Santa Clara University

Departments of Electrical, Computer and Mechanical Engineering
Date: 7 June 2005

I hereby recommend that the thesis prepared under my supervision by
Robyn Kobashigawa, Paul Mahacek, Nader Mehdizadeh,
Aaron Schooley, Dan Wilson, Nazli Zooleh-Yaghoob-Shahi

Entitled
The WASP: an Autonomous Surface Vessel for the University of Alaska

Be accepted in partial fulfillment of the requirements for the degree of

Bachelor of Science in Electrical Engineering	Bachelor of Science in Mechanical Engineering	Bachelor of Science in Computer Engineering
_____ Thesis Advisor	_____ Thesis Advisor	_____ Thesis Advisor
_____ Electrical Engineering Dept. Chair	_____ Mechanical Engineering Dept. Chair	_____ Computer Engineering Dept. Chair

The WASP: an AUTONOMOUS SURFACE VESSEL
for the University of Alaska

By

Robyn Kobashigawa (ME), Paul Mahacek (ME), Nader Mehdizadeh (COEN),
Aaron Schooley (EE), Daniel Wilson (EE), and Nazli Zooleh-Yaghoob-Shahi (COEN)

Senior Thesis

Submitted in partial fulfillment of the requirements for the degree of
Bachelor of Science in Computer Engineering,
Bachelor of Science in Electrical Engineering
Bachelor of Science in Mechanical Engineering
Santa Clara University
School of Engineering

Santa Clara, California
June 7, 2005

The WASP: an Autonomous Surface Vessel for the University of Alaska

Robyn Kobashigawa, Paul Mahacek, Nader Mehdizadeh, Aaron Schooley,
Daniel Wilson, Nazli Zooleh-Yaghoob-Shahi

Department of Computer, Electrical, and Mechanical Engineering
Santa Clara University
2005

ABSTRACT

The vision of the Autonomous Surface Vessel program is to develop an instrumented boat capable of robust, automatic sampling of oceanographic data at specific locations and depths in estuary and coastal waters. The objective of this year's WASP project was to develop an initial prototype of such a system including software, physical hardware and electronic components. The physical structure includes pontoons, vertical supports, and a platform housing the vessel's power source, onboard electronics and payload. The onboard electronics control the movement and functionality of the vessel and the payload of scientific instruments are attached to a winch for taking readings at depth. Also, the user interface developed for the wireless communication and control of the vessel is internet based allowing the user to control the vessel anywhere in the world. This prototype will be integrated and tested in Elkhorn Slough during the Summer of 2005 as part of an SCU-MBARI project. The work performed to date has been critical in the development of the ASV concept which is expected to lead to an operational system deployed in the Kasitsna Bay Laboratory in Alaska and other national estuaries.

Keywords: Autonomous Surface Vessel, Water Analyzing Surface Platform, Winch, SWATH, Pontoon, Thruster, Vertical Struts, Microprocessor, Graphical User Interface.

ACKNOWLEDGEMENTS

The ASV team would like to thank our families for their unconditional love and support throughout the years. We wouldn't be where we are today without you.

We would also like to thank NOAA West Coast and Polar Regions Undersea Research Center and Geoff Wheat, William Kirkwood and Cheri Everlove from Monterey Bay Aquarium Research Institute, the Robotics Systems Laboratory, the Mechanical Engineering Machine Shop and personnel at Santa Clara University especially Don MacCubbin, the Santa Clara University Robotic Systems Laboratory—Professor Christopher Kitts and Mike Rasay, the National Science Foundation, Ingomar Packing Co., Oki Semiconductors—Mike Weseloh, Pascal Stang and Ted Theocheung.

Development of the WASP vehicle has been supported in part by NOAA through a University of Alaska Fairbanks subcontract through Grant No. NA96RU02211 (UAF Purchase Order No. FP000694). Development has also been supported by the National Science Foundation under Grant No. EIA0079815 and EIA0082041; any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation. Development has also been supported by the Santa Clara University Robotic Systems Laboratory.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1.0 INTRODUCTION

1.1 Background

The ocean provides millions of self-sustaining habitats in reefs, tide pools, deep sea and coastal areas. The numerous species of fish, plants and other sea creatures makes marine biology a fascinating field. The increasing amount of pollution in the oceans is also a rising concern that prompts testing and investigation. For marine protected areas (MPAs) the increase of pollution hurts the environment and the creatures that inhabit the area.

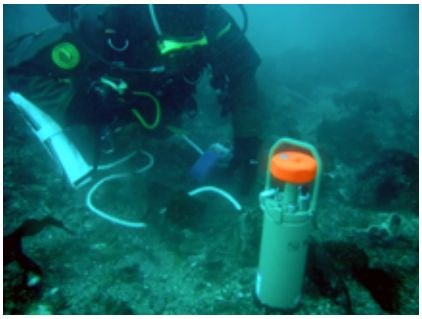


Figure 1.1 Diver taking measurements at the bottom of Kasitsna Bay [1]

Scientists have been researching life underwater for ages and have seen, first hand, the evolution of technology in marine applications. Recently, robotics has started to replace conventional, human-dependent methods of scientific research. The use of robotics in marine applications ranges from underwater remotely operated vehicles (ROVs) that are controlled through tethers, to newly developed autonomous boats that operate without human assistance or supervision. The switch to robotics has many benefits. Safety for humans increases because humans no longer dive to take measurements or capture footage of marine life. Also, the time and effort of researchers can be spent elsewhere instead of in gathering data.

Figure 1.2 Tiburon, an MBARI ROV [2]

Developers across the planet are investing time and money into research and development for marine robotics. Institutes like Monterey Bay Aquarium Research Institution (MBARI) and Scripps, and companies like Deep Ocean Engineering, have been developing underwater ROVs for decades. Over the past few years autonomous underwater vehicles (AUVs) have become a heavily developed research mechanism. Santa Clara University's Robotics Systems Laboratory (RSL) has been involved in numerous projects involving marine research vessels. From the simple SEAFOX ROV to the Mantaris and Nautilus projects, the RSL is no stranger to this growing technology. See Figure 1.3 (a-c).

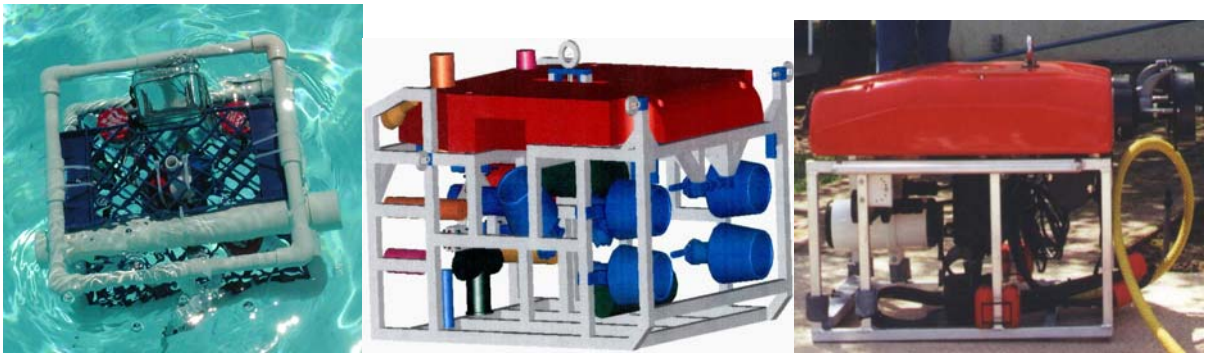


Figure 1.3 (a-c) SCU's RSL projects: a) SEAFOX, b) Mantaris and c) Nautilus [3]

Alaska is known for its dramatically changing tides and abundant marine life migrating to and from its bays and coastal areas. Kachemak bay, along the south-west coast, extends from the Pacific Ocean to Anchorage.

Figure 1.4 Map of Kachemak Bay, location of Kasitsna Bay circled [4]

Along this area there are numerous species of animals that migrate up the bay including sea otters, harbor seals, sea lions, porpoises, a variety of whales, several species of ducks and birds, fish and shellfish [5]. Within the Kachemak Bay is Kasitsna Bay (K-Bay), a small ¼ mile bay protected by the Kenai Peninsula. The National Oceanic and Atmospheric Administration (NOAA) had taken an active interest in the area and had founded the Kasitsna Bay Laboratory through the University of Alaska Fairbanks (UAF). UAF offers several diving programs as well as marine biology programs at the K-Bay Laboratory. The facility consists of resident quarters, dry and wet laboratories, several research vessels and a floating dock, completed this past year [6]. Researchers are currently studying kelp forests, crab and fish populations, and the effect of UV-B exposure in sea-urchin larvae to mention just a few [6].

Figures 1.5 (a,b) a) K-Bay's Dock and b) K-Bay's Wet Laboratory [6]

1.2 Field Literature Review

Autonomous surface vehicles (ASVs) are widely used in Europe, but have not been as heavily utilized in the United States. A British company, ASV Limited, Inc., offers several semi-submersible vessels with digital GPS capabilities [7]. Most ASVs in existence are primarily used as relay vessels accompanying AUVs. The Massachusetts Institute of Technology (MIT) developed an ASV in 1996 [8] and other US companies or individuals have created vehicles using existing boating and water recreation equipment [9-10]. However, none have the capability of lowering instruments into the water and relaying data back to the on-shore computer.

While the design and operation of our autonomous surface vessel was unique, the application to which it was applied had several other solutions already in use. Institutes like MBARI and Scripps, and companies like Deep Ocean Engineering, had been developing underwater ROVs for decades, but in order for these units to operate, a crew was always be present. Remotely operated vehicles were often used to detect maintenance issues on larger ships while the ship was docked, and in limited deep ocean exploration. Their primary constraint was the tether. MBARI's ROV *Ventana* had a collection of data samplers such as a CTD package and several cameras for both observation and navigation.

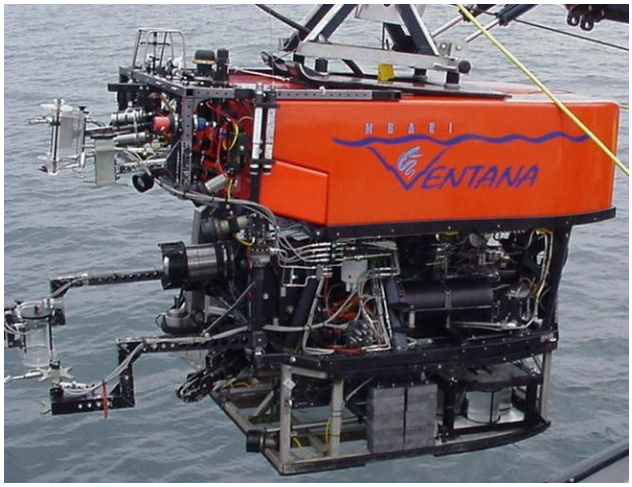


Figure 1.6 MBARI's *Ventana* about to deploy in the Monterey Bay [2]

Ventana can dive up to 2300m using its various electric motors and hydraulic pumps. However, to support such a vast array of systems of this size, *Ventana* was large (10 feet by 5.5 feet by 7.25 feet, weighing over 5000 lbs) and very expensive [2]. While ROVs were very versatile in both motion and payloads, they were still limited by a need for a support vessel and tethers.

Over the past few years AUVs have become a heavily developed research mechanism. MBARI's *Dorado* is deployed weekly in the Monterey Bay with CTD and other mid-water instrumentation. The *Dorado* can be configured with modular sections to add battery capability and other scientific sensor options, but the torpedo shape limits its mobility and periodic surfacing makes navigating to exact coordinates very difficult [2].

Figure 1.7 MBARI's AUV Dorado preparing for deployment [2]

The Spartan, currently being designed by The Naval Undersea Warfare Center, is an unmanned surface vessel. With a bow to stern length of 11 meters, a top speed of 50 knots (almost 60 MPH!), and a range of almost 1000 nautical miles the Spartan is the most capable ASV to date. The max payload of two and a half tons doesn't hurt its capabilities either, but the 120 thousand dollar base price tag, before any sensors, severely limits its availability [9].

Figure 1.8 The NUWC's *Spartan*, to be an unmanned vessel [9]

Our ASV was closer to MIT's AutoCat, which was built in 1996 to perform hydrographic surveys. The AutoCat was manufactured by Bluefin Robotics Corp. for MIT. It was 6 ft long and 3 ft 4 in wide. Weighing only 220 lbs, its electric trolling motor or gas outboard could push it around at a top speed of 8 knots for up to four hours. It used a Digital GPS navigation system which gave it 3 m accuracy for taking measurements related to marine life tracking, hydrographic/sub-bottom survey, and general oceanography [8]. While the basic design of our surface vehicles was similar, the flexibility of instrumentation for our project made it a more versatile ASV.

Figure 1.9 MIT's AutoCat running tests [8]

1.3 Project Objectives

The vision of the ASV program is to develop a research vessel that is capable of robust, automatic sampling of oceanographic data at specific locations and depths in estuary and coastal water. Our specific goal for this year was to develop an initial prototype of such a system. In pursuing this objective, our major contributions have been a structure, including pontoons, vertical struts, a platform and the connecting apparatus. This supported a winch for deployment of scientific instruments, the power source for the vessel, and the electronic components that controlled the movement and functionality of the vessel. Also, the user interface developed for the wireless communication and control of the vessel is internet based allowing the user to control the vessel anywhere in the world. Integration and testing will be performed during Summer of 2005 as part of an SCU-MBARI project. Although our work has been limited to prototyping, these contributions have been crucial to evolving the ASV concept for automated, cost-effective estuary monitoring.

2.0 SYSTEM LEVEL

2. 1 Customer Needs

At the beginning of the fall quarter we met with Geoff Wheat, a representative of the UAF, our sponsor and primary customer, to discuss the basic design specifications and requirements. On November 10, 2004, our design team met with representatives from the University of Alaska, including Geoff Wheat, Raymond Highsmith, and Jennifer Reynolds, to discuss specifications and concerns that they had regarding our design then. We also received responses to our ASV survey from Stanford Ph.D. student Jeff Ota regarding his experience and his potential project in El Salvador. Finally, we received information about the marine environment as well as typical weather conditions from Ted Otis, Area Fin-Fish Research Biologist with the Alaska Department of Fish and Game. Responses from the interviews and the survey can be found in the customer needs tables in Appendix D.

From the interviews and surveys, a general idea of what the customer wanted was compiled. These needs included:

- Ability to span a ¼ mile bay
- Ability to take measurements at 12 stations
- Ability to take measurements and travel faster than a human diver
- Ability to take measurements at a maximum depth of 330 feet
- Ability to station keep
- Ability to function in 2 foot waves
- Ability to operate through tidal changes
- Reasonable life in a salt water environment
- Ability to maneuver and maintain stability through strong winds
- Ability to fit in an air-cargo container of a 737 jet

2.2 Product Design Specifications

From the customer needs defined in the previous section, product design specifications were interpreted and summarized in this section. The complete description of the specification can be found in Appendix E.

First, knowing the ASV would be deployed in a salt water environment, the materials selected need to be corrosion resistant. The life of the vessel was estimated at 5 years, with minor parts being replaced. In order to maintain a level of adaptability, the underwater connections should be made with wet-mateable connectors (instruments and cables that are being lowered into the water must be waterproof). All components need to be protected from corrosion and impact as much as possible.

The vessel needs to traverse a ¼ mile bay, making at least 12 stops. This was interpreted into specifications for at least 8 hours of power. The large power draws are from propulsion and the vertical deployment systems. The microcontrollers are assumed to have very little power draw.

It must be able to operate in saltwater conditions with some wind, waves, and currents. The design of the vessel needs to withstand Force 3 sea conditions as defined by the Beaufort Wind Force Scale [11]. This describes wind speeds of 7-10 knots, waves of 2-3.25 feet height, with "large wavelets, crests begin to break, scattered white caps" [11]. Wind and waves can cause the vessel to tip and it can create problems for measurement integrity. The design was constrained to not have platform displacement amplitude larger than two feet.

In addition to the weight of the vessel itself, the ASV must be able to carry a payload of scientific instruments weighing up to 200 lbs, a combination of which could be mounted on the topside structure or lowered into the water. The flotation subsystem needed to balance the weight of the boat and the payload weight, and also provide positive flotation.

Using some type of navigation system it needs to be able to determine its location with at least ±10 feet of precision while station keeping and taking readings. A wireless data uplink allows communication and provides status updates to the user at a remote location. The controllers on board need to sense the current location and automatically steer and propel the ASV towards the desired location.

The ASV must also be shippable. The access to Kasitsna Bay is very limited, so the unit must be able to be shipped on a 737 cargo plane. The maximum size of a cargo container is 9 feet by 7 feet for the base with 7 feet vertical clearance.

2.3 User Scenario

The system was set up so that anyone with internet access, and the proper authorization, could control the vessel and implement mission plans. After picking each location, the program will prompt the user to determine the depths to take measurements and the type of measurements to take at each depth. When the mission plan is completed, the commands are sent through the internet to the server in the K-Bay Laboratory. The operator of the ASV deploys the ASV from either its docking location or from a trailer it is stored on and turns on the system. The commands are then sent wirelessly to the ASV. The communication system accepts the commands from the shore station and forwards them to the appropriate subsystems for execution. Typical activities include running the propulsion system to navigate to a specified location, dropping the instrument package with the winch, and pointing a camera to provide a pilot's view from the boat. The status of the boat, the power levels and the completion rate of the mission are sent back and displayed on the screen. A more complete description of the process of data transfer can be found in Chapter 5.

