV Subsystem list showing modularity .................................... 6
Figure 3 - Pontoon dimensions (inches) ....................................................... 7
Figure 4 - Pontoon after shaping but before coating and painting .................. 8
Figure 5 - Finished pontoon with aluminum sheet metal glued in place ............ 8
Figure 6 - Vertical frame element with dimensions (inches) ......................... 9
Figure 7 - Pontoons with frame elements mounted ...................................... 9
Figure 8 - Cross section of removable end cap design used for wiring connections 11
Figure 9 - Motor and Paddlewheel assembly ............................................ 12
Figure 10 - Detail of U-bracket for solar panels ....................................... 12
Figure 11 - Solar panels mounted on frame ............................................... 13
Figure 12 - Block diagram of original ASV control system .......................... 14
Figure 13 - New ASV Control System block diagram (red lines are power and black lines are data/control) ................................................................. 15
Figure 14 - TS-7260 ARM-based single board computer from Technologic Systems 16
Figure 15 - Garmin 16x HVS GPS Module ................................................. 17
Figure 16 - BASIC Atom Pro software design flow chart ............................ 18
Figure 17 - ARM Software design flow chart ............................................ 19
Figure 18 - Completed Autonomous Surface Vehicle ................................ 21
Figure 19 - Wiring diagram for ASV systems .......................................... 21
Figure 20 - Diagram of ASV fleet communications .................................... 23
Figure 21 - Intra-ASV sentence structure .................................................. 25
Figure 22 - Main test loop on ARM for Intra-ASV Communications tests ....... 27
Figure 23 - Inter-ASV Sentence Structure .............................................. 28
Figure 24 - XBee Distance testing apparatus

Figure 25 - Inter-ASV Communication system testing - raw data (shoulder height = 1.5 meters)

Figure 26 - Inter-ASV communication system distance testing - 480 meter study raw data

Figure 27 - Inter-ASV communication system distance testing averages of 1.5 meter height readings with 2nd degree poly fit show 90% reliability to 300 meters

Figure 28 - Sample GPS log file output

Figure 29 - Sample map showing GPS track log points with a simple poly line joining them in time sequence

Figure 30 - Statistical analysis of data on bird predation reduction on catfish ponds after Price and Hall 2011

Figure 31 - Sample map showing ASV track buffered using an ArcGIS model
ABSTRACT

As costs have decreased and both computational complexity and robustness have increased, the use of autonomous vehicles in real-world environments has increased dramatically. The development of a fleet of autonomous surface vehicles able to coordinate their actions through communications provides a significant tool for numerous water-based applications such as the reduction of predatory birds on aquaculture ponds, tracking of pollutant gradients, and water quality mapping applications.

A fleet of three autonomous surface vehicles (ASVs) was developed using the best characteristics from earlier designs. Each vehicle is a dual-pontoon, dual-paddlewheel design powered by batteries recharged using a vehicle-mounted solar array that produces a peak output of 30 Watts. The control system consists of two microcontrollers: a TS-7260 ARM-based microcontroller board that handles high-level functions such as navigation, the collection, storage, and analysis of data, and communication; and a BASIC Atom Pro that handles motor control.

A major design goal was modularity. This allows for quick and easy field repairs and upgrades. Communication is essential for fleet success. The dual-microcontroller system in these ASVs has two levels of communication. Intra-ASV communications are handled via serial connections between the ARM and the BASIC Atom Pro on each ASV, whereas inter-ASV communications use XBee Radio Modules with an approximate range of 300 meters. Through the use of relaying, we have an effective range of 600 meters across the fleet of three ASVs. Longer-ranges are possible with other radios. It is desirable to know at all times where the ASV is both with respect to the other ASVs in the fleet and to the data being collected. By collecting and
storing GPS coordinates on a regular basis and especially when a sample is taken, we have the ability to map the data being collected. Maps were constructed demonstrating the potential 50-80% reduction in birds.

The development of the fleet of ASVs provides a novel, inexpensive, highly configurable, mobile platform for experimentation. Future research possibilities exist of significant importance including: gradient tracking of pollutants for both point source and non-point source pollutants; coastal applications including salinity mapping and bathymetry mapping; ecosystem monitoring; biosecurity applications; and others.
CHAPTER 1: INTRODUCTION

Once relegated to research laboratories and classrooms because of their limited usefulness and fragility, autonomous vehicles are becoming more commonplace in research, education, and military applications as the complexity of their control systems and the robustness of their mechanical systems improve. A great deal of research and development has occurred in the area of land-based autonomous vehicle as evidenced by the success of the Defense Advanced Research Projects Agency’s (DARPA) 2004 and 2005 Grand Challenges (Seetharaman et al, 2006) and the Urban Challenge which followed those in 2007. By comparison, relatively little research has been done in the area of autonomous water-based vehicles and much of what has been done is focused on submersible vehicles. Closer to Biological Engineering, autonomous vehicles are finding a home in agriculture (Hall and Price, 2003; Kim et al., 2000; Lindgren et al, 2002; Noguchi et al., 1999; Tang et al., 2000; Wang et al., 2004; Jeon et al., 2009). In all of these areas of research, most of the vehicles operate alone to complete the given task.

This is also how the ASVs designed and built at Louisiana State University AgCenter by Hall and Price laboratories starting early in the 21st century operated (Hall, et al. 2006). With the first ASV being constructed in 2001 (Hall, et al, 2001; Price and Hall, 2002), these ASVs were designed to reduce bird predation on aquaculture ponds in an environmentally and ecologically friendly way at a relatively low cost. For the most part, they accomplished this by simply moving around the ponds and startling the birds off of the surface. These vehicles have always had the ability to be used as mobile sensor platforms, and in fact, from the beginning had environmental sensors on board in the form of global positioning systems (GPS); ambient light
sensors; and, on early ASVs, bump sensors (Figure 1). These sensors provided a way for the ASV to interact with its environment, but did not directly monitor the environmental quality of the water body they were operating within.

Development of the ASVs at LSU continued and new applications were found. Some of the first uses of the ASVs to measure environmental parameters began around 2005 when they were used to sample water quality in both ponds and coastal areas (Hall et al, 2005; Hall et al, 2006). Some research by undergraduate senior design teams has also been done on mounting environmental sensors on individual ASVs for nitrate and bathymetry measurement. Other experiments were done with enhancing the ASVs’ ability to “see” the birds using digital cameras and image analysis using neural networks to locate birds sitting on the water’s surface (Nadimpalli et al, 2006; Hall et al, 2007b). Other sensors and actuators have been added to the ASVs in an attempt to enhance their predation reduction mission. In one version, a pump was used to shoot pond water at a bird that it had located using an infrared sensor. Research has continued in both the environmental monitoring and predation reduction arenas to the current time (Hall et al, 2007a; Hall, et al, 2009; Price and Hall, 2011).

Current research seeks to increase the usefulness of the ASVs especially as mobile environmental monitoring platforms by creating a fleet of autonomous surface vehicles that can communicate with each other and use intelligent decision making to coordinate their actions. This coordination of efforts could also be used to increase the effectiveness of multiple ASVs used to reduce bird predation on large aquaculture ponds as well as gather large amounts of environmental data quickly. Environmental data can be evaluated in real-time by the ASV fleet.
so applications such as gradient tracking of pollutants or tracking of non-point source pollutants become a possibility. A fleet of autonomous surface vehicles (ASVs) provides a unique platform for conducting this research with low cost and good reliability.

The first step in creating such a fleet is the design process. In designing the new fleet, the best features of all of the previous ASVs were incorporated into the new design. The addition of an additional microcontroller and the ability to communicate brought a new level of complexity to the design both at the hardware and software levels. To date three ASVs have been constructed. Since communications is essential for the ASVs to be able to cooperate with each other, communications hardware was incorporated into the design. This addition meant that a grammar and sentences had to be developed to enable communication within the ASV between the two microcontrollers and between the various ASVs in the fleet. Reliability testing was performed on both communication systems and distance testing was performed on the communication system used between the ASVs.

Once a fleet of ASVs has been designed, constructed and tested, it is then possible to begin using the fleet in a number of application areas including environmental monitoring and aquacultural systems. Preliminary testing was done to ensure that the fleet functioned as expected in real-world situations. Initial data was collected and analyzed.

The creation of a low-cost, reliable fleet of autonomous surface vehicles represents a unique research platform for conducting a wide range of experiments in aquatic environments. Two additional vehicles are proposed to expand the fleet to a total of five ASVs. New sensor systems are under investigation for their applicability to the current research project.
2.1: Introduction and Literature Review

As discussed previously, though originally designed for the reduction of bird predation on aquaculture ponds (Price et al, 2001, 2002 and 2011), the application of the autonomous vehicles developed at LSU to water quality and environmental monitoring applications was apparent from the beginning (Hall and Price, 2003; Hall et al., 2006). Early experiments in the environmental arena involved using stand-alone data-logging equipment with attached sensors that were carried on an ASV as it traversed the area being studied (Hall et al, 2006). One attempt at enhancing the environment using the autonomous ASVs was an application of the ASVs’ ability to scare birds away from the water when, in 2007, an ASV was used to scare birds off of a drinking water reservoir thus reducing the amount of avian fecal material in the drinking water supply (Hall et al, 2007a). One early experiment by a Biological & Agricultural Engineering undergraduate senior design team that attempted to do environmental monitoring involved attaching a nitrate sensor to a ASV and having it map nitrate levels in the water of a pond by taking readings at specified intervals as it traveled a random path back and forth across the water. A more recent effort by another senior design team built a mechanism to take depth measurements and associate those with GPS coordinates to allow for the monitoring of sediment accretion. Almost all of these experiments were performed with a single ASV on the water and the few that had multiple vehicles had no method for coordinating the actions of the vehicles.

Numerous researchers have studied robotic, autonomous, or remote control vehicles. In the biological engineering arena, Tang et al. (2000) and Jeon et al. (2009) worked on autonomous vehicles for precise weed control. Another biological engineering application includes an
unmanned tractor (Kim et al., 2000). Others have worked on methods of controlling these agricultural robots including position determination using a simple odometric model (Lindgren, et al., 2002), vision systems (Noguchi, et al. 1999 and Broggi et al, 2000) and the application of neural networks to robotic mowers (Wang, et al. 2004).