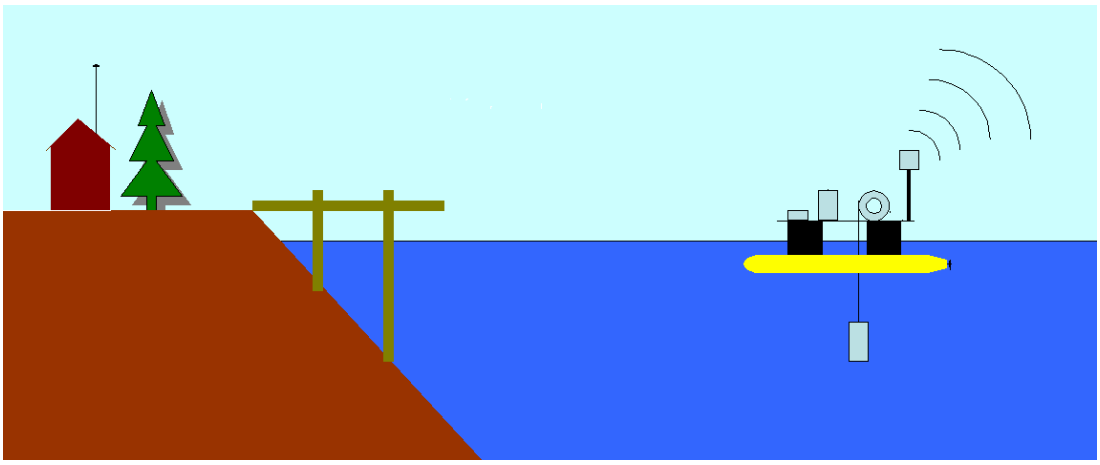


Figure 2.1 Graphical user scenario for deployment of the ASV

Using the Graphical User Interface (GUI), the user will be able to input the mission plan by picking points on the map or by typing the coordinates. A real-time uplink showes the user the current status including location, mission progress, battery life and overall health of the system,

as well as allowing the user to modify the mission plan or even operate the ASV in a manual mode using the onboard video link. Once the ASV completes the mission, it returns to the dock where the operator secures the vessel and recharges the power system. All scientific data is uploaded to the shore computer via the wireless link as well as a detailed report of the mission. If the ASV is being used several days in a row, it can be stored in the water; however, it is recommended that in order to reduce the effects of corrosion and extend the life of the ASV that it should be removed from the water when not in use. Upon removal there are a set of procedures to follow set forth in the user manual for maintenance and upkeep of the ASV found in Appendix N.

2.4 Functional Analysis

The data or electrical flow between the user and the vessel as well as the command flow from the microprocessors to the various components can be seen in the functional diagram in Figure 2.2.

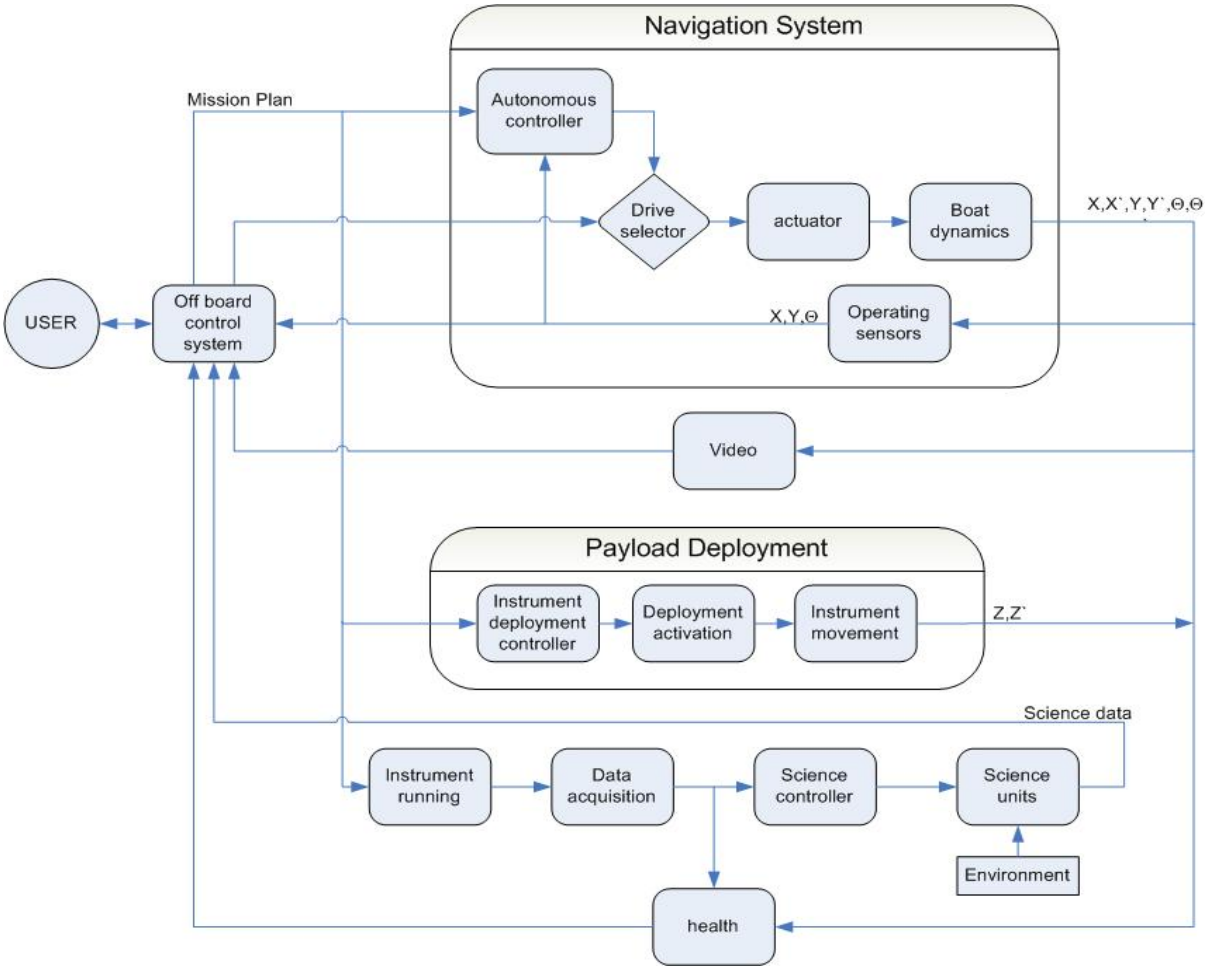


Figure 2.2 Functional Diagram of the ASV

The function of the ASV can be described through inputs and outputs. Table 2.1 describes typical user inputs and environmental inputs that the ASV would see and the outputs it would have sent to the on-shore computer.

User Inputs	**Environmental Inputs**	**Outputs**
Mission Plan	Weather Data	Mission Progress
Navigation Plan	Instrument Package Data	Health
Manual Drive	Navigation Data	

Table 2.1 Functional Inputs and Outputs.

2.5 Key System Issues

In order to meet the needs of the customer as outlined in the PDS, we had several options for the vessel: ROV, AUV or ASV. There were several key issues to consider including the integration of each subsystem, reliability of the system, manufacturability, cost, power consumption of all electrical components, flotation of the vehicle, and stability, static and dynamic balance, and the overall weight and size of the system. These issues, combined with the weather conditions and the input of the UAF's School of Fisheries and Ocean Sciences, led our team to choose an unmanned surface vehicle instead of an ROV or an AUV.

When determining what type of vehicle to utilize, trade offs were made corresponding to our key system level issues. Comparing an autonomous surface vessel, autonomous underwater vehicle and a remotely operated vehicle, the ASV would not require attachments on a submerged hull, it required fewer navigation motors, and it was easier to statically and dynamically balance (especially with two hulls). It was also easier to maintain equilibrium while taking measurements, the onboard instruments were easier to install and access, movement was not limited by a tether, the GPS was running constantly and without an accompanying vessel to relay the signal, and the above surface and sub surface measurements were taken simultaneously. However, the ASV required a vertical deployment system to lower the sub-surface scientific instruments and the overall system was heavy, possibly heavier than either the AUV or the ROV would have been.

One of the main concerns with using an AUV was that when the vessel dove to take measurements, the GPS signal faded. One of the requirements voiced by the scientists during the interviews used to determine customer needs was to have constant contact with both the on-shore computer and the GPS signal. Also, keeping engineering principles in mind, the design of the ASV needed to minimize the amount of environmental impact. With a constantly active GPS signal and the onboard video link, the whereabouts and any potential obstacles in front of the boat were known and the possibility of avoidance increased.

The final design of the ASV was a Small Water-plane Area Twin Hull (SWATH) design. Similar to a catamaran or standard pontoon boat, a SWATH vessel has two pontoons set wide to increase stability, but unlike the standard pontoon boat, the SWATH's pontoons are completely submerged 6 to 10 inches below the surface, and are connected to the main deck by vertical struts. This design increased the stability of the vessel by decoupling the boat from the oscillation due to the waves. This was especially important when the instruments were deployed because oscillations of the platform changed the depth at which the instruments were reading from, and can cause high stress on the cable. Having the pontoons spread out helps to keep the center of buoyancy and the center of gravity move closely aligned, reducing the tendency of the vessel to flip over. We decided to name the ASV the WASP, or Water-Analysis Surface Platform.

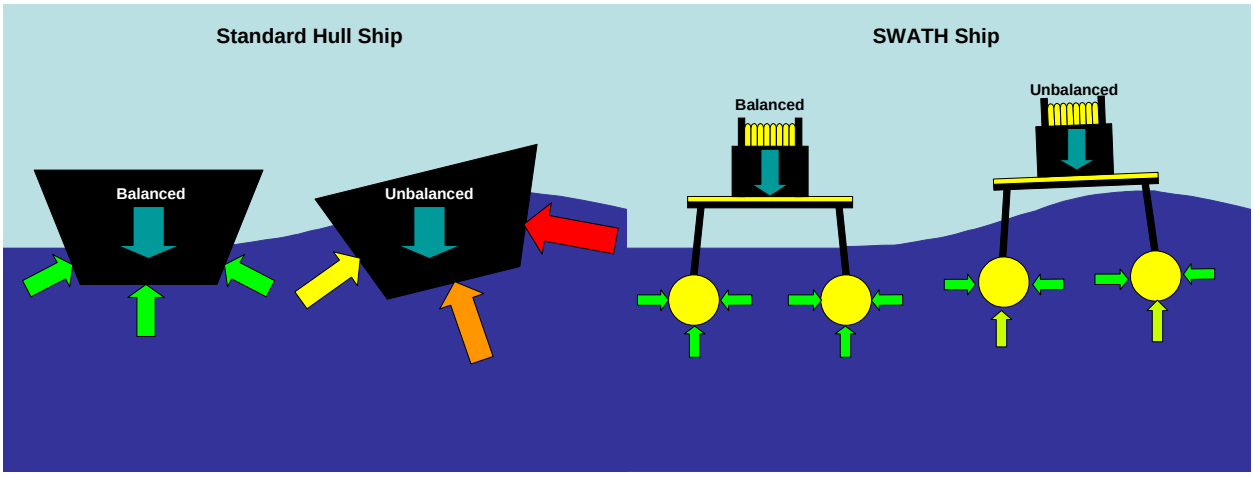


Figure 2.3 Standard Hull Vs. SWATH Design

2.6 Team and Project Management

Our design team consisted of six students: two mechanical engineers, Robyn Kobashigawa and Paul Mahacek, two electrical engineers Aaron Schooley and Daniel Wilson, and two computer engineers Nazli Zooleh-Yaghoob-Shahi and Nader Mehdizadeh. Our faculty advisors on this project were the Robotics Systems Laboratory Director, Professor Christopher Kitts, and the Mechanical Engineering Department Chair, Professor Timothy Hight. The diversity and broad experiences of the team members complemented each other, and created a strong group able to accomplish our goals.

Each of the members had many jobs and responsibilities. Paul was the facilitator and the main contact with the faculty advisor. Robyn was the recorder and the dynamic control engineer. Aaron was the chief processor engineer working on the electrical system, and Dan was the wireless link specialist. Nazli and Nader were responsible for the creation of the Graphical User Interface (GUI) and the storage system for acquired data.

The project was divided and set-up so that each discipline could work independently with common interfaces previously agreed upon. The mechanical work focused on the hardware necessary for the structure of the boat, the means of flotation, propulsion and steering, and the mounting and waterproofing of all components. The electrical scope spanned all components on the ASV that required power, the microcontrollers that communicated with the on-shore computer and controlled all functions of the ASV and all of the interfaces with the instruments. The computer portion of the project was to write and compile the coding for the GUI. This included the waypoint navigation system mapping, selection of locations, user determination of instruments, and communication with the ASV. Together, the team integrated and tested the system, an autonomous vessel that floats, supports instruments, has functioning electrical components, and can be controlled using a simple GUI through a wireless connection. Each discipline has corresponding documentation for its subsystems and can be found in Chapters 3-5.

2.7 Budget

We set out to develop a low cost ASV and recorded our spending to determine how well we followed our budget. The budget to accomplish the Mechanical goals was broken into major

subsystems and components including flotation, platform, thrusters and winch. The Electrical budget consisted of batteries, processors, circuit boards, and other components. The instruments were the most expensive budgeted items and were not purchased this year. The costs for the major components are listed in Table 2.2.

	Preliminary Budget	Current budget	Spent	Donated	Projected	Total
Flotation	$1,000	$350	$350	$150	$50	$400
Platform	$500	$300	$0	$300	$0	$300
Thrusters	$3,000	$1,600	$1,600	$0	$0	$1,600
Winch	$2,000	$8,493	$7,334	$0	$1,200	$8,534
Batteries	$1,000	$836	$836	$0	$0	$836
Processor	$1,000	$1,000	$1,000	$0	$0	$1,000
Circuit Boards	$100	$100	$0	$1,500	$0	$1,500
Radio Modem	$400	$400	$400	$0	$0	$400
Camera and Video	$900	$900	$900	$0	$0	$900
Total ME and EE	**$13,400**	**$17,542**	**$14,820**	**$2,950**	**$2,650**	**$20,420**
Instruments	$65,150	$65,150	$150	$1,500	$50,000	$51,650
Total	**$78,550**	**$82,692**	**$14,970**	**$4,450**	**$52,560**	**$72,070**

Table 2.2 Budget

2.8 Project Challenges

Most of our project challenges arose from the uncertainty of the weather data in the bay. The conditions of the waves, winds, currents, and obstacles can never be predicted with 100% reliability; therefore we recommended the types of conditions the ASV can be sent out in. Because we cannot anticipate all wave heights, steepness and frequency of the waves, we have designed the ASV to be deployed in Force 3 conditions [11]. This recommendation was made so the vehicle would not be deployed in crashing waves that could damage sealed components or any other component onboard, or cause excessive vibrations.

We designed the ASV to withstand 7-10 knots wind speed. Our major concerns were the stability and performance of the vehicle. With increasing wind speed, the platform had a tendency to flip the entire vehicle over. Combined with waves, this could have been detrimental to the ASV and all onboard components. As for performance, the propulsion motors were sized to overcome the water drag and the wind forces up to specified levels. Steering and station keeping was increasingly difficult with the culmination of wind, waves and current.

Another weather uncertainty was the currents, both at the surface and at depth. Currents, like winds and waves, could move the boat in a direction not originally desired by the user. Underwater currents would also create difficulty in placement of the instruments in specific locations. However, we assumed the measurements could be taken in an area of 10 feet radius and if the instruments do not drop vertically, the measurements were still representative of the location. We anticipated the thrusters running during the duration of each mission to keep as steady a position as possible.

3.0 MECHANICAL SUBSYSTEMS

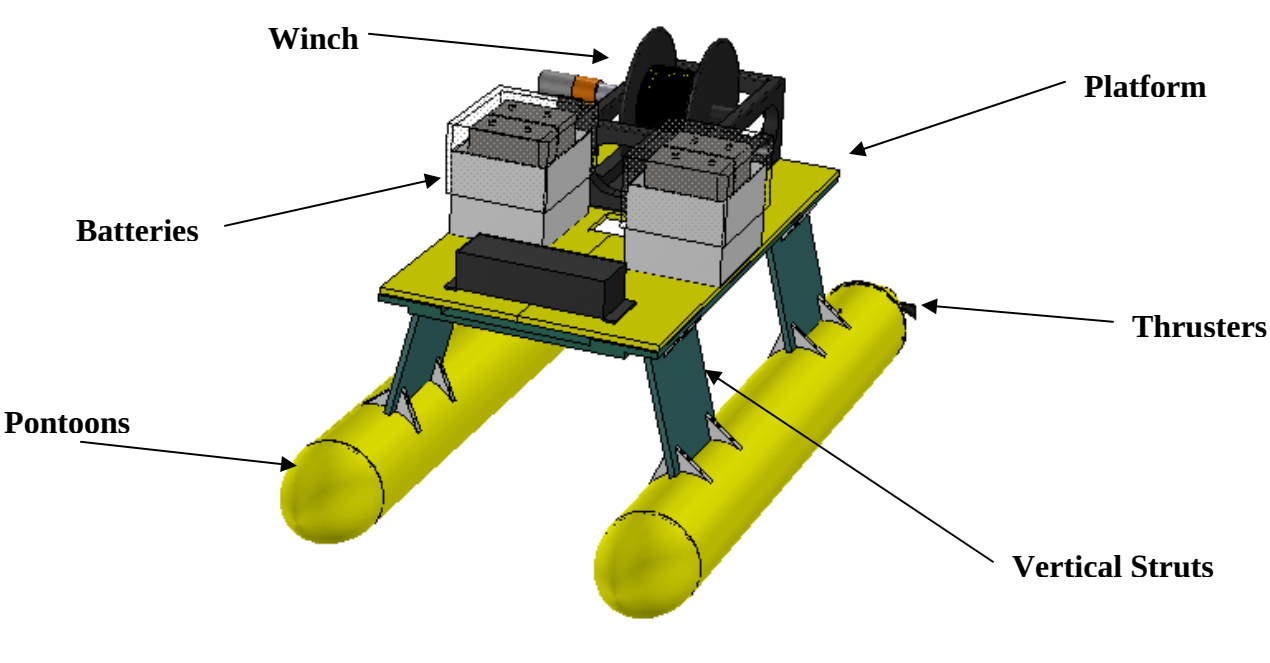


Figure 3.1 Current Design of the ASV

The entire ASV was broken down into discipline-related sections: Mechanical, Electrical, and Computer subsystems. The mechanical subsystems deal with the physical appearance and functionality of the ASV. The electrical subsystems control the movement and data acquisition of the ASV. Finally, the computer subsystems create the human interface between the ASV and the user. Each subsystem had a list of alternatives and a trade-off analysis was performed in the form of a selection matrix, found in Appendix F.

The mechanical subsystems are comprised of a flotation subsystem, power subsystem, propulsion subsystem, and a vertical deployment (or instrument deployment) subsystem. These subsystems control the overall movement of the ASV and the instruments that are deployed.

3.1 Flotation Subsystem

In order for our ASV to operate, it was vital for the vehicle to stay at the surface. Some type of flotation or buoyancy system was decided upon based on a variety of selection criteria. The design team brainstormed many potential solutions for flotation ranging from containers filled with air to less dense materials than water. Listed are the most important functional criteria from

most to least importance: stability, buoyancy, complexity, versatility, reliability, strength, and size (when broken down for shipping). After determining the criteria, each design solution was evaluated according to the criteria and ranked for compliance. The design solutions for the flotation subsystem: Inflatable Pontoons, Fishing Boat Base, Semi-Submergible, Foam Filled Pontoons, and PVC Tubing with Fiberglass End Caps, were listed below with their associated benefits and drawbacks.

Design Solutions:

Inflatable Pontoons - The idea for using inflatable pontoons came from river rafting. By inflating a high density nylon fabric tube to use as flotation, many of the design criteria were fulfilled. This idea scored well in several sections including density, overall time for designing, manufacturing and testing, and especially the size for shipping as the tube could be deflated to take up almost no room. Some of the areas causing the most concern included the strength, rigidity, and reliability of the tubes. Several vendors were found for this product, and one in particular did custom orders in the $800 range. The final decision was that the tubes were too unreliable to use for such an expensive piece of equipment for extended periods of time. The sketch of this solution can be seen in Figure 3.2.

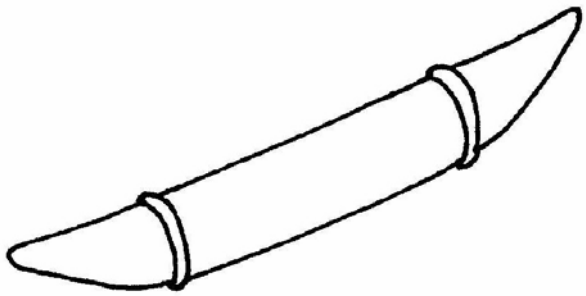


Figure 3.2 Sketch of Flotation Design Solution: Inflatable Pontoons

Fishing Boat Base – This idea was based on a small (1-2 man) fishing boat. Several companies make small boats out of metal, fiberglass or high strength plastic, and most any of these can be found in the $500-$700 range. The fishing boat scored best in the reliability, strength, and rigidity categories, while its limitations in shipping size and weight kept it from being selected as the final choice. The sketch of this solution can be seen in Figure 3.3.

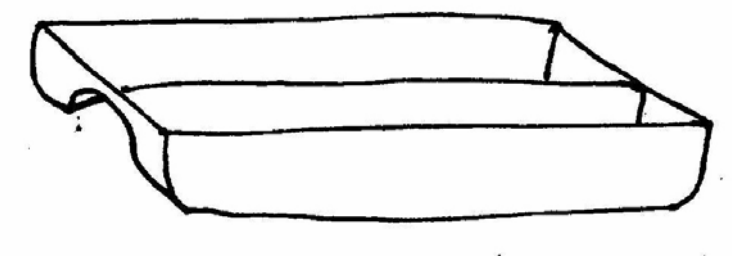


Figure 3.3 Sketch of Flotation Design Solution: Fishing Boat Base

Semi-Submergible – The semi-submergible was one of the most revolutionary ideas out of the UK in ASVs. While it was a very high tech solution its overall score was the lowest mainly due to its complexity and stability issues and the added requirement it would need to integrate all the parts to meet the design criteria, especially lowering the rather large payloads. The sketch of this solution can be seen in Figure 3.4.

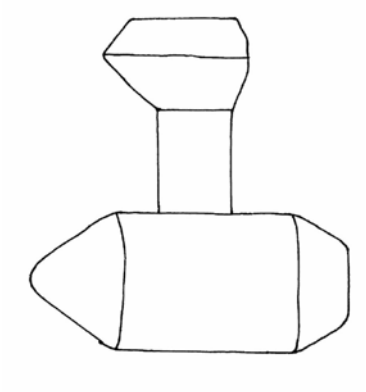


Figure 3.4 Sketch of Flotation Design Solution: Semi-Submergible

Foam Filled Pontoons -Originally, this idea was based on pourable polyurethane foam but after further analysis and research it was decided that a solid polystyrene material would minimize the irregular properties and shrink involved with the polyurethane. The rest of the flotation structure would include an internal structural matrix of steel (similar to rebar in concrete). This would have given the pontoon a point to mount onto the main structure. In order to protect the foam from the harsh UV rays and ocean environment, a hard fiberglass shell would have been molded around it, also giving it resistance to light and medium impacts. Two of these steel-reinforced, fiberglass-covered, foam pontoons would have been used to float the ASV. The foam scored well in the areas of stability, and size due to its ability to be broken down into four parts for shipping. The section in which the foam did the worst was simplicity and rigidity due to the

numerous parts and the non-uniformity of the pour foam. The dynamic response of the two-part pontoons and the fatigue the structure holding the parts together would undergo was a concern in this design. Although it did not receive a very high weight factor, the complexity of using these pontoons will have more ramifications down the line especially in the calculation of its strength and the actual manufacturing of the floats. After careful consideration, this solution is unreliable due to the dynamic impact of the pontoons on the structure it supports. The sketch of this solution can be seen in Figure 3.5.

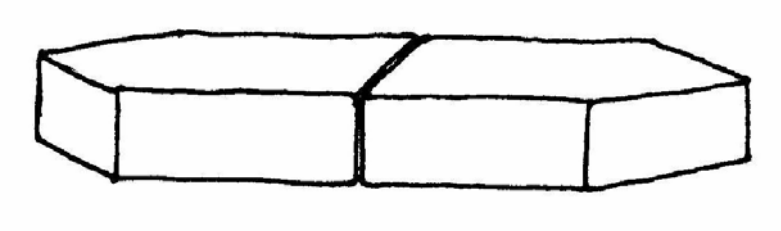


Figure 3.5 Sketch of Flotation Design Solution: Foam Filled Pontoons

PVC Tubing with Fiberglass end caps-When considering possible fatigue damage due to multiple-part pontoons, returning to the idea of single piece pontoons was the favorable option. By using poly-vinyl chloride (PVC) pipes used for plumbing, the pontoons would have been rigid and without the fatigue issue. In order to close the piping on the front and back ends, end caps needed to be designed and mounted. The solution was to create the end caps out of fiberglass and paint them to match the PVC. To create a large buoyancy effect, a large amount of water needed to be displaced. The pontoons were filled with foam because the pontoons were not designed to be sealed. This allowed for access to the inside of the pontoons for mounting various parts. When this design was considered, the benefits were good stability, rigidity, manufacturability, cost, and life. Some drawbacks this design raised were weight, ballasting, and complexity of end caps. The sketch of this solution can be seen in Figure 3.6.

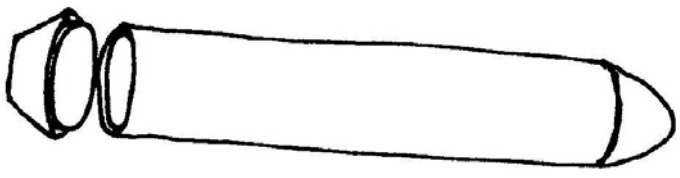


Figure 3.6 Sketch of Flotation Design Solution: PVC Tubing with Fiberglass End Caps

Final Design:

Using the selection matrix seen in Appendix F-1, the highest scoring solution was PVC tubing with fiberglass end caps. The final design for the pontoons can be seen in Figure 3.7. Knowing common diameters of PVC in the construction industry and calculating the approximate amount of water needed to be displaced, we chose a 15 inch diameter pipe that was cut to 8 feet in length (calculations can be found in Appendix G). The shape of the end caps were modeled after an existing AUV shell that had a rounded nose and thrusters protruding out of the end. In the SWATH design, the pontoons were submerged and the platform was raised above the water. The pontoons were designed to ride two to four inches below the surface. By submerging the pontoons, the stability of the craft was raised because of the lessened impact of waves combined with the lower center of gravity. Unfortunately, to keep the pontoons submerged when the instruments were deployed, a dynamic ballasting system needed to be designed. This system was not designed, but was included in the patent search in Chapter 8 and in the Future Work section of the Conclusion.

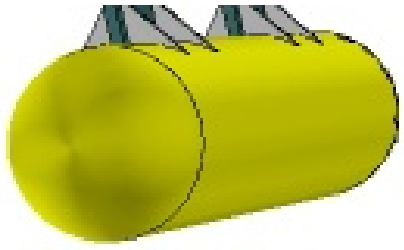


Figure 3.7 CAD drawing for pontoon design

Manufacturing Process:

The process of manufacturing the flotation subsystem was broken into five parts: sizing the PVC, fiberglassing the end caps, pouring the foam inserts, painting and finishing all parts, and assembling the pontoons. To size the 15 inch PVC pipes, they were cut to 8 feet length and the ends were sanded. To get the shape of the end caps, an existing AUV was used as a mold. A novel idea when creating the end caps was to use a section of PVC at the base, where the end cap was designed to connect with the piping of the pontoon, and mold the fiberglass to the AUV and over the PVC. This created a surface that fit securely into the PVC and made mounting the final end caps easier. Each end cap was constructed of two halves and fiberglassed together. The surface was sandpapered and filled to create a smooth surface ready for painting. See Figure 3.8.

Figure 3.8 (a-e) Process used to create end caps from left to right: a) existing AUV shell used as a mold, b) end caps were created in halves, c) halves were joined, d) end caps were sanded and filled, and e) end caps were painted

In order to displace the amount of water needed to float the ASV, the pontoons needed to be filled with foam. The foam was dimensioned ½ inch smaller in diameter than the inside of the PVC so that it can be easily removed and reinstalled. Excess sections of the PVC used to make the pontoons were lined with a cloth and taped for easy removal of the foam and for reuse of the mold. The foam was poured into the mold, removed and the surface was finished. Several sizes of foam inserts were made which allowed for adjustable distribution of buoyancy. See Figure 3.9.

Figure 3.9 (a,b) a) Picture of the Mold for the Foam and b) a Foam Insert

3.2 Propulsion Subsystem

Because the ASV needed to take measurements at several locations, it needed a drive system. The different options for this subsystem were ranked by how well they scored based on several criteria; the most important was thrust followed by reliability, and power consumption. Several

different options and configurations for the ASV's propulsion system were a Structure-Mounted Gasoline Motor, Pontoon-Mounted Electric Motors, and Platform Mounted Electric Motors.

Design Solutions:

Structure-Mounted Gasoline Motor - The gas powered thruster was the most common type of motor for both large and small commercial human operated boats. It allowed for the maximum amount of power and thrust, and the gas thruster scored very well in those areas. However it did not score well in the reliability category due the required maintenance associated with so many parts operating in the Alaskan environment, not to mention the complexity of integrating an electronic control system into a gas motor. The sketch of this solution can be seen in Figure 3.10.

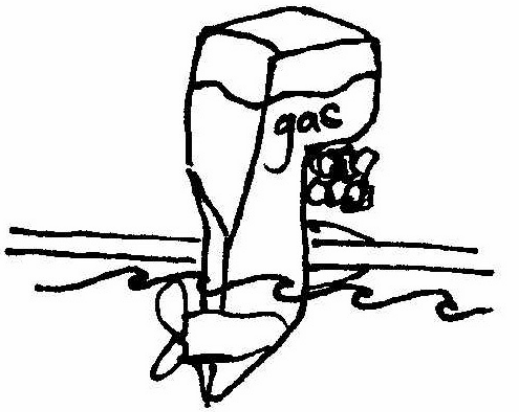


Figure 3.10 Sketch of Propulsion Design Solution: Structure-Mounted Gasoline Motor

Pontoon-Mounted Electric Motors - The idea to use two stationary electric motors came from the current ROVs in use. A differential drive was designed: by thrusting at different rates or in different directions, turning was achieved from stationary thrusters. This would have given the ASV a full range of horizontal motion with only two motors. It also would have eliminated a small portion of the drag resistance by decreasing the surface area. There was less of a safety issue with this design due to the absence of parts extending from the platform into the water. The biggest downsides to the pontoon-mounted thrusters were concern about the application of all the thrust on the pontoons themselves and the complexity of fabricating pontoons that integrate well with this design. This design did score well in weight and reliability due to the reduced amount of material holding the thruster up and the decreased amount of areas where

failure could occur such as electric steering or the shaft that would attach the thruster to the deck. The sketch of this solution can be seen in Figure 3.11.

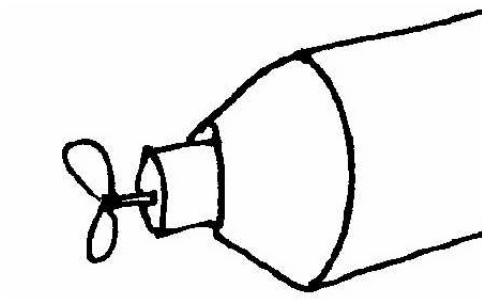


Figure 3.11 Sketch of Propulsion Design Solution: Pontoon-Mounted Electric Motors

Platform-Mounted Electric Motors – This option consisted of two platform-mounted trolling motors with electric steer and 360° of rotation. After taking into consideration the amount of wind and current in the bay, it was determined that it was necessary to utilize two motors in order to have enough thrust to complete any long distance mission in a reasonable amount of time. However, the use of the electric steer complicated the control of the ASV and the safety concerned with the thrusters hanging from the platform was considered. This solution scored well in the thrust and simplicity area because it did not require modifications. The sketch of this solution can be seen in Figure 3.12.

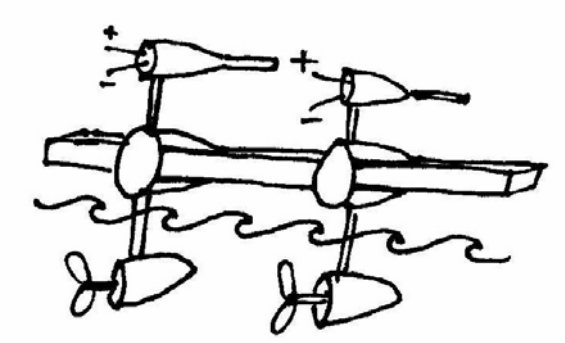


Figure 3.12 Sketch of Propulsion Design Solution: Platform-Mounted Electric Motors

Final Design:

The solution we chose was pontoon mounted electric motors. The thrusters that were purchased were Minn Kota Electric Trolling Motors. These provided 55 pounds of thrust each and drew a stall current of 35 Amps. The motors, without the shaft that led to the electric steer control, were

4 inches in diameter at the widest and 13 inches long. Specifications on these components can be found in Appendix L-2. The drawing of the final design can be seen in Figure 3.13.

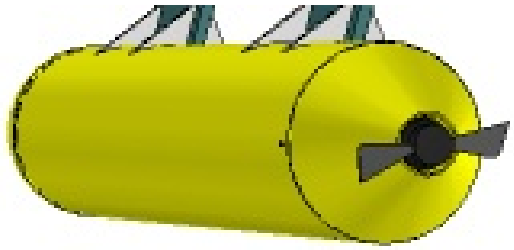


Figure 3.13 CAD drawing of pontoons with thrusters mounted

Manufacturing Process:

The thrusters chosen to drive the ASV were modified Minn Kota Electric Trolling Motors. They had a shaft enabling mounting on a surface approximately two feet above water as well as fins along the motor for better hydrodynamics. However, because the thrusters were embedded in the pontoons, the shaft and the fins needed to be removed and the thruster sealed up. Using a common sawing tool and grinder, the shaft and fins were removed and the surface was smoothed down. The wiring for the motor ran through the shaft; therefore the cut area was sealed with epoxy. See Figure 3.14a.

To mount the motors into the pontoons, a 4.5 inch acrylonitrile butadiene styrene (ABS) pipe was used. See Figure 3.14 (a-c). The pipe was cut to size and a section was cut out for the protruding shaft section and wiring to slide in. A hose clamp was placed around the mount and the motor to secure it in the mount. The protruding shaft helped to keep the motor from spinning within the mounting. To mount the ABS pipe into the pontoon, two ¼ inch diameter all-thread rods were bolted through. Finally, the end caps were filled with foam to add extra support to the thrusters.