In the area of water-based autonomous vehicles, the majority of the research that has been done has involved single vehicles and much of it involved underwater vehicles instead of surface vehicles. Dudek et al. (2007) constructed an amphibious autonomous robot capable of traversing land, the sea floor or the sea surface that used various sensors, including arrays of microphones, to localize its position (also Prahacs et al., 2005; Liu and Milios, 2005). It was based on earlier work on a land-based hexapod robot (Saranli, et al. 2001). AQUA EXPLORER 2 (Kojima et al., 2002) is an underwater autonomous vehicle for performing cable inspections. Thakoor et al. (2004) and Paulson, L.D. (2004) both looked at the usefulness of incorporating biomemetics into robot designs. A number of researchers have written about the application of vision systems to water-based autonomous vehicles including underwater vehicles (Horgan and Toal, 2006) and surface vehicles (Nadimpalli et al., 2006). Foresti et al, (1999) discussed neural network applications to autonomous underwater vehicle guidance. Relatively little has been written on coordinating the activities of multiple autonomous surface vehicles. Benjamin et al. (2006) discusses a method for avoiding collisions between autonomous surface vehicles and Fahimi (2007) modeled a method for formation control for groups of ASVs. More recently, Higinbotham, et al (2008) tested a fleet of ocean going ASVs while Arrichiello et al. (2010) used two ASVs connected with a rope to capture a floating object.
2.2: Materials and Methods

In creating a standard design for the fleet of autonomous surface vehicles (ASVs), the best traits from the various stand-alone ASVs were incorporated into the new design. Some of those traits are: dual paddlewheels for propulsion, adjustable solar panels, dual closed-cell Styrofoam® pontoons, and aluminum frame components. In addition to the traits utilized from the earlier designs, the goal was to create a set of ASVs that were as close as possible to being identical. This made modularity a major design goal. The result is an easily reproducible, inexpensive dual paddlewheel ASV that is ideal for large numbers of aquacultural and environmental applications.

2.2.1: Modularity

Stressing modularity in the design at all levels provides a number of benefits, especially in the areas of repair, experimental repeatability, and control.

Deconstructing the ASV design into its various subsystems provides the first level of modularity in the design process. The second level of modularity occurs within the various subsystems (Figure 2).

Utilizing identical parts allows for a broken part to be quickly replaced from a stock of spare parts that are generic to any ASV in the fleet. Because the ASVs are identical, the experimental results obtained with one should be easily replicable with any other ASV in the fleet. This also means that the relative position of an ASV in a group of ASVs performing some task should not affect the outcome of the task. Lastly, since the ASVs are all identical within the limitations of the
individual components, control system programming is greatly simplified since the same programs run on all the ASVs without modification.

2.2.2: Physical ASV Design and Construction

The physical ASV encompasses all of the subsystems of the ASV except the control subsystem. The design of the ASV begins with designing the frame and floatation. The Frame/Floatation subsystem contains three sub elements: the pontoons, the vertical frames, and the electronics enclosure. The floatation for the ASV consists of twin pontoons with dimensions as shown in Figure 3.

![Figure 3 - Pontoon dimensions (inches)]
The pontoons are constructed from Styrofoam® insulation by first rough cutting the general shape and size of the pontoon from a 4’ x 8’ sheet of the insulation. Then the rough pontoon is shaped using a hot-wire cutting system and plywood guides. The result of this process is seen in Figure 4. To increase the durability of the pontoons, they are coated with multiple thinly applied coats of Steve’s Foam Coat® (www.fxsupply.com). After the last coat of the Foam Coat has set, a 7.5” x 32” (19 cm x 81 cm) piece of 18 gauge 2024 aluminum sheet metal is glued to the top side of the pontoon using Liquid Nails® construction adhesive to provide a surface to which the frame pieces may be securely attached. In addition, a small (1” x 6.25” / 2.5cm x 16cm) piece of the same sheet metal was attached along the center of the side of the pontoon towards the centerline of the ASV to provide an additional mount point for the motors. After the sheet metal was glued into place, it was taped off with masking tape and the pontoons were painted a light green color using Valspar™ spray paint. The final pontoons (without the small piece of sheet metal) are shown in Figure 5.

The frames of the ASVs were constructed from 6061 Aluminum square tubing with a 0.75” (1.9 cm) nominal cross section and 1/8” (0.125” / 3 mm) wall thickness welded together. There are two vertical frame pieces for each ASV that attach to the pontoons using four sheet metal screws.
Each frame piece consists of two horizontal tubes connected by two vertical tubes as shown in Figure 6.

![Figure 6 - Vertical frame element with dimensions (inches)](image)

The longer horizontal tube is attached to the two pontoons using four #8 x 1.5” (4 cm) sheet metal screws. The shorter horizontal tube provides the mounting points for the two solar panels. The welds and ends of the tubes were ground to a 1 mm chamfer to reduce the risk of injury from sharp edges. Figure 7 shows the ASV with the vertical frame pieces mounted on the pontoons.

The final pieces of the frame are two rectangular pieces of the 2024 sheet aluminum that are bent 90 degrees along their long dimension. They then each have two cuts made in the vertical side of the inverted L-shaped bracket, and have two 45 degree bends applied to form the platforms for
the GPS antenna and the XBee antenna. The platforms are mounted fore and aft of the solar panels by bolting them to the underside of the top horizontal tube of the frames with #8 x 1" (2.5 cm) bolts and nuts.

The electronics enclosure is constructed of a 36” (90 cm) long section of sewer pipe with a 6” (15.25 cm) outer diameter (OD). The pipe has an inner diameter (ID) of 5.44” (14 cm). Two custom end caps were created for the electronics enclosure tubes. Both end caps were constructed using an outer ring created by cutting the rounded end off of a standard PVC end cap and gluing that with PVC cement to a 6.75” (17 cm) diameter circle cut from a flat sheet of PVC 1/8” (0.125” / 3 mm) thick. On the end of the electronics enclosure that had no wiring, the cap was glued into place using PVC cement.

For the end cap that had panel-mount connectors for wiring connections into the enclosure (Figure 8), additional construction steps were needed to create the end cap (Figure 8). To the inside of this cap, a smaller ring of PVC was glued leaving ¼” (0.25” / 6 mm) between the inner and outer rings. This smaller ring was created from a slice of standard PVC pipe that had had a 1½” (38 mm) piece removed from the slice. The slice was then bent to form a ring that fit inside the sewer pipe leaving the ¼” (0.25” / 6 mm) space desired and glued into place using PVC cement. Into the ¼” (0.25” / 6 mm) space between the inner and outer rings, an O-ring was inserted to increase the water resistance of the enclosure. The removable end cap is held in place by two nylon straps attached to either side of the cap by screws. At the end of the straps is half of a locking clasp. The other end of each clasp is attached to each side of the sewer pipe with small bolts that are coated in silicon to prevent leaking.
The Propulsion subsystem provides all locomotion to the ASV and is comprised of three subelements: the motors, the motor mounts, and the paddlewheels. The motors used in the design are Pittman 19.1 volt DC motors. They are attached with #10 x ¾” (19 mm) sheet screws to a custom built motor mount constructed of aluminum.

The paddlewheels for the ASVs were created by modifying the molded plastic wheels from a standard hand-propelled fertilizer spreader. The outer portion of each wheel was cut off using a band saw to leave the inner structure of the wheel seen in Figure 9. The dimensions of the final paddlewheel are approximately 8.5” in diameter and 2.4” thick. This structure makes a very effective paddlewheel.

---

Figure 8 - Cross section of removable end cap design used for wiring connections
Power for the systems of the ASV is supplied by the two sub elements that form the power subsystem. These are the batteries and the solar panels. The batteries in use on the ASV are 12 volt, 7 amp-hour (Ah) sealed lead-acid batteries (UB1270 or equivalent). Two batteries are used in parallel resulting in 12 volts with between 14 and 16 Ah total capacity. The ASV uses two solar panels taken from a three-panel kit from Harbor Freight (Item number 90599). Each panel supplies 12 volts at a maximum current of 1.25 amps each under optimal sunlit conditions. With the two panels connected in parallel a maximum current of 2.5 amps is available. The solar panels are also connected in parallel with the batteries to provide charging for the batteries during daylight hours.

The solar panels are joined to each other with two cabinet hinges and are attached to custom made U-shaped brackets using hinges along the outer edge. These U-shaped brackets are machined from solid aluminum block and then tapped with two holes for the hinge and a hole on the bottom of the bracket to accommodate a setscrew. The brackets hold the solar panels to the top horizontal element of the vertical frames (Figure 10) while allowing for the easy removal or
angling of the solar array. In Figure 11, the solar panels are shown mounted on the frame elements.

![Image of solar panels mounted on frame](image.png)

Figure 11 - Solar panels mounted on frame

### 2.2.3: Control System Design and Construction

Designing the control system began with analyzing the advantages and disadvantages of the original ASV control system in light of the more complex types of problems that were to be addressed in the new ASV fleet. Some of the requirements for the new control system included: the ability to perform floating-point calculations for navigation and data analysis, storage of data on a conveniently removable device, ability to handle multiple sensors, and the ability to communicate with the other ASVs in the fleet to enable coordinated task completion.
The original ASV control system (Figure 12) consisted of a single BASIC Stamp 2 series microcontroller from Parallax, Inc. The major advantages to the original BASIC Stamp microcontroller are:

- simplicity of programming,
- existing library of software, and the built-in software to produce the pulse-width modulation signals used to control the motors.

Some of the limitations of this microcontroller are its limited processing speed, inability to perform floating point mathematics, and very limited memory for both program and data storage. The original control system also had nothing in its design to allow for communication between ASVs.

The new control system (Figure 13) is designed around two microcontrollers: a high-level microcontroller responsible for communication, navigation, data analysis and data storage; and a low-level microcontroller responsible for motor control, monitoring certain ASV system parameters, and emergency recovery functions. For the low-level microcontroller, the BASIC Atom Pro 24-M (BAP) from Basic Micro was chosen to replace the BASIC Stamp shown in Figure 10. This 24-pin microcontroller is pin-compatible with the BASIC Stamp 2 (BS2) series microcontrollers and the version of BASIC that it uses is very similar to that of the BS2 series.
making porting existing software relatively simple. There are several advantages to using the BAP, including: hardware-based serial communication (the BS2 series implements serial communication using software), twice the program storage, and a five-fold speed advantage over the BS2s.

Figure 13 - New ASV Control System block diagram (red lines are power and black lines are data/control)
The BASIC Atom Pro is connected to the ARM via a hardware serial port connection. In addition, the BAP is connected to two motor controllers. These motor controllers interpret the pulse wave modulated (PWM) signal sent by the BAP and adjust the speed and direction of the motors appropriately.
The TS-7260 single-board computer (Figure 14) from Technologic Systems, Inc. was chosen as the high-level microcontroller. This system is based on a 32-bit ARM processor and runs an embedded version of the GNU/Linux operating system. The term ARM refers to the architecture of the processor and was originally created by Acorn Computers where it stood for Acorn RISC (Reduced Instruction Set Computer) Machine. Later, when Acorn spun off the division that made the ARM processors, the ARM acronym came to stand for Advanced RISC Machine. ARM Holdings licenses the technology to a number of companies who create processors based on their architecture.