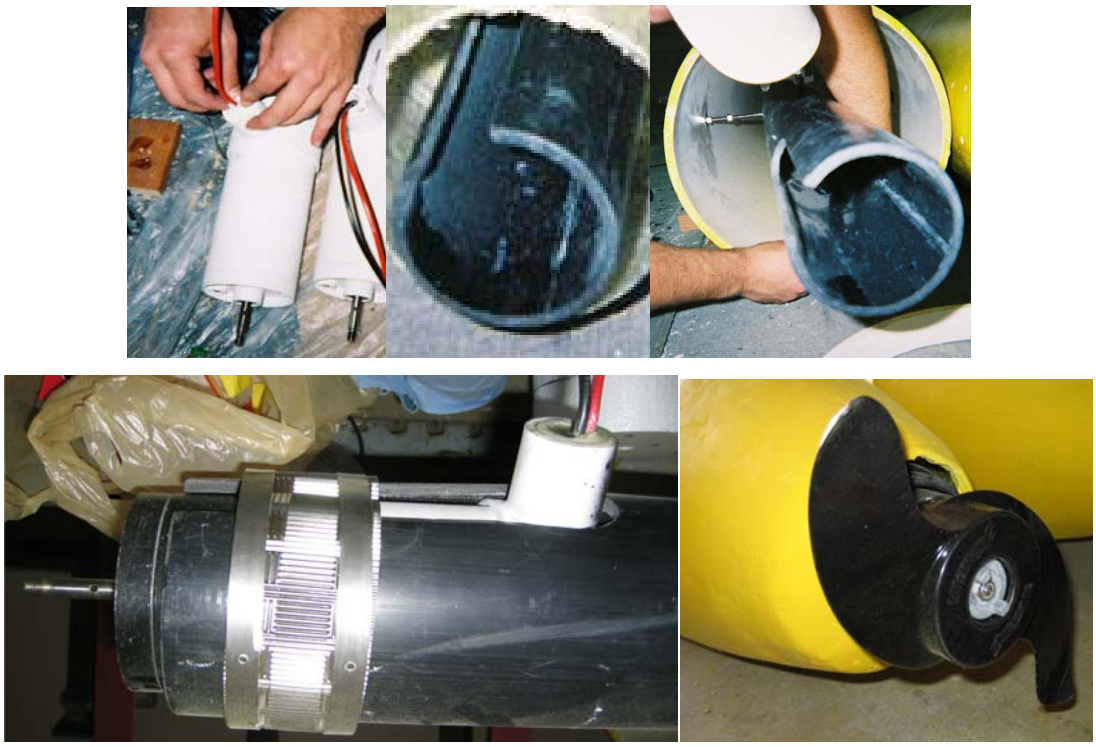


Figure 3.14 (a-e) From left to right, top to bottom: a) filling the cut shaft with epoxy, b) slot cut out from ABS for mounting thruster, c) bolts holding mounting in pontoon, d) thrusters mounted, and e) end cap mounted

3.3 Power Subsystem

The power system was very dependant on the selection of the propulsion system, and the length of the mission because of the large power draw of the thrusters relative to the other systems. The main factors besides capacity were weight, size, and availability. Other factors included operating life, simplicity, cost, the number of parts and the amount of time needed to build mounts and recharge the system. The first criteria that needed to be met was the required amount of amp hours the system needed to provide. The design solutions were Batteries, Solar Panels, a Gasoline Generator, or a Hybrid system with both a gasoline generator and batteries.

Design Solutions:

Batteries - The first option considered was very straightforward. Both the availability and simplicity of using batteries as the sole power source made it a good choice. The limiting factor of this option was that, in order to get a decent run time, quite a few batteries were needed, creating a lot of weight and taking up a good portion of the space on the platform. However,

batteries scored high in the simplicity and number of parts required section of the selection process. The sketch of this solution can be seen in Figure 3.15.

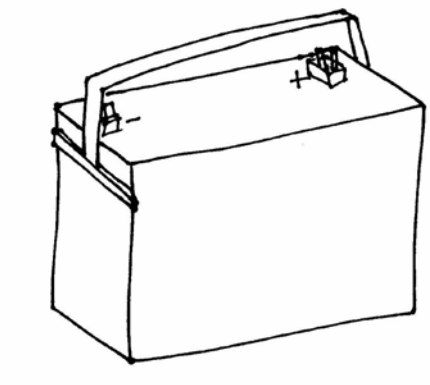


Figure 3.15 Sketch of Power Design Solution: Batteries

Solar Panels - The solar powered option was not a viable option. For the amount of power required, it would have required about 10 square meters of solar panels to run the system. Also, this project was going to be in Alaska where cloud cover and seasonal darkness would have limited the ASV to running for about 50 hours a year. The sketch of this solution can be seen in Figure 3.16.

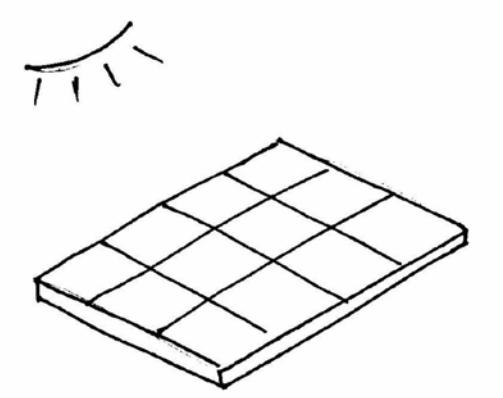


Figure 3.16 Sketch of Power Design Solution: Solar Panels

Gasoline - The gas powered generator had the second highest score in the selection matrix. It did very well in most of the categories but concern for interference with sensitive electronics raised some issues. Also, after selecting the propulsion design, pontoon mounted electric motors, the gasoline generator did not seem the best option. The system would have required at

least a minimal battery to operate, which we took a step further to our last choice. The sketch of this solution can be seen in Figure 3.17.

Figure 3.17 Picture of Power Design Solution: a Honda gasoline generator [12]

Hybrid - The Hybrid system gave us the best of both worlds. Four deep cycle marine batteries gave us a good starting run time, and a generator put back a portion of the power, almost doubling the time while only adding a fraction of the weight. The Hybrid scored very well in every category, especially weight and operating life. The current design called for the use of four 6V batteries with 740 Amp hours, which gave us about six hours of battery life. The addition of a relatively small generator (2000W, <50lbs) increased the mission time by at most ten hours.

Final Design:

The design solution we decided upon was four the Hybrid system. However, we were unable to purchase the generator and implement it in our hardware. We did design for the addition of a generator next year, as described in the Future Work section of the Conclusion. The batteries chosen were Sea Volt 6 Volt deep cycle marine batteries each weighing 113 lbs. Each also has a capacity of 370Ah, 840 reserve minutes, and is capable of putting out 1900 marine cranking amps.

Batteries	6V
	370 Ah
	700 cycles before losing half capacity
	113 lbs

Table 3.1 Specification of Batteries

Manufacturing Process:

Although we did not modify the batteries, we needed to make slight modifications to the battery boxes. The battery boxes that were purchased had space length- and width-wise, but not height-wise. Because there were no boxes that could accommodate the batteries purchased, the base of the originally purchased boxes were used with a plastic container as the lid. This still provided protection from wind and rain, yet allowed air to exchange between the environment and the batteries and kept mounting simple. See Figure 3.18.

Figure 3.18 Modified battery box

3.4 Vertical Deployment Subsystem

The deployment of the instruments was one of the most difficult subsystems. Finding a feasible way in which to deploy the onboard instrumentation with a light weight and compact unit without sacrificing strength had proved a very challenging task. The system requirement specified that the instruments reach about 660 ft in depth. Other important selection criteria were weight, size, time, strength, simplicity, cost, and maximum speed for submergence. Several design solutions were: a Retractable Hydraulic Arm, Attachable ROV or a Cable Handling System.

Design Solutions:

Retractable Hydraulic Arm – The arm idea relied on a series of concentric tubes that collapsed on one another to raise and lower the instruments. This option scored very low on many of the criteria, especially on the strength. In order to reach the desired length many tubes—approximately 200 arms, each having length of 3 feet—would have been needed, making this an

infeasible option in consideration of weight, cost, and simplicity. The sketch of this solution can be seen in Figure 3.19.

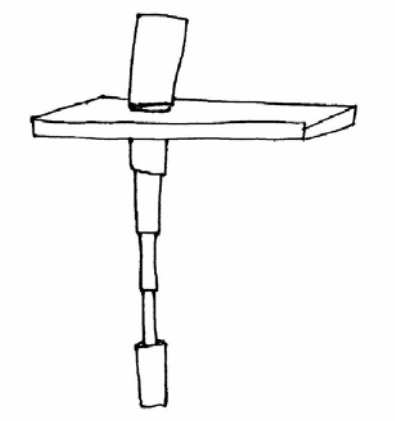


Figure 3.19 Sketch of Vertical Deployment Design Solution: Retractable Hydraulic Arm

Attachable ROV – One option was to develop a very small ROV to deploy the instruments. This would have been a good method if power, time, and money were not an issue. Essentially a small ROV would have been operated by the ASV using sonar to keep the payload directly below the ASV. This would have given us a very precise measurement and could compensate for the current causing the instruments to drift. However a tether handling system would still have been needed, so this was not a viable solution. The sketch of this solution can be seen in Figure 3.20.

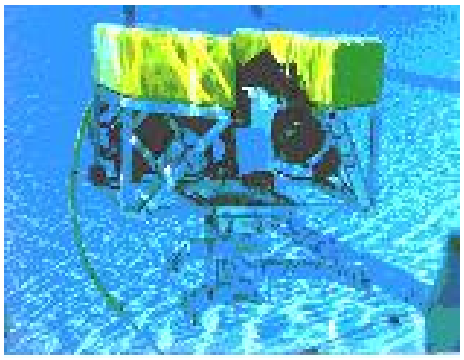


Figure 3.20 Picture of Vertical Deployment Design Solution: Triton, an existing ROV [3]

Cable Handling system – The cable handling system was the most feasible choice for the instrument deployment. Basically, a large winch motor wound or unwound a cable with conductors in it. By utilizing a slip ring, the cable rotated without putting a kink into the line.

One system with a slip ring was developed as a senior design project a few years ago and was available for retrofit but it was limited to only 170 feet, quite short of our goal. The existing unit also ran off of a 75V motor which wouldn't work with our 24V system. An adapter could have been used with the 120V from the generator to make it work. Several vendors have been found that could make custom systems, and it was the best option to order a system. The Cable Handling System fulfilled the entire criteria well, except the amount of time it would have taken to design, build and test the system. The sketch of this solution can be seen in Figure 3.21.

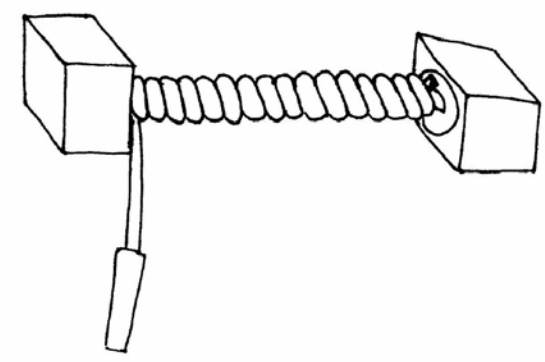


Figure 3.21 Sketch of Vertical Deployment Design Solution: a Winch

Final Design:

The solution we chose was the cable handling system, or winch. We decided to purchase a custom winch from a company in Canada called Shark Marine. The winch had an 8-conductor cable, slip ring, and a built-in level wind, seen in Figure 3.22. The purpose of the level wind was to ensure the cable wound properly. The level wind, the two gray poles with the black block at the end, traveled back and forth across the grooved bar to lay the wire neatly. The motor that was designed for the winch was more than what was required to run the winch, so a separate motor was investigated and purchased. The technical specifications for the winch and reel can be found in appendix K-1.

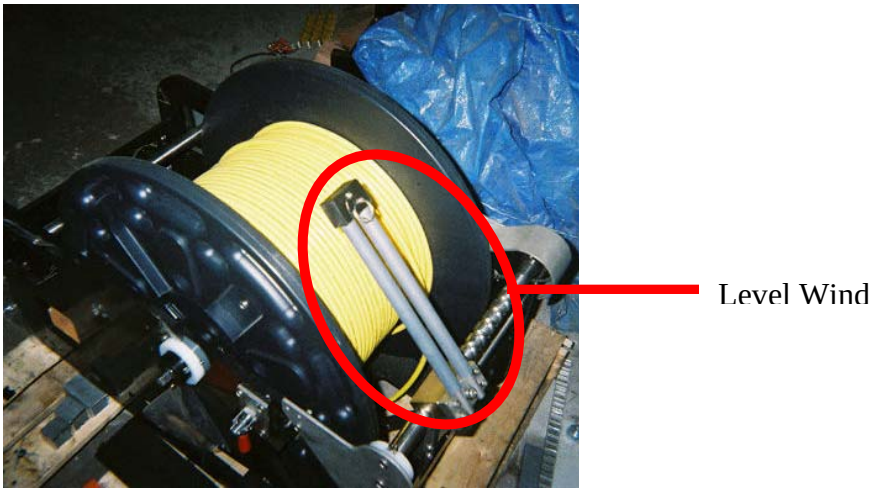


Figure 3.22 The 8-conductor winch purchased from Shark Marine comes with a level wind

Manufacturing Process:

The motor needed to be modified to work with the winch. Due to the desired gearing of the winch motor, the original reel was removed and a mate from the severed reel to the purchased cable handling system was lathed. Because the motor and main reel were purchased separately, a mount needed to be designed and manufactured to lift the motor to be in-line with the drive shaft. This mount was constructed of three pieces of honeycomb aluminum—cut, filled and painted—and was bolted to the grating and to the reel itself. The original motor, the modified attachment, and the motor mounted onto the platform can be seen in Figure 3.23.

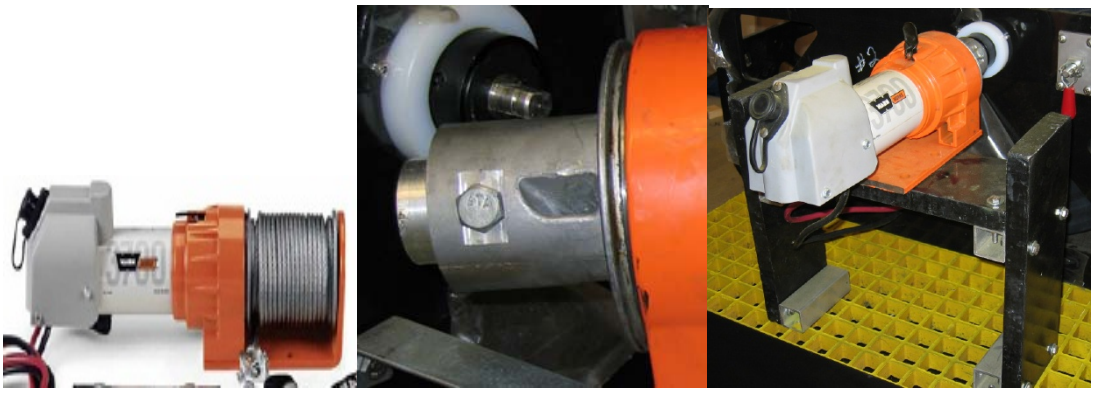


Figure 3.23 (a-c) a) Originally motor purchased, b) motor with reel removed and mate attached, and c) mount for the modified winch motor [13]

3.5 Structure

During the preliminary design stage, the design solutions for the structure were diverse, but the primary function was to create a surface to mount all equipment of the various subsystems and to connect this platform to the flotation. Criteria for selection included strength, rigidity, weight, manufacturability, cost, time, and simplicity. The design solutions for the structure were: Angle Iron Frame, Bent Steel Tubes or Honeycomb Aluminum platform and vertical struts.

Design Solutions:

Angle Iron Frame-This design utilized an angle iron frame, for the platform, that supported a surface either made of fiberglass grating or a thick sheet of metal. The fiberglass grating provided a hard, stiff platform and also made mounting and adjusting components easier. Although the design would have been simple and most materials fairly inexpensive, the strength and rigidity would have relied upon the strength of the welding. Welding was not an area of familiarity and the manufacturability and cost to produce was a negative issue for this design. Also the weight of all the materials used gave it a low score. The sketch of this solution can be seen in Figure 3.24.

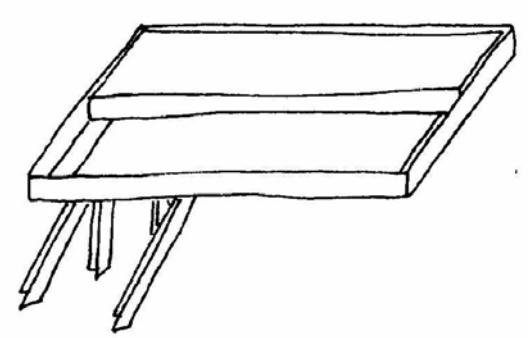


Figure 3.24 Sketch of Structure Design Solution: Angle Iron Frame

Bent Steel Tubes-This design was similar to the angle iron frame, but used bent tubes to support a platform material and connect it to the pontoons. This design required fewer welds, but created integration problems with the pontoons. It scored well for strength, rigidity and simplicity, but no so for weight, manufacturability-the tubes needed to be bent and there was some welding between the platform and tubes, time and cost. The sketch of this solution can be seen in Figure 3.25.

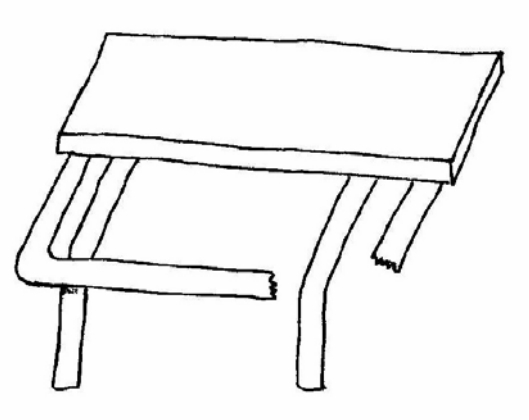


Figure 3.25 Sketch of Structure Design Solution: Bent Steel Tubes

Honeycomb Aluminum Platform and Vertical Struts-Honeycomb Aluminum, a light weight yet strong material, had many applications in the Aerospace Industry and in several projects by MBARI. Plates of the honeycomb aluminum would have been used to support a plastic grating. The plates were also designed as vertical struts-plates tilted at a slight angle from vertical, supporting and connecting the platform to the pontoons. This solution scored very well in the weight, strength, and simplicity areas. With the support of an advisor, the materials were donated, cutting costs, and their manufacturing facilities were available, cutting time and easing manufacturing. The sketch of this solution can be seen in Figure 3.26.

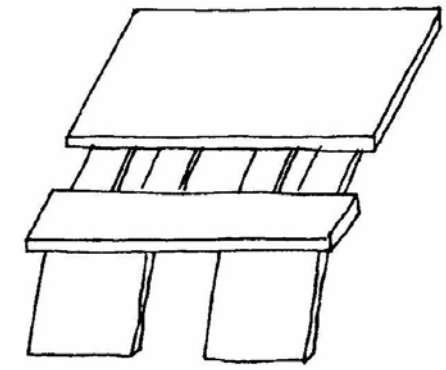


Figure 3.26 Sketch of Structure Design Solution: Honeycomb Aluminum Platform and Vertical Struts

Final Design:

The design chosen was the Honeycomb Aluminum Platform and Vertical Struts. The platform dimensions were 6.7 feet by 3.75 feet and the vertical struts were 1.5 feet by 1.7 feet at a 15 degree angle, giving 1.6 feet between the platform and the pontoons. Two1.1 foot by 6.7 feet

strips ran from fore to aft and 4 2.9 feet by .8 feet strips ran from bow to stern to support the platform. The fiberglass grating-two pieces 1.9 feet by 6.7 feet-was bolted on top of the aluminum plates for mounting components. By using the aluminum only as a support and the grating as the actual platform, the surface are was decreased, allowing more air and water to pass through instead of hitting the platform and flipping the vessel. The CAD drawing of the final design for the structure can be seen in Figure 3.27.

Figure 3.27 CAD drawing of final design for structure

The use of vertical struts constructed of the aluminum plates coincided with the SWATH design. The vertical struts were designed to be partially submerged. The strength of the plates was not tested, but the strength of the plates can be calculated using the material properties in Appendix F. The integration of the vertical struts to the platform required connectors or clamps. Also, the integration of the vertical struts to the pontoons required clamps, preferably with arc-shaped bases for flush mounting onto the PVC pontoons. These clamps were designed to be approximately 3 inches by 4 inches. Before manufacturing the clamps out of ultra-high molecular weight (UHMW) plastic, a finite element analysis was performed. The CAD drawings for the vertical struts and the clamps can be seen in Figure 3.28.

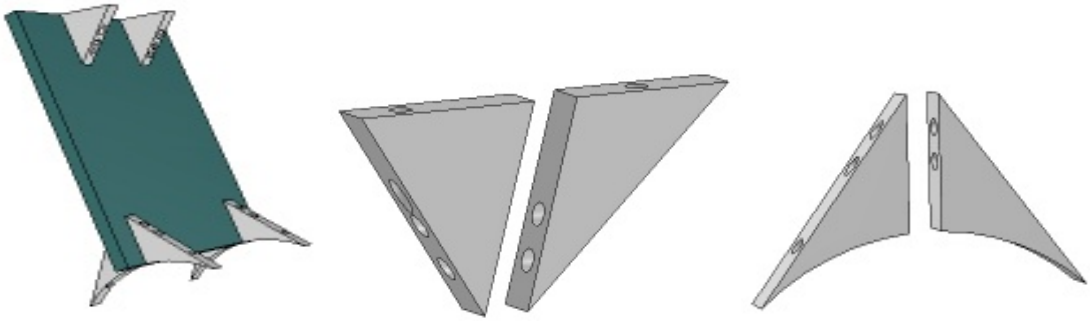


Figure 3.28 (a-c) CAD drawing of a)vertical struts with clamps attached, b) top clamps, and c) bottom clamps

Manufacturing Process:

As described in Chapter 3.5, the structure was constructed of honeycomb aluminum plates. They were cut to size using a band saw and the edges were finished with a filler to prevent water from leaking in and to make the edges safer to handle. Each piece was finished with a coat of black paint. To bolt through the honeycomb, a ½ inch hole was drilled and a ¼ inch inner diameter bushing was adhered within. For mounting of all components, ¼ inch stainless steel bolts and nuts were used along with stainless steel washers or flat stock. Top and bottom mounting can be seen in Figure 3.29.

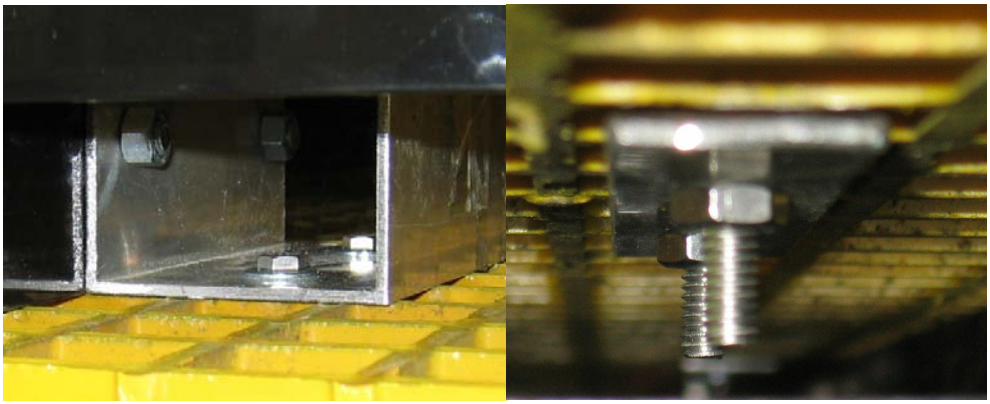


Figure 3.29 (a, b) Mounting bracket for a) topside and b) bottom side

Because the bottom clamps needed an arc-shaped surface to mount on the pontoons, the manufacturing process was more complicated than expected. The SCU machine shop was not equipped with the proper tools to accomplish this task; therefore the clamps were cut using a water jet cutter. Top and bottom clamps attached to the vertical struts and the platform can be seen in Figure 3.30.

Figure 3.30 UHMW clamps fastened to vertical struts

One issue considered was how the clamps were going to be fastened and how the end caps would be bolted down once the foam filled the pipes. To solve this, panels were designed to be cut out of the pontoons to grant access to bolts and nuts holding together the end caps and the clamps. Location of access panels can be seen in Figure 3.31.

Figure 3.31 Access Panel Location

Knowing the ASV will be deployed in a salt-water environment, the materials selected needed to be marine-grade corrosion resistant. Such materials are stainless steel, marine grade aluminum and plastics. The pontoons are made of PVC, a plastic used in plumbing, and fiberglass. All clamps are cut out of ultra high molecular weight plastic known for its strength and rigidity. The bolts, washers and nuts used to secure the clamps and the end caps were all made out of stainless steel. The vertical supports are made out of honeycomb aluminum plates. The outer aluminum plates are made of anodized aluminum and surround the inner honeycomb structure which is made of non-anodized aluminum. To protect the inner aluminum structure from permeation by salt water, protective adhesive filler was applied to the edges. This adhesive also created a safer surface for handling by covering the sharp edges.

4.0 ELECTRICAL SUBSYSTEMS

4.1 Overview

The electronics on the ASV employed common bus distributed avionics architecture with each subsystem being controlled by its own microcontroller. This approach was used for a couple of reasons. It allows for the construction of abstract pieces that can be added, removed or upgraded depending on how the project evolves before its completion and allows groups in future years to work on one system independently without having to know the specifics of the entire project.

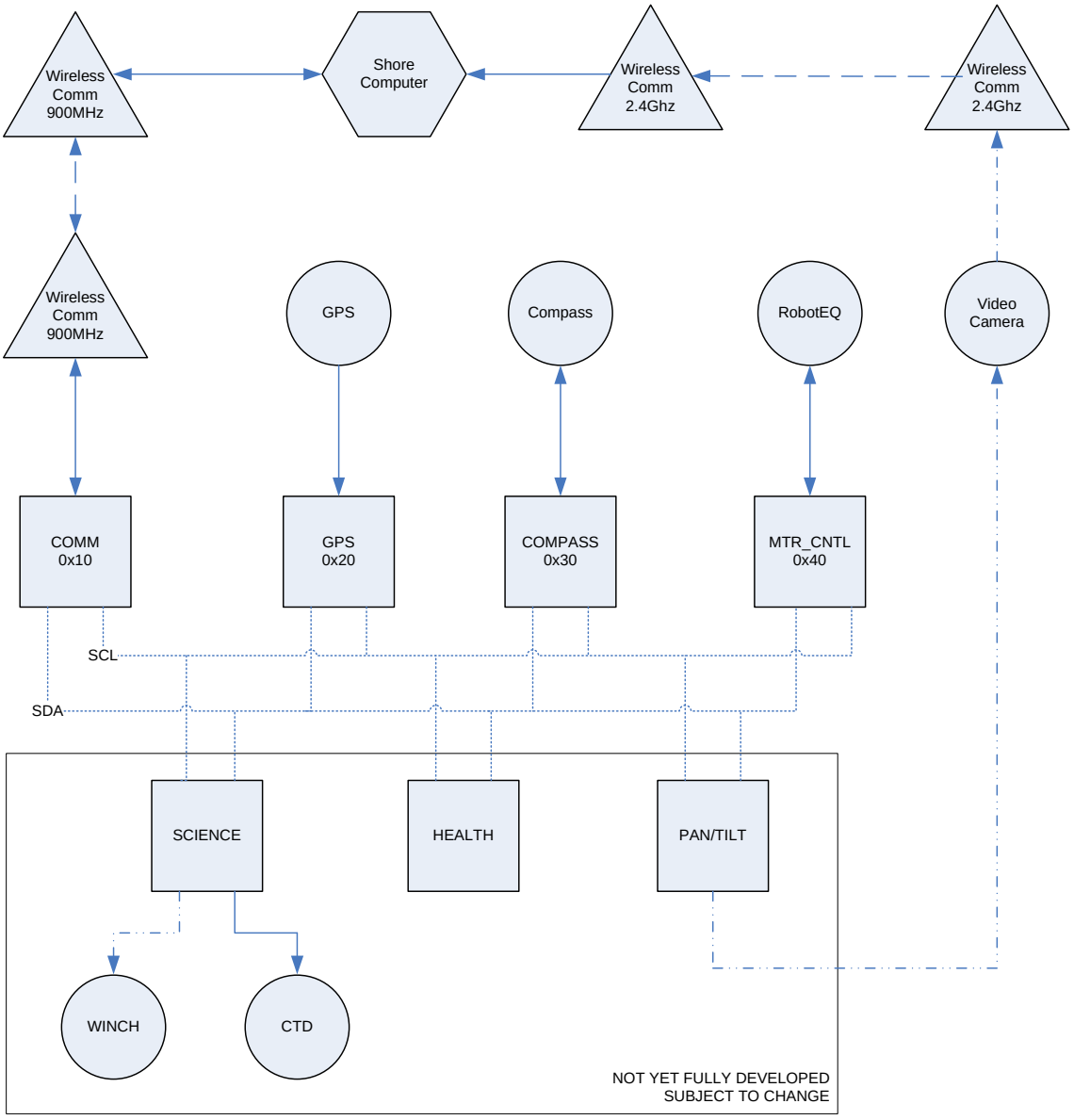


Figure 4.1 ASV Component Block Diagram

A distributed, avionics architecture breaks the components of a system into function specific subsystems. These subsystems each do their own task and work together to create a complete system. Some configurations to connect these systems together are: the star, the single bus or the multiple bus architectures. In a star architecture, one master device connects to each subsystem. This system works well as long as none of the subsystems need information from each other; if this is the case, a method of information forwarding must be developed to handle cross subsystem communications. Devices on a single bus system, such as I^2C, will facilitate a multi-master architecture that allows for cross system communication. For this application, I^2C was chosen over other multi-master setups such as SPI or CAN, because it is simple and does not need additional hardware or modification of existing systems to expand the overall system.

Each of the subsystems in the system block diagram was represented by a square and each contain their own microcontroller. Each of the controllers was connected to the I^2C bus and most have another device connected to its UART.

4.2 Microcontrollers

Each of the subsystems has its own microcontroller. The microcontrollers used are an AVR 8-Bit RISC processor from Atmel. These processors were chosen because of their capabilities and faculty experience and support. We currently are using their ATMega16 and ATMega32 processors but if needed the I^2C could be replaced with a second UART by using the ATMega162. The devices we used feature a UART, I^2C, SPI, JTAG interface, 3 timers, 4 PWM channels, 10-bit A/D converter, 16k of Flash, 0.5k EEPROM, and 1k SRAM on the ATMega16 and twice the Flash, EEPROM and SRAM on the ATMega32. The list of features included in this package, while most likely not all used in any one system, allow for the same processor to be used for any task that may be needed in any particular subsystem.

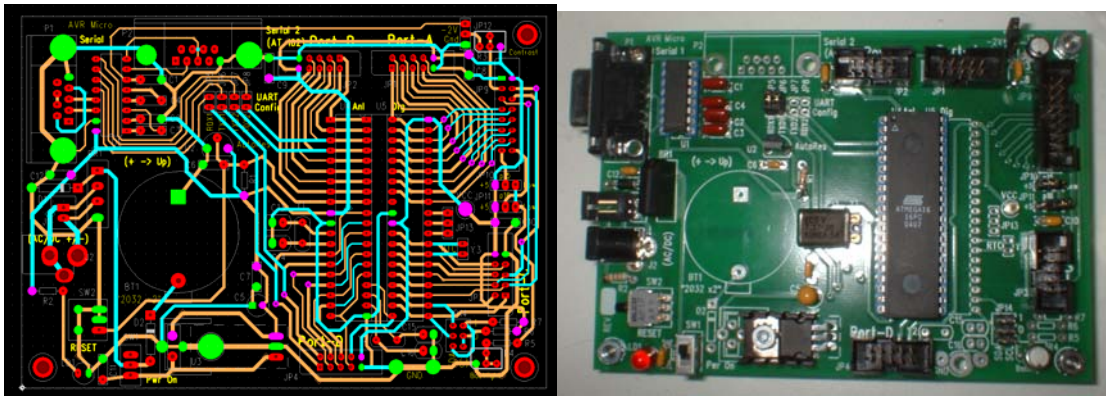


Figure 4.2 (a,b) a) Processor layout in CAD and b) actual board layout

The pc board that we designed and had fabricated will accept any of Atmel's AVR processors in a 40 pin DIP package. Some of the features that we included are separate 10 pin headers, a separate LCD header pined out for HD44780 based LCD modules, I^2C header, very low drop voltage regulator and a lithium backup battery, a real time crystal, and space for a second UART. The schematics and parts list created with Orcad Capture, and layout created with PadsPCB can be found in Appendix N.

The currently used microcontrollers have been loaded with a bootloader using Atmel's STK500 board, a general development board. The specific bootloader that is being used can be found at http://hubbard.engr.scu.edu/embedded/ and is designed to be run at 7.37 Mhz [14]. Using a bootloader allows for in system programming during the development phase but will need to be removed from some of the devices before project completion. Any device that will potentially have data being received by the UART on system startup will need to have the bootloader removed before project completion. If the device receives any data within the first 3 seconds after being powered on, it will stay in the bootloader and never execute the program code.

All of the code for the microcontrollers was written in C and compiled on the AVR-GCC compiler which can also be found at http://hubbard.engr.scu.edu/embedded/. The specific version of AVR-GCC that we used was 20040720 because some of the newest versions had conflicts with our function libraries, AVRlib, also available at http://hubbard.engr.scu.edu/embedded/ . All of the code for the microcontrollers not included in AVRlib can be found in Appendix O.

After writing and compiling our code, the processor's flash was loaded using Atmel's AVR Studio. Version 4 is used for loading the bootloader on the STK500 board and version 3 is used if loading from the bootloader. Both versions of AVR Studio can be found on Atmel's website, http://www.atmel.com [15].

4.3 Navigation

The roles of the navigation controllers were to acquire the boat's current position and heading, and based on this information, control the thrusters to move the boat to a given waypoint. The secondary roles of the navigation controllers were to keep the boat at the last completed waypoint until another is given. This station keeping will keep the boat form drifting while the instruments are deployed and taking measurements.

An initial prototype of this system was constructed using 2 Basic-X stamps, a digital compass from Devantech and a GPS receiver from Garmin. The capabilities of the Basic-X stamps are quite limited and could not completely satisfy the requirements needed for this application. The system was able to parse GPS packets and acquire heading data from the compass but the Basic-X stamps lack hardware I^2C support. Bit banged I^2C seemed to cause problems in a multi master system and the Basic-X solution was replaced with processors from Atmel that support hardware I^2C.

The new navigation controller was broken down into three separate controllers, each with a specific job. One controller is responsible for acquiring a heading from the compass, another is dedicated to receiving and parsing GPS strings and the third is used to control a separate motor driver board which drives the thrusters.