The processor on the TS-7260 is the Cirrus EP9302 ARM9 CPU. The board has a number of hardware features that are important. It has two USB 2.0 compliant ports, one of which is used in the ASV design to store data on removable USB Flash drives. There are three on-board RS-232 serial ports that are used for communicating with the GPS receiver, the BAP, and the XBee radio. Lastly, it has a number of general-purpose input/output pins for control outputs and sensor inputs. It is programmed using the C programming language.

The radios chosen for the ASVs were the XBee Pro Series 2 from Digi International. They operate in the 2.4 GHz frequency spectrum. The primary reasons that they were chosen are the availability of multiple versions with different power and antenna options that are all pin compatible with each other and the built-in networking software that handles most of the low-level network setup and routing automatically. Since several different power levels are available...
within this series, the current systems can be upgraded at a future date to enhance the fleet’s range of communication.

The control system uses information that it receives from three external sources to determine what actions to take. These are: the Global Positioning System sensor (GPS), the attached environmental sensors, and data received from the other ASVs in the fleet. Of these, the single most important sensor is the GPS since without it navigation becomes impossible.

The GPS sensor (Figure 15) chosen for the ASV control system was the Garmin GPS 16x HVS from Garmin Ltd. The HVS in the model name indicates that this GPS unit is designed to run on unregulated direct current (DC) voltages ranging from 8 to 30 volts. This means that it can be connected directly to the ASVs power subsystem without the need for additional power regulation. It utilizes a 12-channel receiver that allows it to track up to 12 GPS satellites simultaneously. This, along with the correction available from the Wide Area Augmentation System (WAAS) enables the GPS to provide a location that is within 3 meters of the actual ASV location 95% of the time in a typical operational scenario. In practice, we have found that on open water with an unobstructed view of the sky, positions are typically accurate to within one meter. The GPS is connected to the ARM microcontroller using the first serial port (COM 1) on the ARM board. Since COM 1 is normally the console port for the ARM microcontroller, the transmit data pin is not connected to the GPS. This prevents spurious output to COM 1 from another program inadvertently reprogramming the GPS.
2.2.4: Control System Software Design

The software for each ASV consists of two separate programs running on the two microcontrollers on that ASV. The program running on the BASIC Atom Pro is responsible for low-level functions such as direct motor control, limited system monitoring, and, in the future, emergency recovery operations. The ARM software provides the primary control of the ASV. It is responsible for all communication with other ASVs in the fleet, for problem solving, and for navigational choices. In addition, the software of each ASV in the fleet interacts with that of all of the other ASVs making the software system much more complex.

![Flow Chart](image)

**Figure 16 - BASIC Atom Pro software design flow chart**

Figure 16 shows a simplified flow chart of the BAP software design. The small orange hexagons with “A<>B” in them represent points at which communication occurs between the ARM and BAP. Looking at the flow chart, we observe that the BAP software consists of a loop that runs continuously with no exit conditions. The BAP must always be active for the ASV to operate since it controls the motors according to the directions given it by the ARM.
The ARM flowchart (Figure 17) only shows the main program execution loop during normal operation. It appears to be an endless loop like the BAP, but exit conditions do exist and are just not shown in order to simplify the diagram. Some examples of exit conditions are reaching the programmed goal (i.e. finding pollutant) or receiving an override signal from the land-based station.

Again, the “A<>B” icons represent communication with the BAP. In addition to communicating with the BAP, communication with the other ASVs in the fleet occurs on a regular basis so that every ASV in the fleet knows the location and most recent environmental sensor reading of every other ASV.

### 2.3: Results

Beginning in March 2010, three ASVs were constructed using the above techniques. The cost for each control system was approximately $800.00 and includes all of the electronic and electrical components used in their construction (Table 1). The approximate costs for the aluminum tubing, aluminum sheet metal, foam pontoons, foam coating, and paint used in the
frame and floatation systems were under $200.00 for each ASV. This resulted in a total construction cost of approximately $1000.00 per ASV.

Using the volume obtained from the CAD drawing of a pontoon of ~1,320 in$^3$ (21,630 cm$^3$ or 21.6 liters) each, gives us a total of 2,640 in$^3$ (43,260 cm$^3$ or 43.26 liters) of foam for each ASV. Since the foam is less dense than water it floats on the surface, and the total floatation provided by the pontoons is equal to their displacement. A volume of 2,640 in$^3$ (43,260 cm$^3$ or 43.26 liters) of water is equal to 11.43 gallons (43.26 liters). At 8.35 pounds per gallon (2.2 kg per liter), gives us a total weight of 95.44 pounds or 43.26 kg. Since the weight of water displaced is equal to the amount of weight the pontoons can float (including their own weight), we have a total displacement of 95.19 pounds. The weight of a fully loaded ASV (Figure 18) is 60 pounds giving us approximately 35 pounds of ‘extra’ floatation for adding sensors, additional batteries or other equipment to the ASV. The ASV wiring was constructed as indicated in Figure 19.

<table>
<thead>
<tr>
<th>Component</th>
<th>Unit Cost</th>
<th>Qty</th>
<th>Extended Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>TS-7260 - ARM</td>
<td>$250.00</td>
<td>3</td>
<td>$750.00</td>
</tr>
<tr>
<td>BASIC Atom Pro</td>
<td>$70.00</td>
<td>3</td>
<td>$210.00</td>
</tr>
<tr>
<td>Super Carrier Board</td>
<td>$20.00</td>
<td>3</td>
<td>$60.00</td>
</tr>
<tr>
<td>Garmin 16x HVS</td>
<td>$110.00</td>
<td>3</td>
<td>$330.00</td>
</tr>
<tr>
<td>Motor Controller</td>
<td>$25.00</td>
<td>6</td>
<td>$150.00</td>
</tr>
<tr>
<td>5V Regulator</td>
<td>$13.00</td>
<td>3</td>
<td>$39.00</td>
</tr>
<tr>
<td>3.3 V Regulator</td>
<td>$13.00</td>
<td>3</td>
<td>$39.00</td>
</tr>
<tr>
<td>Xbee Radio</td>
<td>$30.00</td>
<td>3</td>
<td>$90.00</td>
</tr>
<tr>
<td>Sipex</td>
<td>$2.00</td>
<td>3</td>
<td>$6.00</td>
</tr>
<tr>
<td>Logic Level Converter</td>
<td>$2.00</td>
<td>3</td>
<td>$6.00</td>
</tr>
<tr>
<td>Thermocouple Amp</td>
<td>$12.00</td>
<td>3</td>
<td>$36.00</td>
</tr>
<tr>
<td>Board for Thermo Amp</td>
<td>$3.00</td>
<td>3</td>
<td>$9.00</td>
</tr>
<tr>
<td>Motors</td>
<td>$15.00</td>
<td>6</td>
<td>$90.00</td>
</tr>
<tr>
<td>Solar Panels</td>
<td>$60.00</td>
<td>6</td>
<td>$360.00</td>
</tr>
<tr>
<td>Batteries</td>
<td>$16.00</td>
<td>6</td>
<td>$96.00</td>
</tr>
<tr>
<td>Wiring, connectors, etc</td>
<td>$40.00</td>
<td>3</td>
<td>$120.00</td>
</tr>
<tr>
<td><strong>Total Cost for All Electronics</strong></td>
<td></td>
<td></td>
<td><strong>$2,391.00</strong></td>
</tr>
<tr>
<td><strong>Total Cost per AVS</strong></td>
<td></td>
<td></td>
<td><strong>$797.00</strong></td>
</tr>
</tbody>
</table>
Each ASV, as constructed, collects GPS data and environmental data and communicates that data to the other ASVs in the fleet to allow for coordination of actions to complete a given task. The resulting data files provide a means of analyzing the environmental data both temporally and geospatially.
2.4: Conclusions

The fleet of three ASVs constructed to date provides an inexpensive and versatile platform for water-based experiments. The addition to the single, stand-alone ASV design of a method for communication between the various ASVs in the fleet greatly enhances both the usefulness for applications already being researched and opens new avenues of exploration. The data resulting from these experiments will provide new ways of locating problems in various environments and in tracking changes in the ecosystems where the ASVs operate. Additional ASVs will be added to the fleet in the future in order to explore the coordination and effectiveness of five or more ASVs tasked with a single problem.
CHAPTER 3: DESIGN AND TESTING OF A COMMUNICATIONS NETWORK FOR MULTIPLE AUTONOMOUS SURFACE VEHICLES

3.1: Introduction and Literature Review

The concept of a ‘fleet’ of autonomous surface vehicles (ASVs) implies the existence of some level of cooperation and coordination between the members of the fleet and that requires a way for them to coordinate their activities (Figure 20). This coordination is facilitated by an inter-ASV communication system utilizing radios that allow each ASV to communicate digital data with every other ASV in the fleet. The design of this system is discussed including some of the design challenges that were dealt with and the construction of messages to be sent using the system. In addition, initial testing was done and the results showing excellent reliability in message passing and long-distance communication are presented and discussed.

A number of researchers have worked in the area of communications between multiple ASVs and between ASVs and shore-based control stations. Ghabcheloo, et al. (2009) describe their work on developing a system for simulating the coordination of the actions of multiple vehicles in the presence of communication dropouts and delays. Also in 2009, Martins et al presented a protocol for communicating between multiple vehicles and communicating sensors.
Since the control system design for the ASVs utilizes two microcontrollers that must work together, a second communication system is required to handle transferring commands and data between the two microcontrollers within an ASV. The hardware and software challenges of this system are discussed. The list of currently allowed messages is presented as well as initial test data which shows the system to be highly reliable at transferring messages within the ASV.

3.2: Materials and Methods

Two communication systems exist on each ASV within the fleet: an intra-ASV system and an inter-ASV system. Each system has unique hardware and software requirements. The intra-ASV communication system consists of a serial connection between the ARM microcontroller and the BASIC Atom Pro (BAP) while the inter-ASV system consists of an XBee radio module connected to a serial port on the ARM microcontroller.

Messages sent and received in both systems are ‘sentences’ composed of ASCII (American Standard Code for Information Interchange) characters with no control characters. Taking a cue from the National Marine Electronics Association (NMEA), the messages have a start character, a dollar sign ‘$’ (ASCII 36) and consist of a four character command/message followed by a comma delimited list of parameters and terminated by a semi-colon ‘;’ (ASCII 59). Neither the start character nor the termination character is allowed to appear anywhere else in a sentence. The specific sentence structure is discussed in more detail in the discussion of the particular communication system.

3.2.1: Intra-ASV Communications

The intra-ASV communication system allows the ARM and the BAP to communicate. The ARM transmits messages to the BAP instructions on how the motors should be controlled based on the navigation calculations that the ARM has made. The BAP, in turn, sends messages to the
ARM indicating the status of the ASV subsystems that it is responsible for. Any message sent from one microcontroller to the other is echoed back to the originating microcontroller to verify correct reception of the message. In the event of an incorrect echo, the message is resent.