4.3.1 Compass

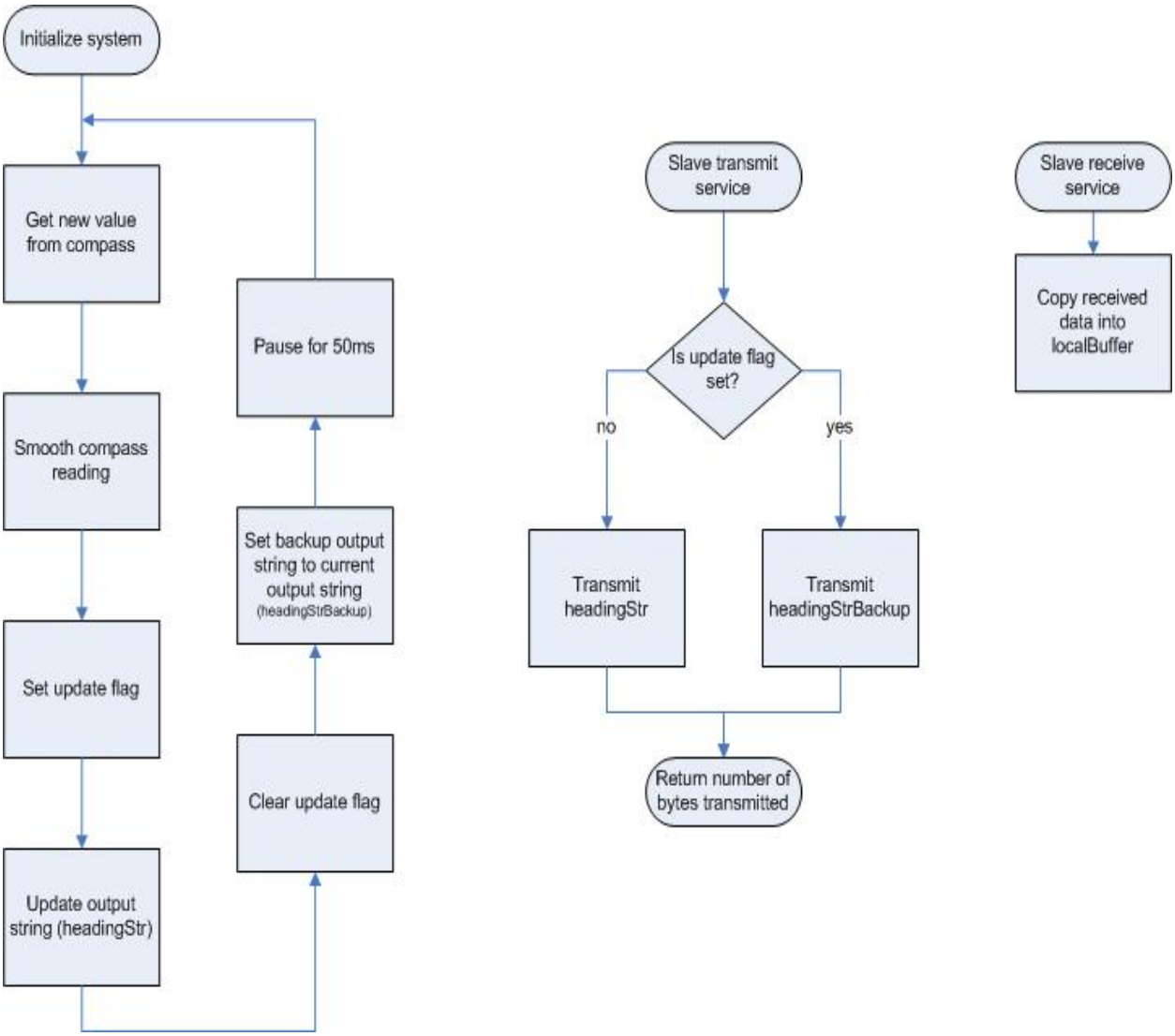


Figure 4.3 Compass Flow Chart

The compass controller is responsible for collecting heading information from the digital compass and filtering the data into a steady heading reading. Currently the digital compass used is a Devantech CMPS03. This is a small inexpensive compass that works well for testing of the smoothing algorithms and simple system testing but it should not be used in the final design. Another much more complicated compass has been purchased but not yet integrated into the system. This is a tilt compensated compass from PNI, model TCM3. This tilt compensated 3 axis compass will give an accurate heading reading when tilted up to 80° and will work in high

latitudes. This is extremely important because as the ASV will list while in open water and will eventually be stationed in a high latitude in Alaska.

The current compass is interfaced to the compass controller through I^2C and complete protocol documentation can be found in Appendix K. The new compass interfaces with RS232 and can not easily be substituted for the Devantech. Although the interface protocol will need to be rewritten for this new device, from a system view point, the change will be transparent and can be done with out modifying other systems.

Currently a simple low pass filter has been implemented in the compass controller's software. The current algorithm has been implemented, but not extensively tested, and may need to be upgraded to a more advanced system such as a kalman filter.

The controller constantly collects and computes the ASV's heading in the background and will act as a slave on the I^2C bus. If another controller on the bus needs heading information it will address the compass controller and it will respond with the current heading reading. To avoid the possibility of sending partially written data, this system double buffers the data. If a request is made while compass data is being written to the output buffer, the write process will be paused and the slave request granted, thus sending data that is only partially complete if a second buffer were not used. If the data is in the process of being updated rather than the output buffer being sent, the backup buffer is sent. After the slave request terminates and the program resumes, it will finish writing to the output buffer and then update the backup buffer.

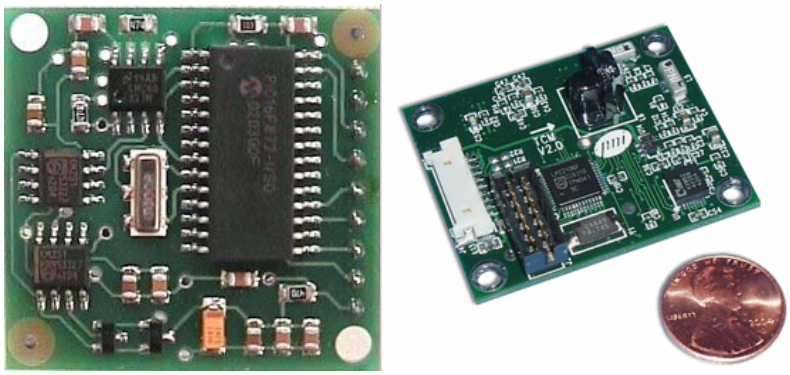


Figure 4.4 (a,b) a) Devantech CMPS03 board and b) PNI TCM3 board[16]

4.3.2 GPS

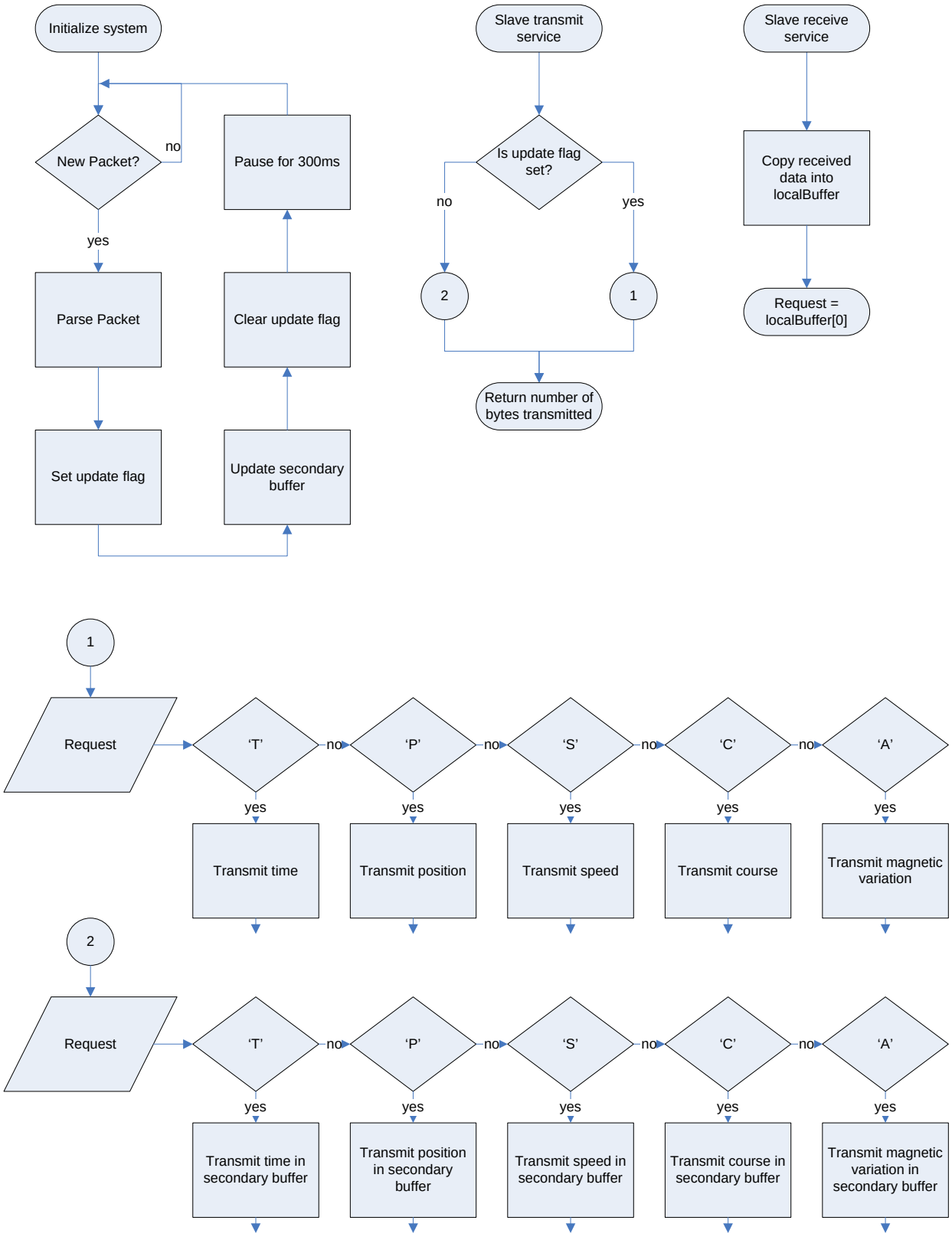


Figure 4.5 GPS Flow Chart

The GPS controller is responsible for interfacing to our GPS receiver and interpreting GPS strings for use in our system. Currently we are using a Garmin model GPS18 receiver. The unit was preconfigured using Garmin's included software to transmit the GPRMC sentence before it was used in the ASV system. Once the receiver had been initially configured, there was no need to reprogram it; thus, no setup commands have been included in this microcontroller's interface instructions.

Figure 4.6 GPS18 with 12-volt cigarette lighter adapter [17]

Once the GPS unit is turned on, it immediately transmits GPS sentences at a rate of 1 per second. The GPS controller then reads these sentences from the UART buffer, parses the information accordingly and saves it for request by other systems. Current attributes available for request are: time of fix, position, speed, course, and magnetic variation. Because the GPS receiver immediately transmits data when powered up, the controller will lock in the bootloader, not allowing the program code to run. To remedy this for bench testing, simply power on the microcontroller at least 3 seconds prior to the GPS receiver or don't connect the GPS receiver until after the bootloader has exited. When the code has been completed for this controller, the device should have the bootloader removed and have the final version of code uploaded using the STK500.

Included in the GPS receiver is a position smoothing algorithm. This feature is selectable and may be turned on or off. Currently the feature is off but tests should be run with it on to determine whether it is adequate enough or if a separate filter needs to be done in the controller's software.

To access the GPS information from the controller, a master on the bus will send a character to the device that indicates what information to send on the next request. Then when the master next sends a receive command, the information sent will be of the type just requested. As with the compass controller, the GPS controller double buffers data that will be transmitted by the slave transmit service to ensure valid data.

4.3.3 Motor Controller

The motor controller is responsible for autonomous navigation and interfacing to the RobotEQ dual motor driver board. The RobotEQ is a 60 amp dual motor driver with many convenient built in features. Some of the more important features for this application are its ability to accept a forward/reverse vector and a desired turning vector for a tank style differential drive system, an automatic acceleration control, a current limiter, a watchdog timer and a RS232 interface.

The RobotEQ can be setup with either its own software independent of the ASV or it can have individual parameters set by the motor controller using the RS232 connection. Not all of the board's options have been included in the motor controller's command list but the ability to turn the watch dog timer on and off, set the current limiter and set the acceleration were included. As with the GPS controller, the bootloader will have to be removed before the controller is integrated into the final system.

The motor controller in addition to interfacing with the RobotEQ also interprets both the user's manual and autonomous drive commands. In manual mode the forwarded lateral and rotational throttle vectors from the communications controller are recognized as manual controls, interpreted and translated, and then passed on to the RobotEQ to drive the ASV. Instructions are given as a percentage of throttle 0%-100% and are handled in real time. Both commands are sent unless the heading control is on. If the heading control is on, the turning vector will be overwritten to follow the heading that the ASV was at when the heading control was engaged. The proportional gain controls how quickly the ASV reacts to track this course is easily changed from shore to simplify initial testing. The unsigned char pGain is initially set to 4 during system initialization, but if increased will slow the ASV's reaction, if decreased, it will increase the reaction. pGain controls the heading control tracking gain and autonomous tracking gain.

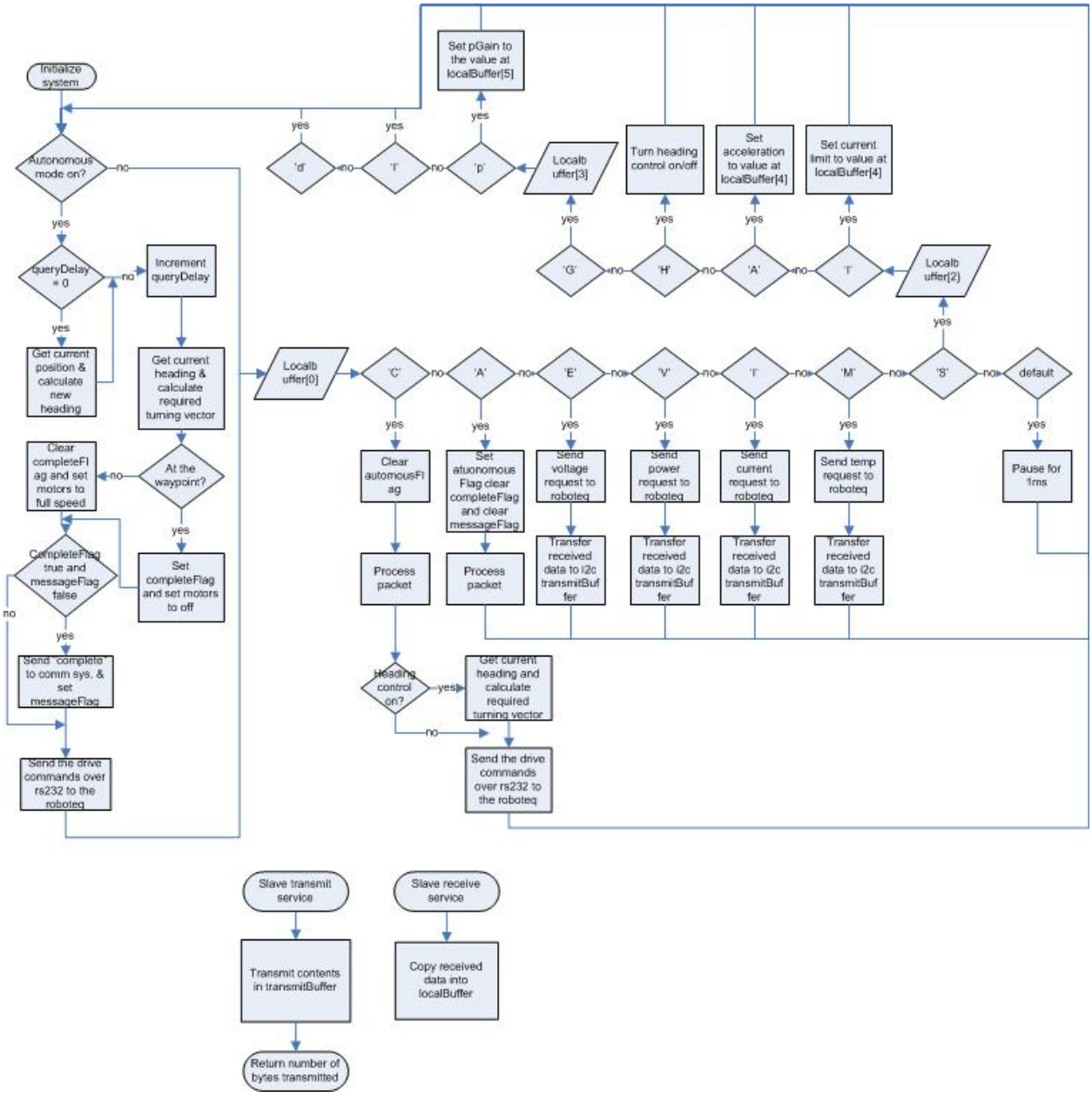


Figure 4.7 Motor Controller Flow Chart

In autonomous mode, latitude and longitude coordinates are forwarded up from the communications controller and based on current location, control the RobotEQ to drive to the desired location. When a new coordinate is received by the motor controller, the autonomous mode is automatically turned on and will stay on until manual mode is entered. When autonomous mode is turned on, the coordinate entered is compared to the current position of the ASV, requested for and received from the GPS controller, and a needed heading is generated. In each loop of the program, the current heading is compared to the desired heading, requested for and received from the compass controller, and a turning vector is generated and sent to the RobotEQ. Once every 256 iterations of the program loop the current position is compared to the coordinate and a new desired heading is generated, this is done to compensate for any drift that may occur while operating.

Currently the system is not set up to store a mission plan on board and relies on receiving one coordinate at a time. Therefore, when the ASV arrives at its destination, it will send back a complete flag to the communications controller. This will signal the shore that another instruction is needed, either a new coordinate or a science task. The unit turns off the motors but is still in autonomous mode, therefore constantly polling current location. If the boat drifts from our coordinate, the motors will be turned back on and a course will be followed to return to the coordinate where the motors will be shut off again.

Figure 4.8 RobotEQ AX3500 [18]

4.4 Science Instrument Controller

This controller will be responsible for controlling the CTD and raising and lowering the winch. Currently the only functionality this controller has is an open loop winch control because no science packages such as the CTD have been available to use. The winch is controlled via an expansion board connected to port b. Port bits 0 and 1 drive the winch up and down respectively. There is a simple watchdog timer attached to these commands that will automatically stop the winch if no update command is received by the time the timer runs out.

The expansion board uses a SN75462 dual peripheral driver to drive the relays that control the winch motor. The SN75462 uses a TTL AND logic gate to activate a transistor used to drive the relays. For this application the logic inputs have been tied together and it is being used as an buffer. There is a three wire interface to the winch. By connecting either the up or down control wire to a common, the winch will be activated respectively. A schematic and board layout can be found in Appendix N.

4.5 Wireless Communication

There are two types of wireless communication systems between the ASV and shore. One is a bi-directional data modem from Maxstream operating at 900 Mhz and the other is 2.4Ghz Analog video system from www.wirelessvideocameras.com for live video.

4.5.1 Communications Controller

The data modem is used to facilitate communications between the shore and the ASV. One data modem is connected to the shore computer and the other to the communications controller, which serves as the spokesperson for the ASV. Any communication to or from shore comes through this controller. Its primary functions are to receive incoming packets from either the shore or other controllers. If the packet is from shore is will validate its check sum, acknowledge the received packet as valid or not, and forward the packet to the appropriate controller. If the packet is received from another controller over the I^2C bus, it will forward it to shore.

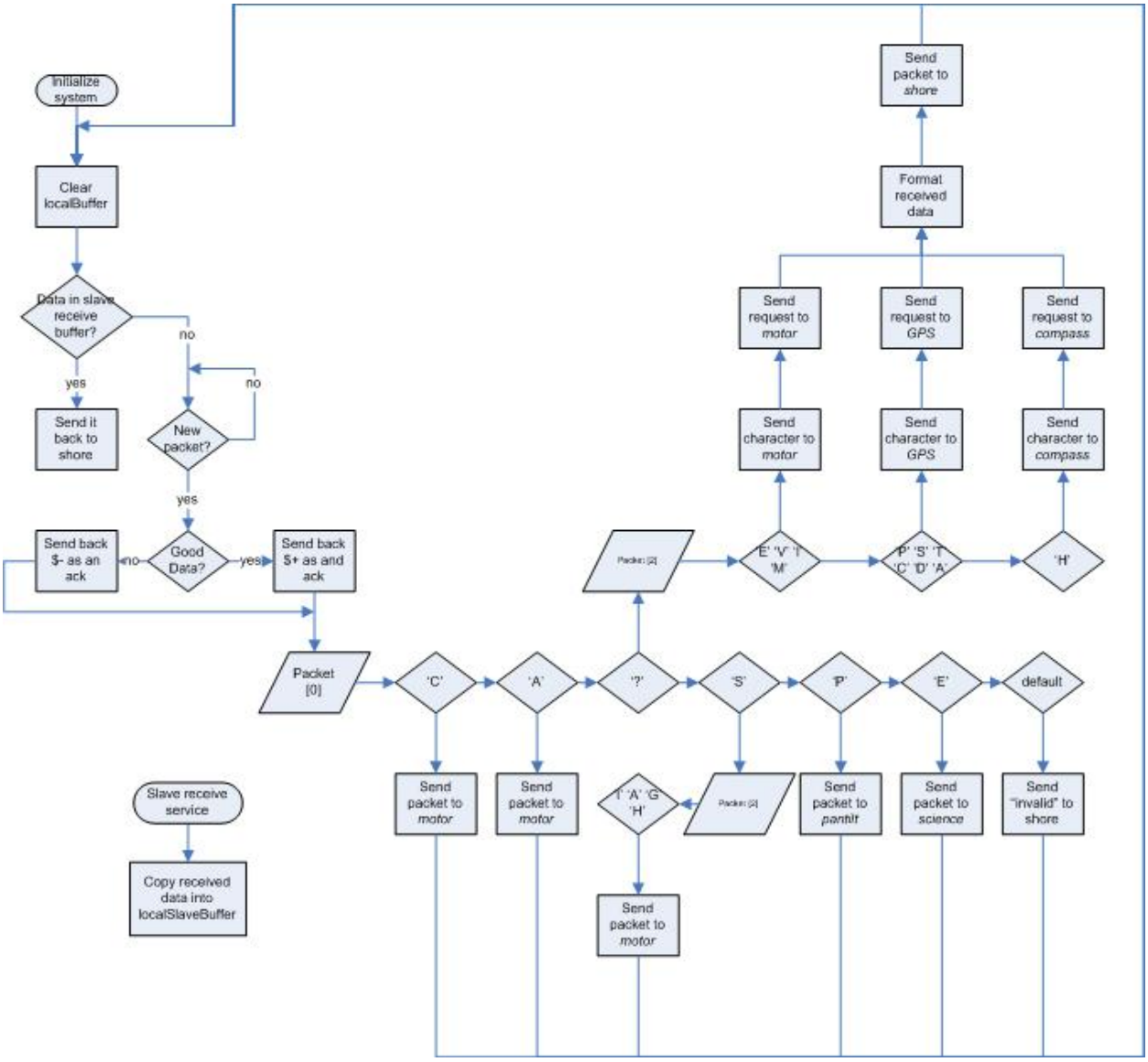


Figure 4.9 Communications Controller Flow Chart

Figure 4.10: 9XTend-PKG-R™ 900 MHz RS-232/RS-485 RF Modem [19].

4.5.2 Video

The ASV is equipped with a wireless video camera from www.wirelessvideocameras.com for visual feedback when manually driving the ASV. This camera is rated for a range of 2 miles and was built for this project by wirelessvideocameras.com. It is completely plug and play and simply needs to be installed on the ASV. The system is a one way system utilizing a transmitter and receiver. The transmitter onboard the ASV is connected to a video camera. The receiver on shore is connected to a video capture card that interfaces with the shore computer.

4.5.3 Camera Pan

This controller is responsible for driving the DC motor attached to the camera mast. Currently the only functionality this controller has is an open loop control. The motor is controlled via an expansion board connected to port b. Port bits 0 and 1 control the motor with bit 0 selecting the winding polarity and bit 1 turning the motor on or off. There is a simple watchdog timer attached to these commands that will automatically stop the motor if no update command is received by the time the timer runs out.

The expansion board uses a SN75462 dual peripheral driver to drive the relays that control the DC motor. The SN75462 uses a TTL AND logic gate to activate a transistor used to drive the

relays. For this application the logic inputs have been tied together and it is being used as an buffer. A schematic and board layout can be found in Appendix N.

4.6 Shore to ASV Protocol

The protocol used to communicate between the shore and the ASV was modeled after GPS NMEA sentences. The start byte is '$' followed by the instruction type, then the instruction parameters separated by commas, then a '*' as a stop byte followed by a generated XOR checksum and a carriage return and a line feed as a packet delimiter. A full set of these commands is included in Appendix O.

This protocol is extremely simple with one command doing one task. The types of commands include manual drive commands, autonomous coordinate uploading, query commands, parameter setting commands and instrument actuation commands.

Every command received by the ASV is acknowledged with a validation of the checksum. If the checksum matches '$+\r\n' is returned, if it does not, '$-\r\n' is returned. Although there is some form of error checking, there currently isn't any form of error correction such as CRC-16. If a packet is received with a bad checksum, currently the system will still try to process the request. This was done for development reasons but should be changed before system implementation by adding a simple condition statement. It should be noted that if the corrupted bits are in the packet start or stop delimiters, the packet will never be found and no acknowledgement sent.

For most commands only an acknowledgement will be sent. Only the query command will have a returned packet. After the query acknowledgement, a properly formatted packet will follow it containing a header, command id, query type, query parameters, end byte, checksum and delimiter.

Currently a format has not been decided to handle invalid commands or waypoint completion flags, simply "invalid\r\n" and "complete\r\n" are transmitted.

4.7 Electrical Hardware

To house the processor boards for the navigation and science instrument controllers, wafers were water jetted and inserted into clear plastic boats. The boards were bolted onto the wafers and the box was sealed. The hardware can be seen in Figure 4.11.

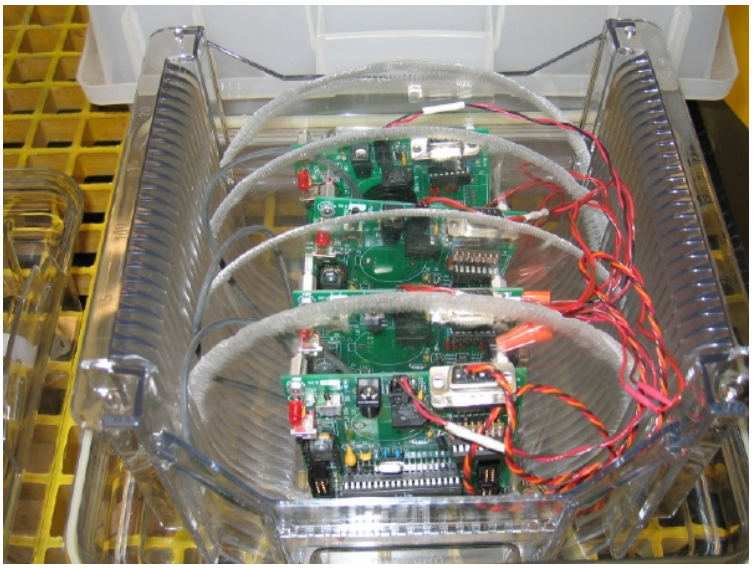


Figure 4.11 Electrical Hardware

5.0 COMPUTER SUBSYSTEMS

The software subsystem is responsible for communicating commands to the robot, and receiving status information and data from the robot to present to the user.

5.1 Software System Requirements

The primary requirements for this system are real time performance to navigate the robot, and to guarantee timely and accurate presentation of the status information to the user, received from the robot:

- Data distribution and availability requirement
- Real time system performance

In the process of working on this project we also considered it suitable to add the following two items to the list of the requirements:

- Standard and high performance components
- Enhanced debugging

In the following sections each of the above requirements will be discussed in detail.

5.1.1 Real Time System Performance

Although the ASV can operate autonomous based on an uploaded route, typical of all remotely controlled analytical and research devices it is also required to be controlled manually. The manual control has to be in real time were the user navigation and other commands are both transmitted to the robot, and are received by the robot in real time. Furthermore, the system is required to receive data from the ASV in real time and be able to make it available to the user as soon as it arrives.

Based on an earlier discussion, we define the real time tolerance for this system to be 300ms or better – In other words as soon as the user issues a command, we expect the robot to receive and react to the command within 300ms. Similarly, when the robot transmits data back to the user, the data should be available for viewing within 300ms or better – The latter requirement does not necessarily mean that the user should see the data as it arrives. Rather, it means the user may be viewing some historic data and currently not interested in the newly generated data. He/she regardless, has the option to view the newly arrived data within 300ms of arrival.

For this project, we have therefore focused on performance to make sure that the system will be able to respond and act upon commands as they are issued. Short of a custom developed real time Micro kernel, we made this system as responsive to real time commands and events as possible by giving the delivery of the command to the serial port for transmission, the highest priority. We accomplished this task by having active agents to constantly monitor the data ports where commands and data were exchanged with the robot.

To benchmark the system architecture performance for what we had in mind, we wrote a simple Java application to connect to another application via TCP/IP, and transmit the current system time. The receiving application would then acknowledge the receipt of the data and send back the received system time, as well as the original transmitted system time and would then write the data into a file.

Although these simple applications were written to work on two remote systems, for accurate timing purposes, they were running on the same computer to guarantee the same system time. Running on the same computer enabled us not to have to synchronize the system time between two different computers. We ran the data 10,000 times while sleeping between 1 and 10 seconds randomly.

In another experiment, we wrote a JSP servlet, running under Tomcat, our target webserver. Upon invocation, this servlet would send the system time as a response. Using a browser, we wrote a javascript that similar to the above application, would randomly sleep between 1 and 10 seconds and then send the current system time to the webserver as a parameter to the JSP servlet. We also ran this test for 10,000 requests.

These experiments ran for nearly 15 hours and offered similar if not identical time delays from transmission to acknowledgement.

After running these in house developed benchmarks which did in fact closely resemble the target design system, and based on the Computer/Internet/Online industry growth, standards, and

requirements, we found out that there was no overhead using an HTTP browser client instead of a TCP/IP based java application.

It is our understanding that browsers such as Microsoft IE have been in development for over a decade and their code is optimized to exchange data over HTTP over TCP/IP. The HTTP optimization over TCP/IP was an industry focus for a number of years, offering efficient and optimized performance.

In an actual test observation at the SCU Engineering lab (e.g. see Figure 5.1), an oscilloscope was attached to the Joystick on one side and to the robot on the other. We used a 700MHz PIII with 512MB RAM computer, running Tomcat under Windows 2000 as our server. This server was connected to the SCU ENGR wireless 802.11b domain via an 802.11b wireless adapter. Connecting to the server, the client system (e.g. the user) was a Pentium Mobile 1.6GHz with 512MB of RAM, running Windows XP Professional. Connecting to the web, this system used the built in Wireless 802.11a/b/g Mobile 1400 Dual Band WLAN Mini-PCI card. The established connection was 802.11b both because of the SCU ENGR wireless router which is an 802.11b, and also because the server was equipped with only an 802.11b wireless card. With all the computers connected wirelessly online (e.g. internet and inside the firewall), we observed that the joystick commands were consistently received under 90 milliseconds. The following diagram (e.g. figure 5.1) is a screen shot of the oscilloscope, displaying the above test.

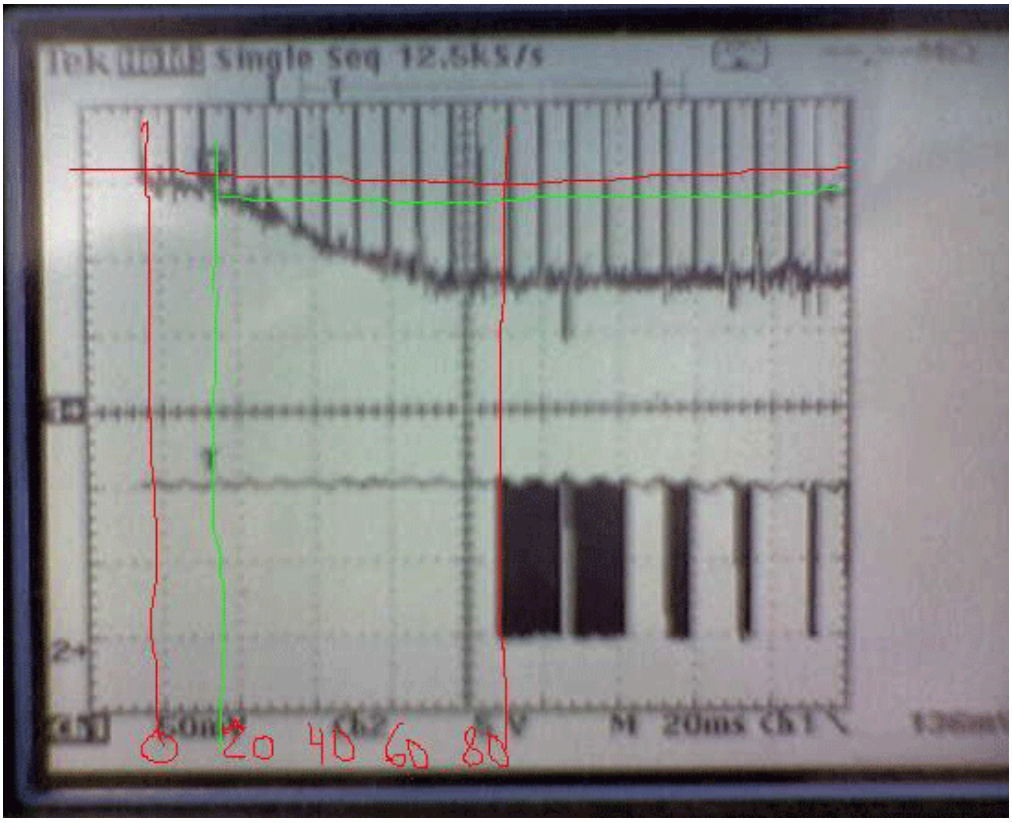


Figure 5.1 Dual Trace Oscilloscope Screen Capture

5.1.2 Data Distribution and Availability Requirement

Another one of the requirements for this application was Data Availability and Data Distribution to both different people and other applications.

We interpreted this requirement to mean that the data needs to be centralized to avoid ad hoc data management and at the same time made available programmatically to all authorized users and applications.

Although storing the data in a local file, using a local file system is an option, synchronizing access to the data and authorizing data modifications would all have to be implemented.

Using a database, particularly a relational database, with available connecting libraries such as JDBC was the obvious solution.