Intra-ASV communication occurs between the second serial port (COM 2) on the ARM board and the hardware UART (Universal Asynchronous Receiver/Transmitter) on the BAP. The hardware serial port on the BAP uses pins 14 and 15 on the chip. The existence of hardware-based serial communication is the primary reason for switching from a BASIC Stamp to the BAP in the design. Hardware-based serial is more reliable than the software-based serial the Stamp uses. Both serial ports are configured for 300 baud, 8-bit data, no parity bits, and one stop bit. No flow control is used in this system. The COM ports on the ARM are RS-232 level serial ports, meaning that the voltage levels range from -12 volts to +12 volts, while the BAP serial port uses TTL (Transistor-Transistor Logic) levels of 0 volts to +5 volts. To convert from one to the other, a Sipex SB3232 chip is used to convert the voltage levels from RS-232 levels to TTL levels and back.

The sentence structure for Intra-ASV communications is shown in Figure 21. A message consists of a total of 10 characters including the start and termination characters. The starting ‘$’ is followed by one of the four character commands listed in Table 2, then a comma ‘,’ a three digit (0-9) parameter, and the terminating semi-colon ‘;’. Each command has the numeric parameter, but not every command utilizes it in which case, zeros are sent. For several commands, zeros are transmitted as placeholders so that every sentence is exactly ten characters long. The $SYNC

<table>
<thead>
<tr>
<th>Format:</th>
<th>$&lt;1&gt;,&lt;2&gt;;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Where:</td>
<td>$</td>
</tr>
<tr>
<td></td>
<td>- Start Character</td>
</tr>
<tr>
<td>&lt;1&gt;</td>
<td>- Command</td>
</tr>
<tr>
<td>,</td>
<td>- Delimiter</td>
</tr>
<tr>
<td>&lt;2&gt;</td>
<td>- Parameter</td>
</tr>
<tr>
<td>;</td>
<td>- Termination Character</td>
</tr>
</tbody>
</table>

Example: $SYNC,000;

Figure 21 - Intra-ASV sentence structure
command was added to the command list because it was discovered during testing that even though the BAP would often receive the correct message the first time it was sent, the ARM would not receive the echo correctly.

Table 2 - Intra-ASV Command List

<table>
<thead>
<tr>
<th>Command</th>
<th>Command Description</th>
<th>Parameter Description</th>
<th>Originator</th>
</tr>
</thead>
<tbody>
<tr>
<td>$RTRN,000;</td>
<td>Right Turn</td>
<td>Angle in degrees to turn</td>
<td>ARM</td>
</tr>
<tr>
<td>$LTRN,000;</td>
<td>Left Turn</td>
<td>Angle in degrees to turn</td>
<td>ARM</td>
</tr>
<tr>
<td>$FRWD,000;</td>
<td>Forward</td>
<td>Percent of full-speed</td>
<td>ARM</td>
</tr>
<tr>
<td>$BACK,000;</td>
<td>Backward</td>
<td>Percent of full-speed</td>
<td>ARM</td>
</tr>
<tr>
<td>$STOP,000;</td>
<td>Stop</td>
<td>Placeholders</td>
<td>ARM</td>
</tr>
<tr>
<td>$LGHt,000;</td>
<td>Ambient Light Data</td>
<td>000 = Dark, 001 = Light</td>
<td>Atom</td>
</tr>
<tr>
<td>$VOLT,000;</td>
<td>Voltage Data</td>
<td>Integer battery voltage</td>
<td>Atom</td>
</tr>
<tr>
<td>$SYNC,000;</td>
<td>Used to sync COM Ports</td>
<td>Placeholders</td>
<td>ARM / Atom</td>
</tr>
</tbody>
</table>

To allow the ARM and BAP to synchronize the timing and buffering between them in order to ensure correct reception of the actual messages, $SYNC commands are sent at the beginning of any transmission from one to the other. This continues until the correct echo is received at which point the serial ports are in sync with each other and the real message can be safely sent.

Upon receipt of any command from the ARM, the BAP echoes the command back to the ARM and then attempts to parse the command. If the command received is $SYNC, then nothing is done. For any of the other commands, one or more actions are taken by the BAP to cause the ASV to perform whatever action was indicated in the command. Likewise, when the BAP is sending data to the ARM, the received sentence is echoed back to the BAP and then parsed. If the command is $SYNC, then nothing is done. For the other data commands, the data is parsed and passed to the ASV control logic for use in determining future actions.
Testing of the intra-ASV communications system was done by creating a test program on the ARM that opened the COM port connecting it to the BAP and sending various commands to the BAP (Figure 22). The test program contained a loop that looped 30 times. Within each loop, a number was computed by taking the loop index variable modulo 5. This number was used in a SELECT..CASE statement to pick one of the five motor commands that the BAP understands. The send_data function was then called to sync the ARM and BAP, transfer the command, and return. Inside the send_data function, each $SYNC command and its echo are output to the console as well as the command sent and its echo. By running the test program and redirecting the output from the console to a text file, data was collected. The test was run five times and between each test run, the ARM was rebooted and the BAP was powered off and back on to ensure that the test environment was as close to identical for each test as possible.

3.2.2: Inter-ASV Communications

The inter-ASV communication system exists to allow the various ASVs in the fleet to communicate with one another and with the land-based station. This is more complex because of
the potential for difficulties such as: radio interference, an ASV being out of range, and multiple ASVs transmitting simultaneously. Each ASV broadcasts a sentence containing the date/time stamp from the GPS, the ASV’s GPS location, speed, heading, and any environmental data it may have to the land-based station and to the other ASVs in the fleet (Figure 23). The land-based station echoes what it received back to the ASV in confirmation of correct reception. Unlike the Intra-ASV case, no $SYNC command is needed when using the XBee radios since they handle the hand shaking and synchronization on their own.

<table>
<thead>
<tr>
<th>Format:</th>
</tr>
</thead>
<tbody>
<tr>
<td>$AVS,2,&lt;3,,4,,5,,6,,7,,8,,9,,10,,&lt;N&gt;;</td>
</tr>
<tr>
<td>Where:</td>
</tr>
<tr>
<td>$     - Start Character</td>
</tr>
<tr>
<td>,    - Delimiter</td>
</tr>
<tr>
<td>;   - Termination Character</td>
</tr>
<tr>
<td>&lt;2&gt; - Source ASV</td>
</tr>
<tr>
<td>&lt;3&gt; - Destination ASV</td>
</tr>
<tr>
<td>&lt;4&gt; - Date from GPS</td>
</tr>
<tr>
<td>&lt;5&gt; - Time from GPS</td>
</tr>
<tr>
<td>&lt;6&gt; - Latitude from GPS</td>
</tr>
<tr>
<td>&lt;7&gt; - Longitude from GPS</td>
</tr>
<tr>
<td>&lt;8&gt; - Speed from GPS</td>
</tr>
<tr>
<td>&lt;9&gt; - Bearing from GPS</td>
</tr>
<tr>
<td>&lt;10&gt;,&lt;N&gt; - Environmental Data</td>
</tr>
</tbody>
</table>

Example: $AVS,01,02,080511,113245,1234.1234,12345.1234,000.0,000.0;

Figure 23 - Inter-ASV Sentence Structure
The inter-ASV communication system utilizes the third serial port (COM 3) on the ARM board. The COM port is configured for 9600 Baud, 8-bit data, no parity, and one stop bit. Like COM 2, this is a true serial port necessitating the use of the Sipex chip to convert the RS-232 voltages to TTL voltages. In the case of the inter-ASV system, the XBee radio is a CMOS (Complementary Metal-Oxide Semiconductor) device which means that it utilizes voltage levels from 0 volts to
3.3 volts. This requires the use of a logic-level convertor to convert the TTL levels coming out of the Sipex chip to the CMOS levels required by the radio and vice versa.

The sentence structure used in the inter-ASV communication system (Figure 19) is similar to the one used for intra-ASV communications in that the sentence starts with a dollar sign ‘$’, followed by a four character message, a comma-delimited parameter list, and is terminated by a semi-colon ‘;’.

One of the major differences found in the sentences used in the inter-ASV system is that they must contain an ID field that identifies which boat in the fleet sent the sentence. This allows all of the ASVs to correctly keep track of the data coming from every other ASV in the fleet. The ASV loads its ID from a file stored on the USB flash drive used for program and data storage. A destination ID field is included in the sentence structure for future use, but is not in use at this time. Date, time and locational information fill fields 4 through 9 and field 10 begins a comma-delimited list of whatever environmental data needs to be sent. The total sentence size may not exceed 255 characters including the start and termination characters.

Two different tests were conducted on the inter-ASV communication system: distance testing and communications reliability testing. The distance testing of the XBee radios was done with two XBee’s connected to two BASIC Stamps. Communication reliability testing was completed once the XBee’s were connected to the ARM microcontrollers.

The distance testing apparatus shown in Figure 24 was constructed using BASIC Stamp Homework Boards in order to make the test apparatus more portable since the 9-volt battery powering the BASIC Stamp Homework Board could also handle the power required by the XBee radios. Two units were constructed that were electrically identical, but ran different software. The software for the mobile ‘echo’ unit constantly polled the XBee radio looking for an
incoming message. Once a message was read, it was immediately echoed back to the originating
station. The software on the ‘base’ unit was more complex. It sent out the series of messages
consisting of the single digits 0-9, compared the echoed messages to ensure that no messages
were dropped, and kept a count of the number of correctly echoed messages in each test run. For
each test run, ten messages were sent. Three repetitions were run at each distance. The distance
between the base and echo unit was determined by a hand-held laser range finder set to read
distances in meters. The echo unit was hand-carried at approximately shoulder height and the
test was run at numerous distances. In addition, several small studies were done varying height
of the echo unit above ground and various distances from the base unit at knee height.

For communications reliability, the test message was constructed using the sentence structure
given in Figure 23 above to provide a more realistic test. The test message was sent from an
XBee attached to an ARM microcontroller to an XBee attached to a BASIC Atom Pro acting as
the ‘echo’ unit in this test. The test message was sent 30 times for each repetition and the test
was repeated five times. The total number of messages sent and the total number echoed correctly were recorded for each test.

3.3: Results

3.3.1: Intra-ASV Results

The intra-ASV system was tested by repeatedly sending commands from the ARM to the BAP and counting the number of $SYNC commands that were exchanged before the actual command could be sent as well as the number of commands correctly sent and received after syncing had occurred. The five test runs yielded 150 data points. The fewest $SYNC commands that were ever sent in a test was one since even if the COM ports are perfectly in sync, the program sends at least one $SYNC to be sure. The largest number observed in the testing was 13 $SYNCs sent for a single message. Of the 150 messages sent and echoed, only 1 message was not echoed correctly.

3.3.2: Inter-ASV Results

The distance testing generated the data shown in Table 3. The numbers in the Rep1, Rep2, and Rep 3 columns are the number of messages sent that were correctly echoed back. Each repetition consisted of ten messages (the digits 0-9) and three repetitions were run.

Table 3 - XBee distance testing shoulder height raw data

<table>
<thead>
<tr>
<th>Distance (m)</th>
<th>Rep1</th>
<th>Rep2</th>
<th>Rep3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>19.9</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>48.9</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>108.4</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>159</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>194</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>236</td>
<td>10</td>
<td>10</td>
<td>8</td>
</tr>
<tr>
<td>270</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>293.7</td>
<td>10</td>
<td>10</td>
<td>8</td>
</tr>
<tr>
<td>318</td>
<td>10</td>
<td>10</td>
<td>6</td>
</tr>
<tr>
<td>341</td>
<td>0</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>364</td>
<td>7</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>388</td>
<td>10</td>
<td>10</td>
<td>8</td>
</tr>
<tr>
<td>410</td>
<td>0</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>434</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>457</td>
<td>6</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>480.9</td>
<td>8</td>
<td>10</td>
<td>7</td>
</tr>
<tr>
<td>504.2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
for each distance recorded. The raw data from Table 3 is graphed in Figure 25. From the graph, the anomalous data observed at the 480-meter distance becomes apparent.

Figure 25 - Inter-ASV Communication system testing - raw data (shoulder height = 1.5 meters)
A separate study was performed with the echo unit at various heights for that distance. The raw results for that study are shown in Figure 26.

For the reliability testing, 150 messages were sent from the ARM attached XBee to the BASIC Atom attached XBee and echoed back. Of 150 messages sent, 150 were received and correctly echoed back to the sending station.
3.4: Data Analysis

3.4.1: Intra-ASV

The intra-ASV communication system showed a maximum number of $SYNC commands on a single message of 13 and only required more than a single $SYNC on 5 out of 150 messages sent. This means that less than 4% of the time messages are sent between the ARM and the BAP the system will need to send more than one $SYNC. The average number of $SYNCs sent in the repetition with the worst single case was 1.43 $SYNCs per message and the overall average number of $SYNCs sent for all 150 messages was 1.17 $SYNCs per message.

Of the 150 messages sent between the two microcontrollers, only 1 was echoed incorrectly. This results in a 99.3% probability that a message sent over the intra-ASV communications system will be transmitted correctly the first time. One way of dealing with the possibility of an incorrect message being received is to double send each message. This reduces the chance of an incorrect message from 0.7% or 0.007 to 0.00049 or ~.05%. Since an incorrect action taken by
the BAP will be detectable by the ARM almost immediately and a correction sent, double sending each message was deemed excessive. In the event that an incorrect message does get transmitted, a number of steps can be taken to ameliorate the possible consequences by placing logical constraints within the software to check for impossible or incorrect parameters before implementing the command.

3.4.2: Inter-ASV

Figure 27 charts the averages at each distance and shows a simple second-degree polynomial fit. Examining the averages of the raw data for the Inter-ASV distance testing, we observe a drop-off in correct messages beginning at a distance of approximately 300 meters. This is more than double the manufacturers listed range of 120 meters for this unit and the unit showed perfect results well past the manufacturers specifications. Past 300 meters the unit’s behavior becomes somewhat erratic. The anomalous valleys and peaks may be explained by slight changes in elevation in the outdoor testing environment.

![Plot of Average Distance Test Data](image)

Figure 27 - Inter-ASV communication system distance testing averages of 1.5 meter height readings with 2nd degree poly fit show 90% reliability to 300 meters

The study done at a distance of 480 meters from the base unit with the echo unit held at various heights shows the importance of height above ground for the XBee radio units. This is
reasonable since, at an operating frequency of 2.4 GHz, signals from these units will not penetrate obstacles well.

With a completion rate of 100% correctly echoed messages, the reliability testing on the inter-ASV communication system demonstrates how well these units can be expected to perform over distances shorter than their maximum range.

### 3.5: Conclusions

The design of the ASVs in the fleet demands two communications systems: an internal one (intra-ASV) and an external one (inter-ASV). The intra-ASV communication system provides a means for the two microcontrollers within the ASV to communicate commands and system data to each other with 99.3% reliability in the tests performed. In rare cases, the COM ports may get out of sync with each other, but the addition of the $SYNC mechanism provides a means of recovering from this problem and resuming reliable communications. The inter-ASV communication system gives the ASV the ability to communicate with its peers in the fleet. The communication provided by using XBee radios has proven to be very reliable in transmitting the necessary data with greater than 90% reliability over distances of 300 meters and 100% reliability over shorter distances. These results far exceed the design expectations. This level of communication is critical for a multi-agent autonomous surface fleet to become something more than just multiple stand-alone ASVs co-located on the same body of water.

Once a practical fleet of ASVs with reliable communications has been constructed, the application of that fleet to real-world problems becomes a possibility. Some of those areas include: gradient-tracking, bird predation reduction, and environmental/biosecurity monitoring. In all of these applications, the ability to record the temporal and spatial locations of the ASVs
and display those on various maps becomes a critical step in both the solution to the application and to the verification of the ASV fleet’s proper operation.
CHAPTER 4: DATA ANALYSIS OF SPATIAL AND ENVIRONMENTAL DATA FROM MULTIPLE AUTONOMOUS SURFACE VEHICLES

4.1: Introduction and Literature Review

Once multiple identical ASVs have been constructed and once they have been made into a fleet with the addition of a method of reliably communicating data to each other, many applications become possible. In every application involving the fleet, knowing where the ASVs are at all times is critical to the fleet being able to cooperatively solve problems. Recording that spatial and temporal data allows the generation of maps.

In a very real sense, every piece of data that an Autonomous Surface Vehicle (ASV) collects is geospatial data from the location data to level of ambient light striking the ASV to the temperature of the water or the presence of a toxic substance in the water. Environmental data without its associated geospatial data is meaningless. We differentiate then between data that is strictly geospatial in nature and environmental data that has a geospatial component by its nature. Analyzing the geospatial data and environmental data that each ASV collects and stores provides a useful tool for evaluating the correct behavior of the ASV in relation to the environment and the other members of the fleet. This analysis may also indicate areas of hardware or software malfunction or possible areas for hardware and software improvements.

4.2: Materials and Methods

Data collected by the ASVs are stored in various files, all of which are stored on a USB flash drive connected to the ARM microcontroller. At the end of each test run, the USB drive is retrieved and the data files copied for backup and analysis. All data files are opened, written to, and closed in as close to an atomic (single uninterrupted operation) manner as possible to reduce the possibility of file corruption.
4.2.1: Environmental Data

The environmental data collected will depend on the particular task assigned to the fleet of ASVs. In the simplest case, the data being collected is a single instantaneously available measurement such as a temperature reading made with a J-Type thermocouple. Each time the ARM reads the thermocouple value it stores that value along with the most current position from the Global Positioning System (GPS) sensor in a data file on the USB drive. This data is used along with information from the GPS to construct a ‘sentence’ in the format of the inter-ASV communication system. This sentence, in addition to being sent to the other ASVs in the fleet is written to a file formatted with one line for each sensor reading taken. A line is added to this file only when a thermocouple reading is taken so each line represents a unique spatial and temporal combination. Since the sentence structure of the inter-ASV communication system allows for multiple environmental parameters to be passed in each sentence, more than one sensor value may be read and stored in each line of data within the data file.

4.2.2: Spatial Data

Spatial data obtained from the GPS is recorded into a file every time the GPS is queried. The result is a log of the boats track accurate to within the time frame between successive GPS readings. The GPS outputs a new position every second in the form of National Marine Electronics Association (NMEA) sentences. Each sentence is an ASCII character string consisting of a start character “$” followed by a sentence identifier composed of alphabetic characters, a comma-delimited list of parameters ending with an asterisk “*” followed by a two hexadecimal checksum and a carriage return/line-feed (CR,LF) combination. Although the GPS can output numerous NMEA sentences, the GPS units on the ASVs have been programmed to only output the $GPRMC sentence which they do once every second. The five character
GPRMC is divided into a two character field (“GP”) that indicates the source of the data (“GP” stands for GPS transmitter), and a three character field indicating the type of message (RMC is the third (“C”) of three Recommended Minimum sentences for GPS locations). The GPRMC sentence contains all of the parameters necessary for the ASV to accurately record its position and thus the position of environmental data being collected. In addition, the data necessary to allow the ASV to compute new navigational headings is also contained in this sentence.

4.3: Results

Testing was done on the control system in a laboratory environment that yielded data in the form of the data files output from a single ASV control system. Field testing resulted in a set of data files consisting of a GPS track log file and an environmental data file for each ASV. The spatial data collected by the ASV during tested resulted in the creation of a GPS log file with an entry for each GPS point read from the GPS by the ARM microcontroller. This data file is then mapped showing the path the ASV took during its operation. A sample GPS log file is shown in Figure 28. Each line in the figure represents a unique GPS point that was recorded.

```
$GPRMC,193224,A,3024.5395,N,09110.6897,W,000.0,356.0,140511,001.3,E*6F
$GPRMC,193226,A,3024.5394,N,09110.6895,W,000.0,356.0,140511,001.3,E*6E
$GPRMC,193227,A,3024.5394,N,09110.6895,W,000.0,356.0,140511,001.3,E*6F
$GPRMC,193229,A,3024.5394,N,09110.6893,W,000.0,356.0,140511,001.3,E*67
$GPRMC,193230,A,3024.5393,N,09110.6892,W,000.0,356.0,140511,001.3,E*69
$GPRMC,193232,A,3024.5392,N,09110.6890,W,000.0,356.0,140511,001.3,E*68
$GPRMC,193236,A,3024.5390,N,09110.6888,W,000.0,356.0,140511,001.3,E*67
$GPRMC,193238,A,3024.5390,N,09110.6886,W,000.0,356.0,140511,001.3,E*69
$GPRMC,193240,A,3024.5390,N,09110.6889,W,000.0,356.0,140511,001.3,E*67
$GPRMC,193241,A,3024.5391,N,09110.6889,W,000.0,356.0,140511,001.3,E*67
$GPRMC,193243,A,3024.5391,N,09110.6890,W,000.0,356.0,140511,001.3,E*6D
```

Figure 28 - Sample GPS log file output
The sentence is broken down as follows (using the first line in Figure 28): ‘$’ indicates the start of the sentence, GPRMC indicates that the data is one of the recommended minimum GPS
location sentences from a GPS receiver, 193224 is the UTC time the position was recorded in HHMMSS format, the ‘A’ indicates that the receiver was working correctly, “3024.5395,N” is the latitude in degrees and decimal minutes, “09110.6897,W” is the longitude in degrees and decimal minutes, 000.0 is the speed over ground in knots, 356.0 is the true course, 140511 is the date in DDMMYY format, “001.3,E” is the magnetic variation, and “*6F” is the checksum for the string. Figure 29 shows a sample map with the GPS track log points displayed and a line joining them overlaid on an image of the lake the data was collected on.

Figure 29 - Sample map showing GPS track log points with a simple poly line joining them in time sequence
4.4: Data Analysis

Data analysis was performed by mapping both sets of data created by each ASV and comparing them to each other to ensure that locations for the environmental data were being stored correctly. Maps were also generated with the environmental data from all of the ASVs in the fleet displayed. As an example of statistical analysis that can be performed, data from Price and Hall (2011) (Figure 30) shows a reduction of 50 – 70% in predatory bird activity when autonomous vehicles were active on ponds.

![Graph: Number of New Birds Landing on Pond Surface versus Amount of Boat Movement]

**Figure 30 - Statistical analysis of data on bird predation reduction on catfish ponds after Price and Hall 2011**

4.5: Conclusions

Environmental data is by its nature locational data. A data point that contains information about the environment but doesn’t have the associated location where the data originated is vague at best and more likely useless. It is imperative that as the ASVs collect environmental data, they store the location that the sample was taken with the data collected. The use of simple text files
written to a USB flash drive as the data is collected by the ASV provides the necessary storage so that the data collected can be analyzed once the ASV has been retrieved. Since it is useful to compare the environmental data collected with the overall path the ASV takes while completing its task, a GPS track log is also kept as a separate text file on the same USB flash drive.

The use of mapping to display the data collected by the ASV fleet provides a method for evaluating the success of the fleet at correctly completing a particular task. The use of maps also makes the data and the conclusions derived from that data more easily accessible and provide a way of showing the relationships between the data points.

The fleet of autonomous surface vehicles constructed to date incorporates both GPS receivers and radio communications. These two features, along with the more advanced control system enable the ASVs to cooperate in solving tasks. The data collected along the way fall into two categories: GPS track log data, and environmental data. Both of these enable us to map those data streams to visualize the operation of the ASV fleet and the data being collected. With the addition of new sensors, more environmental applications can be envisioned. Some of these might include: bathymetry readings in coastal estuaries, new ways of reducing bird predation on aquaculture ponds, and potential biosecurity applications.
 CHAPTER 5: CONCLUSION

Autonomous surface vehicles have proven to be useful tools in aquacultural applications and in limited environmental applications, but always as single units performing a task (Hall, et al, 2001; Price and Hall, 2002; Hall, et al. 2006, Price and Hall, 2011). The creation of a fleet of autonomous surface vehicles represents the advent of a unique tool for solving significant problems in a number of water-based fields. Multiple cooperating vehicles have applications in environmental monitoring, ecosystem monitoring, and aquaculture. The ability of the ASVs in the fleet to communicate the position and environmental data they collect to one another enables group problem solving.

Construction of a fleet of three Autonomous Surface Vehicles was completed and two more ASVs are currently under construction to bring the total number of vehicles in the fleet to five. The vehicles are identical insofar as is possible within the tolerances of the individual components comprising the systems. Modularity as a primary design goal has been implemented in every area of the ASV design from the design of the physical vehicle to the design of the electronic control system and the software that runs on the control system. A dual-microcontroller control system was designed and constructed using an ARM-based microcontroller board, the TS-7260 from Technologic Systems (www.embeddedarm.com), and a BASIC Atom Pro 24M from Basic Micro (www.basicmicro.com). Tasks within the ASV were partitioned between the two microcontrollers with the ARM taking the governing role and the BASIC Atom Pro taking a subservient role.

Communications are a necessary part of this design both within a single ASV and across the fleet. The two microcontrollers within an ASV utilize a direct serial port – serial port connection
to transfer data and commands back and forth. Communication between ASVs within the fleet is a task given to the ARM microcontroller on each ASV and is handled through a serial port attached to an XBee Pro Series 2 radio from Digi International (www.digi.com). These radios form a mesh network that handles routing messages automatically from the source to the destination. Autonomous Surface Vehicles within the fleet exchange data regarding their current location and any environmental measurements they have made. This exchange allows the ASVs to work together in solving problems such as point source pollution tracking.

The data collected by each ASV may be analyzed during a post-processing phase to demonstrate the effectiveness of the control systems and the usefulness of inter-ASV communications. The data collected is also used to evaluate possible improvements to the ASV system in terms of both hardware and software upgrades. The maps produced from the data provide a visual method of evaluating the ASVs effectiveness and help convey the impact of the environmental data collected.

The design, construction, and testing of the fleet of Autonomous Surface Vehicles for Environmental Applications is only the first step in the life of a unique tool for environmental monitoring, aquaculture system monitoring and other as yet undiscovered areas of application.

An example of simple environmental monitoring could be simple temperature measurements made using a thermocouple. This data would then be mapped separately for each ASV as well as jointly for the whole fleet. Using a model in ArcGIS® (www.esri.com), the temperature data will be mapped and the pond shaded to reflect the temperature gradients detected by the ASV fleet. Figure 31 shows a sample map produced using an ArcGIS model showing pond coverage by a single boat being used for bird predation reduction.
Future work will include: the application of artificial intelligence programming on the ASVs to enhance their problem solving capability, enhanced sensors for environmental monitoring for biosecurity applications, ecosystem monitoring over wide areas for coastal restoration and protection, on-board machine vision for multiple applications including bird predation reduction, and non-point source pollution monitoring.

Figure 31 - Sample map showing ASV track buffered using an ArcGIS model


Jeon, Hong Y., Lei F. Tian, Direct application end effector for a precise weed control robot, Biosystems Engineering (2009), doi:10.1016/j.biosystemseng.2009.09.005


Parallaxinc.com website, 2009.


Sparkfun.com website, 2009.


APPENDIX B: ARM SOFTWARE SOURCE CODE

ARM Software source code listing for ASV control system.

```c
#include <sys/types.h>
#include <sys/uio.h>
#include <unistd.h>
#include <fcntl.h>
#include <stdio.h>
#include <sys/select.h>
#include <sys/time.h>
#include <string.h>
#include <termios.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include <stdlib.h>
#include <errno.h>

/**************************************************************************/
/**************************************************************************/
/*************************** Serial Port Functions **************************/
/**************************************************************************/
/**************************************************************************/

/* Function setup_Com1: */
/* */
/* */
/* This function sets up COM1 to serve as the communications port */
/* for ARM - GPS communications. */
/* */
/* Takes as input the following: */
/* */
/* ComPort:  a pointer to a serial port file descriptor */
/* oldtio :  a pointer to a termios structure to hold the */
/* existing COM1 settings so that we can restore them */
/* */
/* Output from this function is the following: */
/* oldtio   : Old com port settings */
/* */
/* Calls the following external functions: */
/* tcgetattr: Gets the current settings of the Com port */
/* bzero   : Zeros out enough space for the object */
/* tcflush : Flushes the selected buffers on Com port */
/* tcsetattr: Writes the new settings to the Com port */
/* */

int setup_Com1(int Com1, struct termios oldtio) {
```
struct termios newtio;  // Create two
termios structures
    int return_val = 1;  // Variable to
hold return_val set to 1 initially

    // Assume we succeed in the test
return_val = 0;

    // Get the current com port settings and store them so we can restore them later.
tcgetattr(Com1,&oldtio);
    // Clear enough memory for a new termios structure.
bzero(&newtio, sizeof(newtio));

    // set up the UART settings in newtio
newtio.c_cflag = B4800 | CS8 | CLOCAL | CREAD;

    // Flush input buffer of UART and write new settings immediately.
tcflush(Com1,TCIFLUSH);
tcsetattr(Com1,TCSANOW,&newtio);

return 0;
}

/***********************************************************************/

int reset_Com1(int Com1, struct termios oldtio) {

    printf("Inside reset function\n");
    // Restore previous com port settings before exiting.
tcsetattr(Com1,TCSANOW,&oldtio);

return 0;
}
int setup_Com2(int Com2, struct termios oldtio) {

    struct termios newtio; // Create two termios structures

    int return_val = 1; // Variable to hold return_val set to 1 initially

    // Assume we succeed in the test
    return_val = 0;

    // Get the current com port settings and store them so we can restore them later.
    tcgetattr(Com2,&oldtio);
    // Clear enough memory for a new termios structure.
    bzero(&newtio, sizeof(newtio));

    // set up the UART settings in newtio
    newtio.c_cflag = B300 | CS8 | CLOCAL | CREAD;

    // Flush input buffer of UART and write new settings immediately.
    tcflush(Com2, TCIFLUSH);
    tcsetattr(Com2,TCSANOW,&newtio);

    return 0;
}

/***************************************************************************/
/***************************************************************************/
int setup_Com2(int Com2, struct termios oldtio) {

    struct termios newtio; // Create two termios structures

    int return_val = 1; // Variable to hold return_val set to 1 initially

    // Assume we succeed in the test
    return_val = 0;

    // Get the current com port settings and store them so we can restore them later.
    tcgetattr(Com2,&oldtio);
    // Clear enough memory for a new termios structure.
    bzero(&newtio, sizeof(newtio));

    // set up the UART settings in newtio
    newtio.c_cflag = B300 | CS8 | CLOCAL | CREAD;

    // Flush input buffer of UART and write new settings immediately.
    tcflush(Com2, TCIFLUSH);
    tcsetattr(Com2,TCSANOW,&newtio);

    return 0;
}

/*************************************************************************/
int reset_Com2(int Com2, struct termios oldtio) {
    printf("Inside reset function\n");
    // Restore previous com port settings before exiting.
    tcsetattr(Com2, TCSANOW, &oldtio);
    return 0;
}

int send_data(int ComPort, int delay, int BAPCom, char *message, int messlen) {
    // This function sends a message out a serial port and checks
    // to make sure the message is correctly echoed back from the receiver.
    //
    // Takes as input the following:
    // ComPort: a pointer to a serial port file descriptor
    // delay: an integer indicating how long to delay
    // BAPCom: an integer flag indicating message is for Atom
    // message: a character array (max 80 chars) containing message
    // messlen: an integer containing the length of the message
    // returned in message.
    //
    // Output from this function is the following:
    // return_val: an integer indicating success or failure
    //
    // Calls the following external functions:
    // strcpy: Copies the contents of one string into another
    // tcflush: Flushes the specified buffer on the Com Port
    // write: Writes data out the com port
    // printf: Prints messages to the console (for debugging)

    // Your implementation here
}
int setup_Com3(int Com3, struct termios oldtio) {

    struct termios newtio; // Create two termios structures
    int return_val = 1; // Variable to hold return_val set to 1 initially

    // Assume we succeed in the test
    return_val = 0;

    // Get the current com port settings and store them so we can restore them later.
    tcgetattr(Com3,&oldtio);
    // Clear enough memory for a new termios structure.
    bzero(&newtio, sizeof(newtio));

    // set up the UART settings in newtio
    newtio.c_cflag = B9600 | CS8 | CLOCAL | CREAD;

    // Flush input buffer of UART and write new settings immediately.
    tcflush(Com3, TCIFLUSH);
    tcsetattr(Com3,TCSANOW,&newtio);

    return 0;
}

int reset_Com3(int Com3, struct termios oldtio) {

    // Takes as input the following:
    // ComPort:  a pointer to a serial port file descriptor
    // oldtio: a pointer to a termios structure holding the existing COM2 settings so that we can restore them

    // Calls the following external functions:
    // tcsetattr: Writes the new settings to the Com port
    // printf: Writes debug messages to the console

printf("Inside reset function\n");
// Restore previous com port settings before exiting.
tcsetattr(Com3,TCSANOW,&oldtio);

return 0;
}
/**/
int syncBAP(int ComPort, int delay)
{
    /* Local Variables */
    int return_val = 1;          // Variable to hold return_val set to 1 initially
    char sinput[80];            // Array to hold test string
    char resp[80];              // Array to hold response from BAP
    int resplen;                // Length of response string sent by BAP
    int synced;                 // Flag to help sync up with BAP

    // Assume we're not synced
    synced = 0;
    // Copy Sync string into sinput for sending
    strcpy(sinput, "$SYNC,000;" );

    while (!synced)
    {
        // Flush the I/O buffers
        tcflush(ComPort, TCIOFLUSH);

        // Send sync string to BAP
        write(ComPort,sinput,strlen(sinput));
        printf("Sent: %s\n",sinput);

        // Wait for a bit
        usleep(delay);

        // Blank string that holds the response received.
        strcpy(resp," ");
        resplen = read(ComPort,resp,10);
        printf("Recd: %s\n",resp);

        // Wait for a bit
        usleep(delay);

        // Check to see if the BASIC Atom Pro correctly echoed the command string
        if (!strcmp(resp,sinput))
        {
            // Yes, we're synced. Exit the while loop
        }
    }
}
synced = 1;
}
}
return 0;
}

int send_data(int ComPort, int delay, int BAPCom, char message[], int messlen){

    // Local Variables
    int return_val = 1; // Variable to hold return_val set to 1 initially
    char sinput[80]; // Array to hold test string
    char resp[80]; // Array to hold response from BAP
    int resplen;
    // Length of response string sent by BAP
    int synced; // Flag to help sync up with BAP

    // Let console know where execution is
    printf("Inside send_data\n");
// Assume we succeed in the test
return_val = 0;

// Are we communicating with the BAP (BASIC Atom Pro)?
// If so, we need to sync up the com ports before starting
if (BAPCom)
{
    syncBAP(ComPort,delay);
}

// NOW SEND THE REAL MESSAGE

// Flush Input and Output buffers on the serial port
tcflush(ComPort, TCIOFLUSH);

// Copy message into sinput
strcpy(sinput, message);

// Send message out
write(ComPort,sinput, strlen(sinput));
printf("Sent: \%s\n",sinput);

// Wait for .1 seconds
usleep(delay);

// Blank string that holds the response received.
strcpy(resp,"                                        ");
// Read response
resplen = read(ComPort,resp,strlen(sinput));
// Echo response
printf("Recd: \%s\n\n",resp);

// Wait for .1 seconds
usleep(delay);

// Check for correct echo. If message echoed incorrectly, set return_val=-1
if (!strcmp(sinput, resp))
{
    return_val = -1;
}

// Return to main program.
return return_val;
}
int read_gps(int ComPort, int delay, char message[]) {
    // Local Variables
    char resp[80];
    int resplen;
    int iloop;
    unsigned char tempchr;
    int synced;
    // Flag to help sync up with BAP

    // Let console know where execution is
    printf("Inside read_gps\n");
    // Flush Input and Output buffers on the serial port
    tcflush(ComPort, TCIOFLUSH);

    // Clear response string
    for (iloop=0; iloop<80; iloop++)
        resp[iloop] = ' ';

    // Assume we succeed in the test
    // return_val = 0;
    // If so, we need to sync up the com ports before starting
    // Assume we're not synced
    // syncBAP(ComPort,delay);

    // Wait a bit
    usleep(delay);
    resplen = read(ComPort,resp,74);
    printf("GPS: %sn",resp);
    strcpy(message, resp);
    return 0;
}

int parseGPS(char message[], int messlen, char GPSdate[], char GPStime[],
    double *lat, double *lon, double *bearing, double *speed) {
    // Local Variables
    int return_val=1;

    if (strncpy(message,"GPRMC",5) == 0) {
        return_val = 0;
        printf("Got the right string\n");
    } else {
        return_val = 1;
    }
printf("Not the right string\n");
}

return return_val;
}

// *****************************************
// Function main:     */
// Takes as input the following:
//  ComPort: a pointer to a serial port file descriptor */
//  delay : an integer indicating how long to delay */
//  BAPCom : an integer flag indicating message is for Atom */
//  message: a character array (max 80 chars) containing message*/
//  messlen: an integer containing the length of the message */
//  contained in message. */

// Output from this function is the following:
//  return_val: an integer indicating success or failure */

// Calls the following external functions:
//  strcpy : Copies the contents of one string into another */
//  tcflush : Flashes the specified buffer on the Com Port */
//  write : Writes data out the com port */
//  printf : Prints messages to the console (for debugging) */
//  usleep : Pauses execution for a number of microseconds */
//  read : Reads data in from the com port */

int main(int argc, char *argv[]) {
  int Com2, Com3, Com1; // Pointer to file descriptor
  int BAPCom;        // Flag to indicate intraboat communication to BAP
  int iloop;            // Length of message stored in message array
  int messlen;        // Generic looping variable
  int tstcnt;           // Counts number of times we've tried to
  send to BAP

int success; // Flag for success of function call
char message[80]; // Holds message to send to either the XBee or the BAP
double lat=0.0; // Latitude, Longitude, Bearing and Speed from GPS
double lon=0.0;
double bearing=0.0;
double speed=0.0;
char GPSdate[6]; // GPS Date and Time
char GPStime[6];
struct termios Com2_old, Com3_old, Com1_old;

printf("Inside Main\n");

// Attempt to open com1 and point Com1 to it.
// Com1 = open("/dev/ttyAM0", O_RDWR | O_NOCTTY | O_NDELAY);
// See if we succeeded, if not, print an error message and quit.
if (Com1 < 0) {
    perror("TSUART:open:/dev/ttyAM0");
    return 1;
}
else {
    printf("Succeeded in opening Com1\n");
}

// We succeeded if we got here. Setup Com1 like we need it.
if (setup_Com1(Com1, Com1_old)) {
    printf("Setting up uart for Com1 failed!\n");
    return 1;
}
else {
    printf("Successfully setup Com1\n");
}

// Attempt to open com2 and point Com2 to it.
Com2 = open("/dev/ttyAM1", O_RDWR | O_NOCTTY | O_NDELAY);
// See if we succeeded, if not, print an error message and quit.
if (Com2 < 0) {
    perror("TSUART:open:/dev/ttyAM1");
    return 1;
}
else {
    printf("Succeeded in opening Com2\n");
}

// We succeeded if we got here. Setup Com2 like we need it.
if (setup_Com2(Com2, Com2_old))
{  printf("Setting up uart for Com2 failed!\n");
  return 1;
}
else
  printf("Successfully setup Com2\n");

/**********************

// Attempt to open Com3 and point Com3 to it.
Com3 = open("/dev/ttyTS0", O_RDWR | O_NOCTTY | O_NDELAY);

// See if we succeeded, if not, print an error message and quit.
if (Com3 < 0) {
  perror("TSUART:open:/dev/ttyTS0");
  return 1;
}
else
{
  printf("Succeeded in opening Com3\n");

  // We succeeded if we got here. Setup Com2 like we need it.
  if (setup_Com3(Com3, Com3_old))
     {  printf("Setting up uart for Com3 failed!\n");
        return 1;
     }
  else
     {  printf("Successfully setup Com3\n");
            

            /*********************/
            
            for (iloop=0; iloop<5; iloop++)
            {
                  // Clear response string
                  for (tstcnt=0; tstcnt<80; tstcnt++)
                     message[tstcnt] = ' ';

                  // Test Com1 - GPS

                  // read_gps(Com1,1500000,message,messlen);
                  // printf("\n");

                  // Test Com2 - Intraboat Communication
                  tstcnt = 0;  // Start counter at 0
                  success = -1;  // Assume failure
                  BAPCom = 1;
                  strcpy(message,"$QGPS,000; ");
                  messlen = strlen(message);
                  printf("Message = %sn",message);
                  printf("Length = %dn",messlen);
                  while ((tstcnt < 5) && (success < 0))
                    {
success = send_data(Com2,1000000, BAPCom, message, messlen);
tstcnt++;
}
if (tstcnt == 5)
    printf("Failed to send message!\n");
else
{
    read_gps(Com2,1200000,message);
    messlen = strlen(message);
    printf("GPS: %s\n",message);
    printf("GPSlen = %d\n",messlen);
}
if (parseGPS(message,messlen,GPSdate,GPStime,&lat,&lon,&bearing,&speed) == 0)
{
// writedata("GPSrawdata.txt",message,messlen);
// buildASV(message,messlen,GPSDate,GPSTime,lat,lon,bearing,speed);
// writedata("ASVdata.txt",message,messlen);

// Test Com3 - Interboat Communication
BAPCom = 0;
printf("Message = %s\n",message);
printf("Length = %d\n",messlen);
send_data(Com3,1000000, BAPCom, message, messlen);
printf("\n\n");
}
printf("Back from test_data\n");
if (reset_Com2(Com2, Com2_old))
{
    printf("OOPS! Can't reset terminal settings for Com2\n");
    return 1;
}

// Close file descriptor that points to the serial port
close(Com2);

if (reset_Com3(Com3, Com3_old))
{
    printf("OOPS! Can't reset terminal settings for Com3\n");
    return 1;
}

// Close file descriptor that points to the serial port
close(Com3);

return 0;
BASIC Atom Pro Software source code listing:

SETHSERIAL1 H300, H8DATABITS, HNOPARITY, H1STOPBITS

Purpose: This program runs on the BASIC Stamp microcontroller as part of the overall control system of the autonomous vehicles. This algorithm performs the following functions:
1. Provides all motor control for the AVS.
2. Monitors ambient lighting to detect nighttime.
3. Receives motor commands from the ARM.
4. Reports ambient lighting conditions to the ARM.

--- Pin Definitions ----------------
TM1 CON P10 ' TM1 - From MMBe 1 TO Stamp (Right Paddlewheel)
FM1 CON P11 ' FM1 - From MMBe 1 from Stamp (Right Paddlewheel)
TM2 CON P12 ' TM2 - From MMBe 2 TO Stamp (Left Paddlewheel)
FM2 CON P13 ' FM2 - From MMBe 2 from Stamp (Left Paddlewheel)

' Timing constant for turning
TurnTime CON 15 'Time in ms FOR Boat TO Turn 5 degrees
ARMBaud CON H9600 'Baud Rate constant for serial communications with ARM
ConBaud CON I9600 'Baud Rate constant for serial output to the console
GPSBaud CON I4800 'Baud Rate constant for serial input from GPS

' Generic Variables
i VAR Byte ' generic control variable for loops, etc.
j VAR Byte ' generic control variable for loops, etc.
' Motor Controller Variables
status       VAR      Byte                  ' status byte for motor controllers

' Serial Data Variables

cmdStr       VAR      Byte(10)               ' 10 character byte array
for receiving string data

cmdStrLen    VAR      Word                   ' Holds length of command string received

ackStr       VAR      Byte(5)                ' Holds acknowledgement string from ARM

' Command String Variables

argVal       VAR      Word                   ' Holds decimal equivalent of three character number portion of cmdStr

GPSData      VAR      Byte(73)               ' Holds GPS Data

synced       VAR      Byte                   ' Holds flag to indicate serial sync achieved with ARM

'************************************************
'********          Main Program          ********
'************************************************

Main:
' Initializations

CLEAR
' Clear all serial buffers

GOSUB MotorInit
' Initialize the motors to the stopped position

SEROUT ConPin, ConBaud, ["Inside Main Program", 13]
' Begin main program loop

WHILE (1)                      ' Loop forever

GOSUB RecData
' See if the ARM is sending a command

' Check for valid command. If cmdStrLen = 0, then no command or invalid command

IF (cmdStrLen > 0) THEN

GOSUB parseCmd
' Parse and execute the command

ENDIF

' GOSUB GPSSend
' Send GPS Data to ARM

' Perform Boat System checks

'GOSUB ambCheck
' Check ambient light conditions

'GOSUB voltCheck
' Check battery system voltages

WEND
' End of main program While loop

END

'*******************************************************************************
RecData:

    ' Initialize the cmdStrLen to 0 since we have no command yet
    cmdStrLen = 0
    ' Clear out the cmdStr array so no false data comes through
    FOR j=0 to 9 STEP 1
        cmdStr(j) = 32
    NEXT
    ' Clear acknowledgement string
    FOR j=0 to 4 STEP 1
        ackStr(j) = 32
    NEXT

    ' Read from the serial port waiting for a $ which indicates the beginning of a command string.
    HSERIN 1, NoData, 1000000, [WAIT("$"),STR cmdStr\8;";"
    SEROUT ConPin, ConBaud, ["Got: ",STR cmdStr,13]     ' Echo the received string to the terminal.
    ' Echo received command back to ARM for verification
    HSEROUT 1, ["$",STR cmdStr\8,";",10]
    ' Read the serial port again looking for an ack/nak signal
    HSERIN 1, [WAIT("$"),STR ackStr\3;";"
    ' SEROUT ConPin, ConBaud, ["Good Exchange? : ",STR ackStr\3,13]     ' Echo the ack/nak string received.

    ' Check for Correct reception
    IF (ackStr(0) = "A") THEN
        cmdStrLen = 8
        ' ACK received
        ' Good command, set the length to 8
        ELSEIF (ackStr(0) = "N")
        ' NAK received
        ' Got a command, but didn't receive it right.  Try again.
        ' ENDIF
    ' Got garbage.  Quit and return to main

NoData:

    ' Received nothing from the ARM, bail out and go back to main.
    SEROUT ConPin, ConBaud, ["Leaving RecData: Command = ",STR cmdStr\10," and length = ",DEC3 cmdStrLen,13]

RETURN     ' Return from call to RecData
parseCMD:

' We know we have a good command. Pull out the numeric value
' of the argument to pass to the subroutine.
argVal = ((cmdStr(5)-48)*100) + ((cmdStr(6)-48)*10) + (cmdStr(7)-48)

' Parse using first letter of command. (Can be expanded if multiple commands share a common
first letter.)
'(FOR TESTING, commented out subroutine calls and just echoed what it would do)
IF (cmdStr(0) = "R") THEN
    SEROUT ConPin, ConBaud, ["Turning Right ",DEC3 argVal," degrees",13]
    GOSUB TurnRight
ELSEIF (cmdStr(0) = "L")
    SEROUT ConPin, ConBaud, ["Turning Left ",DEC3 argVal," degrees",13]
    GOSUB TurnLeft
ELSEIF (cmdStr(0) = "F")
    SEROUT ConPin, ConBaud, ["Going forward at ",DEC3 argVal," percent speed",13]
    GOSUB Forward
ELSEIF (cmdStr(0) = "B")
    SEROUT ConPin, ConBaud, ["Going backward at ",DEC3 argVal," percent speed",13]
    GOSUB Backward
ELSEIF (cmdStr(0) = "Q")
    SEROUT ConPin, ConBaud, ["Queried GPS",13]
    'GOSUB GPSsend
ELSEIF (cmdStr(0) = "S") AND (cmdStr(1) = "T")
    SEROUT ConPin, ConBaud, ["All STOP - ",DEC3 argVal," should be all 000",13]
    GOSUB StopMtr
        ' We know it's valid and it starts with 'ST'. Must be Stop command.
ELSEIF (cmdStr(0) = "S") AND (cmdStr(1) = "Y")
    SEROUT ConPin, ConBaud, ["SYNC - ",DEC3 argVal," should be all 000",13]
ENDIF

RETURN

' Coming soon
Paddlewheel Drive Subroutines

Initialization for paddlewheel drivers

MotorInit:
  SEROUT ConPin, ConBaud, [ "Initializing motor controllers", 13]
  'Get out of SPDCON mode
  LOW FM1
  LOW FM2
  PAUSE 300
  HIGH FM1
  HIGH FM2
  PAUSE 25
  PAUSE 1000

  'Stop the motors
  GOSUB StopMtr

  'Make sure the motors are set for forward motion
  '  GOSUB InitDir

  RETURN ' End of MotorInit:

'--------------------------------------------------------

Move the boat forward

Forward:
  SEROUT ConPin, ConBaud, [ "Going Forward", 13]
  'Make sure both motors are set for forward
  GOSUB InitDir

  'Smoothly ramp up to 100%
  FOR i = 55 TO 255 STEP 20
    SEROUT FM1,I2400,[$55,$03,i]
    SEROUT FM2,I2400,[$55,$03,i]
    PAUSE 100
  NEXT

  RETURN ' End of Forward:

'--------------------------------------------------------

Move the boat backward

Backward:
  SEROUT ConPin, ConBaud, [ "Going backwards", 13]
  'Make sure both motors are set for forward
  GOSUB InitDir

  'Switch the Motor Controllers' directions to reverse
  SEROUT FM1,I2400,[$55,$81]
  SEROUT FM2,I2400,[$55,$81]
' Smoothly ramp up to 100%
FOR i = 55 TO 255 STEP 20
   SEROUT FM1,I2400,[$55,$03,i]
   SEROUT FM2,I2400,[$55,$03,i]
   PAUSE 100
NEXT
RETURN ' End of Backward:

' Stop the boat
StopMtr:

' Ramp motors down to a stop
FOR i = 255 TO 0 STEP 20
   SEROUT FM1,I2400,[$55,$03,i]
   SEROUT FM2,I2400,[$55,$03,i]
   PAUSE 100
NEXT
' Send Stop command
SEROUT FM1,I2400,[$55,$80]
SEROUT FM2,I2400,[$55,$80]
SEROUT ConPin, ConBaud, [ "STOP",13]
RETURN ' End of StopMtr:

' Right turn
TurnRight:
SEROUT ConPin, ConBaud, [ "Turning Right", 13]
' Initialize the motors to forward
GOSUB InitDir
' Reverse the right motor direction
SEROUT FM1,I2400,[$55,$81]

' Smoothly ramp up both to 100%
FOR i = 55 TO 255 STEP 20
   SEROUT FM1,I2400,[$55,$03,i]
   SEROUT FM2,I2400,[$55,$03,i]
   PAUSE 100
NEXT
RETURN ' End of TurnRight:

' left turn
TurnLeft:
SEROUT ConPin, ConBaud, [ "Turning Left", 13]
' Initialize the motors to forward
GOSUB InitDir
' Reverse the left motor direction
SEROUT FM2,I2400,$[$55,$81]

' Smoothly ramp up both to 100%
FOR i = 55 TO 255 STEP 20
  SEROUT FM1,I2400,$[$55,$03,i]
  SEROUT FM2,I2400,$[$55,$03,i]
  PAUSE 100
NEXT

RETURN ' End of TurnLeft:
'-------------------------------------------------------

' Make sure the motor controllers' are going forward
' NOTE: This routine stops the motors before trying to change
'       the direction setting to prevent damage.
InitDir:
SEROUT ConPin, ConBaud, [ "Setting both motor controllers to forward", 13]
' First stop both motors
GOSUB StopMtr

' Controller 1
REPEAT
' Send command to GPS to get Controller 1's status
SEROUT FM1, I2400, $[$55,$85]
' Read Status
SERIN TM1, I2400, 100, InitDir, [Status]

SEROUT ConPin, ConBaud, [ "Controller 1 status bit 0 = ", DEC1 Status, 13]

' Make sure motor is set in forward mode
IF (Status.BIT0 = 1) THEN
  ' Motor is set to reverse direction
  SEROUT FM1,I2400,$[$55,$81]        ' Reverse direction to forward
ENDIF
UNTIL (Status.BIT0 = 0)

' Controller 2
REPEAT
' Send command to GPS to get Controller 1's status
SEROUT FM2, I2400, $[$55,$85]
' Read Status
SERIN TM2, I2400, 100, InitDir, [Status]

SEROUT ConPin, ConBaud, [ "Controller 2 status bit 0 = ", DEC1 Status, 13]

' Make sure motor is set in forward mode
IF (Status.BIT0 = 1) THEN
  ' Reverse direction
  SEROUT FM2,I2400,$[$55,$81]
ENDIF
UNTIL (Status.BIT0 = 0)
RETURN 'End of CheckDir:
'--------------------------------------------------

'****************************************************************************************
VITA

Daniel Davis Smith was born on June 2, 1967, in West Monroe, Louisiana. He graduated from McKinley Senior High School in Baton Rouge, Louisiana, in 1985. In the fall of 1985, he began attending Louisiana State University and pursued an undergraduate degree in Computer engineering with a minor in computer science that he completed in August of 1992. He immediately began a Master of Science program in The Department of Electrical and Computer Engineering that was terminated by illness. In 2005, Daniel returned to Louisiana State University for a master’s in the Department of Biological and Agricultural Engineering. He is currently working on the “Design, Development, and Testing of a Fleet of Autonomous Vehicles for Environmental Applications” under the guidance of Dr. Steven G. Hall, Associate Professor, Department of Biological and Agricultural Engineering, Louisiana State University.