5.1.3 Standard and High Performance Components

To focus our efforts on data collection, and data presentation and to allow for ubiquitous communication with the robot, and easy future extensions, and to have minimal impact on the budget, we also imposed using widely used open source software as an extended requirement.

For instance, a highly popular “user browser paradigm” is running Internet Explorer under Windows. The server is required to run under Linux while the database can reside on yet another machine, also running under any of the known operating systems today such as Windows, Linux, Solaris, or Macs.

5.1.4 Enhanced Debugging

Due to unavailability of the robot at all time, either due to logistics of having a robot/submarine/boat for every developer wherever they are, or simply because the robot may be under additional development, we enhanced the system with a debugging, no-robot feature that will simulate the interaction with the robot.

Instead of an actual robot, a simple and small software agent reads and writes to the serial port, displaying the commands received on that port, or transmitting data on that same serial port, as if it was sent by the robot, which in turn is picked up and displayed on the pilot’s console.

5.2 System Requirements and the High Level Architecture

As described above, the primary objective of the software system was to deliver commands to the robot accurately and in real time, and to receive the collected data and status from the robot, in real time and accurately.

To satisfy these requirements, we resorted to a simple and common, yet dynamic architecture. The software subsystem is comprised of a number of widely used software components depicted in figure 5.2.

User←→Browser←HTTP→WebServer←HTTP→DataCollector←Wireless/Wired→Robot

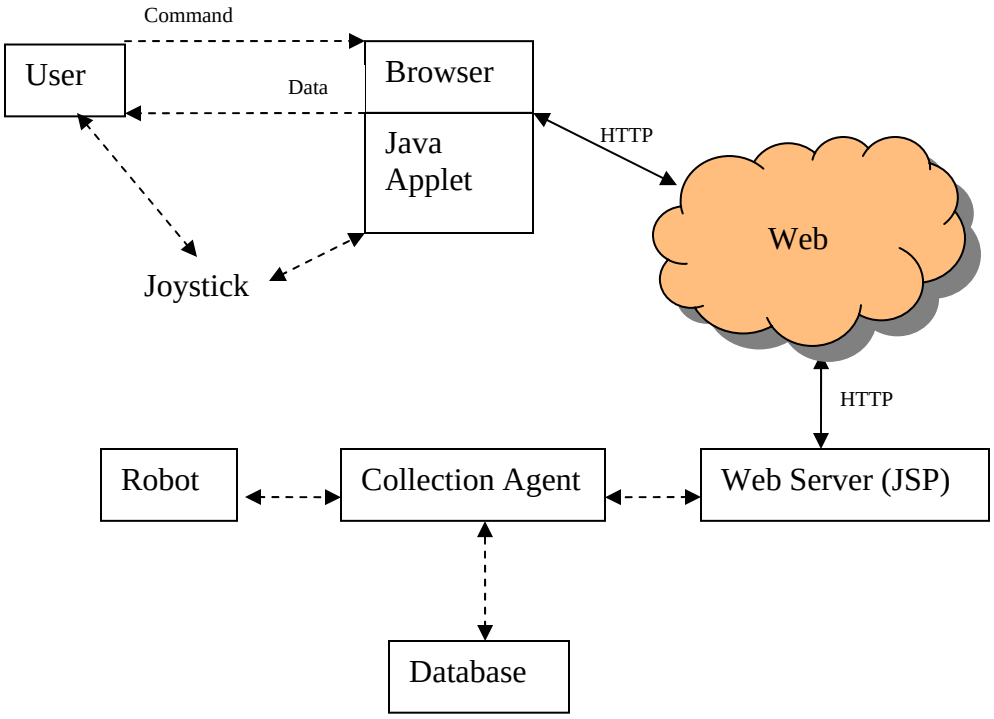


Figure 5.2 Software Architecture Overview

We chose this architecture over any other architecture for a number of reasons such as:

1. **Widely known Paradigm** – The internet is the best known client application paradigm. With nearly 2.7billion and 2.3 billion daily clicks, Yahoo and Google are the proofs for such a claim. No other paradigm, no other application can claim such a astonishing usage. With 53,748, 603 active domains and another 446,308 on-hold, as of today, the HTTP style of applications connectivity is definitely the highest connection model, over which upwards of 840, 000, 000 users communicate and share information and click over 8 billion times a day (e.g. according to GlobalResearch Searching agency). No other application can make such a claim. Table 5.1 below displays some of such statistics.

Domain Counts						
			Daily Changes (last 24hrs)			
Active	Deleted	On-Hold	New	Deleted	Transfered	TLD
38,631,970	26,380,428	346,746	354,909	285,583	51,890	.COM
5,860,473	4,279,319	63,376	15,465	7,462	5,995	.NET
3,632,614	2,371,702	33,401	14,873	3,368	3,213	.ORG
3,591,415	1,082,267	1,274	3,812	909	1,893	.INFO
1,174,507	538,036	1,021	6,520	647	1,000	.BIZ
857,624	688,552	490	709	629	1,129	.US
53,748,603	35,193,089	446,308	396,288	298,598	65,120	Total
						Last Updated 6/15/2005

Table 5.1 Some Internet Domains and Web Usage Statistics

2. **Outdated Client / Server model** – In the previous popular Client and Server, a connection would persist between the user and the provider. Furthermore, the connection between the two was typically TCP/IP. With this younger, yet more widely and frequently user paradigm, namely the internet, connections never persists. The session will be remembered and reconnected to exchange further data, but the connection itself is established and dropped for every transmission.
3. **Request brokers** – Although request brokers have been around for more than a decade and have given birth to technologies and industry standards such as CORBA, they too have retooled and instead of TCP/IP, they use HTTP to communicate. A great deal of request brokering services are also passed on to the Webservers.
4. **Documentation Availability** – The systems we used such as Tomcat and MYSQL are all documented in detail and professionally, from how to install and use them to how programmatically connect to them. Availability of such documents promotes faster and more optimal development: The API is described and one will not need to reverse engineer the code to figure out what it does.
5. **Global Support** – Since both the application executable and the source code are widely available, the number of people who know how to interact with these components, or are able to actually read the code and other documents to come up to speed and support it, greatly increases, improving the quality and availability of any level of support.

6. **Simple Installation, Trend setting** – Given their widespread use and their install base, these applications, these services, and these channels are under a great deal of scrutiny. Given the "contributing" nature of Open Source, they are also constantly under development and improvement. For instance it takes minimal effort to install MySQL or Tomcat on literally any system. Also because of the great number of install base, their popularity, they set trends for what is and what is to come.
7. **Security and Privacy** – Given their diverse use, from people who use them to the environments within which they run, security has always been a developing and concerning issue. From Spams and Ad ware to identity thefts, Security has been a challenging and developing issue, particularly because it is expected to assist not just one, but all applications, and not just in one environment, but all environments. The proposed architecture facilitates implementing different security features, using already built in functionality.
8. **Component Isolation (Encapsulation)** – Again, due to their widespread use and diverse operating environments, web components cannot assume anything about their environment or available other components – They need to carry with them all they need in order to properly and effectively function. Such self containment is what we consider Encapsulation.
9. **Ubiquity and component availability** – Because they are open source, it is easy to find these components online, anywhere and from a number of sources. Because the system is web based, it can be monitored and controlled from anywhere.

The user issues a command, moves the robot, or requests for data. The interaction is via the keyboard and the joystick. In a typical scenario, here is what takes place:

1. The user connects to the system via the browser.
2. The user issues a command, moves the robot, or requests for data. The interaction is via the keyboard and the joystick.
3. The command is communicated to the Web Server as a JSP servlet request.
4. The web server invokes the appropriate JSP servlet and executes the command.
5. The command execution is generally means connecting to the database and looking up or writing data.

6. A Java agent that monitors the port, and the database, detects arrival of the new data and writes it to the port.
7. The port driver transmits the information (wireless) and the robot receives the command.
8. The command is recognized and executed by the robot, and a resulting data, if needed, is sent back.
9. The port agent receives the data and writes it to the database.
10. If the user updates the screen, the new data, if qualified, will be sent to the browser and displayed.

The GUI is responsible for interpreting user actions into specific commands and for sending these commands to the robot. In a text/command driven user interface, the software needs to validate commands to make sure that they were accurately entered. A GUI, not only greatly simplifies interactivity, it also eliminates the need for validating the commands because commands are generated by the computer based on the user actions.

All the commands that the user sends to and the data received from the robot are stored in a relational database for fast and easy reporting – A researcher for instance may want to look at the set of data collected at a certain time of day, or look for abnormalities in the data, or data values in a particular range. Using a relational database with standard SQL query capabilities, will greatly facilitate such investigations. Using a relational database also enables the researchers to easily archive the data they don't need, purge it, or modify the data that was incorrectly collected due to equipment malfunctions (e.g. not properly calibrated).

5.2.1 Real Time Design Overview

To accomplish the real time data delivery we used data collection agents. The data collection agent is an interrupt driven software daemon. Similar to all other software daemons, it goes to sleep when there is nothing to do, and is awakened when data arrives at the port and an event is generated. This agent has no user interface and its job is to basically read the data from the serial port sent by the robot, and write the user commands back to that port for the robot to execute.

The agent writes the data to the database after it has sent it across the serial port to the robot. In other words, as soon as a command is issued, it is written to the port, heading for the robot and then recorded for future access.

The use of MYSQL database, specifically optimized to run in a minimal PC environment and highly suitable for this application that the volume of data exchanged does not exceed a few kilo bytes at a time, and substantially less than mega bytes throughout the day. The database similar to all other system resources, although multi-user, is used by only one user – Unless the database is being queried by another user, there is generally one connection open to the database. In our implementation, we keep the database connection open for as long as the application is running. We also never reset the cursor, the place holder. Writing data of this size, to this repository is similar to writing a block of data to the file system. In other words the use of the database does not tax the application performance.

5.2.2 Data Availability and Distribution Design Overview

We used a typical Relational Database, with SQL reporting and a query support engine not only to satisfy portions of the real time data access, but also to make data available. It is the database's job to facilitate and to improve such data management tasks as data security, fast optimal access to the requested data, and even proper selective or incremental back up for fast recovery.

The database is therefore responsible for locking individual records when records are accessed for writing or modifications. Furthermore, the database is also responsible for authorizing and authenticating users, when they connect to the database to make sure they are who they are, and also to make sure they have proper access rights to read, write, or modify the recorded data.

The database is also responsible for managing connections to the data. Using JDBC to connect to the database, any user and any application can simultaneously connect to the database, locally or from their remote locations, to access the data.

In this design, there is no connection between an application and the agents responsible for reading data from or writing data to the ports. The port data is always written to the database and

the user application retrieves that data by going to the database and interacting with the database and never with the agent.

For instance in our specific application, the browser sends a request to the web server asking to see a specific range of data. The server invokes the appropriate JSP Servlet to connect to the database and to read the required data. The server then sends the retrieved data to the browser to be displayed.

The port agents that are responsible for reading robot status from the port, and writing user commands to the port, are always running. They are oblivious to any user requests or even that a user exists. These agents write the collected data to the database and have no interaction or connection with any user applications.

5.2.3 Standard High Performance Component Design Overview

The project currently runs under Windows. Because it is written in Java/JSP, and because it uses JDBC to connect to the database, it therefore does not need to be Homogenous, nor need all components to run under the same system. From the code that runs in the browser, to the Web Server and the JSP/Servlet, to the database, they can all run under different computers, with different operating systems.

The commands are issued to the robot using the services of a browser making an HTTP request, and the robot response, is also displayed in a browser. Browsers are freely available on literally all computers today.

The commands are honored and responded to using the services of a web server with JSP servlet support. We used Apache with Tomcat JSP/Servlet support from the open source which runs under Linux, Windows and Macs.

Finally, as mentioned above, the data is written to a typical relational database. We used MySQL, also open source and the most widely used relational database today, at no cost.

5.3 Protocol

The data is repeatedly read from a serial port, specified in a properties file, with the following attributes:

- Baud Rate: 9600
- Handshaking: NO
- Parity: NO
- #of Bits: 8Bits
- 1 stop bit.

Every command to the robot is sent as simple text with an easily calculated checksum. All the data from the robot to the data collector is also sent using ASCII text.

The message body is between $ … and … * followed by a checksum, carriage return, and new line. Commas are used as delimiters and are calculated towards the Checksum. Fields that take one of several values are identified in brackets **[…]** in table 5.2 below.

Command	Abbreviation	Arguments	Example
Update	?	$?,[EVIMlLTH]*CS\r\n	$?,H*CS\r\n
Manual Drive	C	$C,X,Y*CS\r\n	$C,127,-127*CS\r\n
Autonomous	A	$A,Lg,Lt*CS\r\n	$A,37231571,121950088*CS\r\n
Camera Pan	P	$P,[-1,0,+1]*CS/r/n	$P,-1*CS\r\n

Table 5.2 Current List of Supported Commands

Table 5.3 describes the abbreviations and characters used when issuing the above commands to the robot:

$	Start of the message character
E	Battery Voltage
V	Power Applied to Motors
I	Current Consumed by Motors
M	RobotEQ Temp
L	Latitude

l	Longitude
T	Time
H	Heading
*	End of the message character
CS	Checksum XOR of all the previous bytes, not including the comas.
#	Number of bytes sent <one byte long> including the comas, $?, *, and checksum itself.
X	The target movement in the X-axis direction (on the scale of -127, 127)
Y	The target movement in the X-axis direction (on the scale of -127, 127)
Lg	The target longitude coordinate for the robot
Lt	The target latitude coordinate for the robot
-1	Move Left
0	Stop
+1	Move Right

Table 5.3 Current List of Command Parameters

5.4 Software Components

From the Computer Engineering perspective, it's worthwhile to examine the procedure in terms of the steps that the user request has to go through until it's handled by the computer hardware.

The client application, rendering the robot status as well as providing the GUI for controlling the robot, is an applet running within common Web Browser. Once the browser is up, the user enters the server IP address and will be presented with the ASV data and control panels.

The software subsystem is comprised of a number of components:

1. Browser
2. Applet responsible for handling keyboard and joystick events
3. Keyboard and Joystick Device Drivers
4. The Web Server responsible for honoring web requests
5. The Servlet/s responsible for sending and receiving information to the database
6. The SQL database for storing all the pertinent robot data and commands

7. Port agents responsible for receiving and sending information between the port on the other end of which the robot resides, and the database on the host machine.

Figure 5.3 is an actual screen capture of what the user sees using Microsoft IE. The status data as well as the collected data is displayed in the scroll box on the left, while the user commands are entered in the text box on the right.

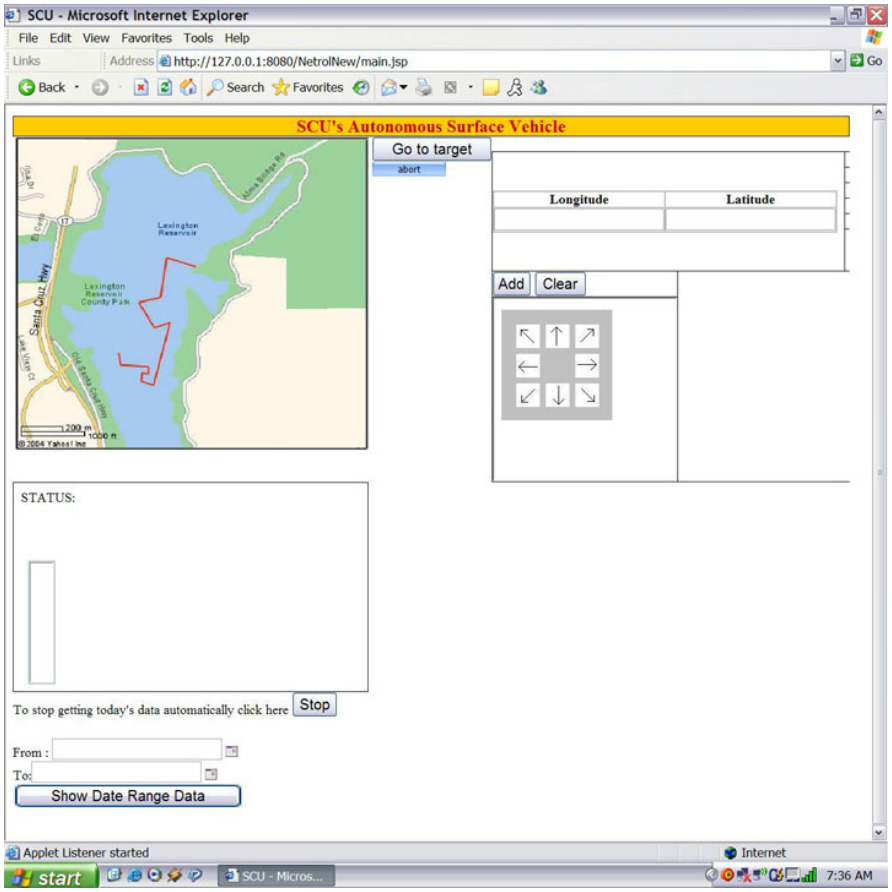


Figure 5.3 Sample Browser Screen

5.4.1 Browser

For this application a common, widely available browser is used. Both IE and Mozzila were tested successfully under Windows and Linux.

A browser was used primarily to eliminate the need for any custom rendering engine, offering platform independence, and promoting communications via the most ubiquitous paradigm over the most common protocol known today, namely the Internet and HTTP over TCP/IP

respectively. With browsers available on practically every computer nowadays, a rendering engine neither needs to be ported from one computer to another (e.g. from a PC platform, to a Macintosh, or a Solaris/Linux computer) nor does it need to be installed – It is already available on the computer. Further more, browsers are the most mature programs available today, with native support for graphics, available tools for custom page designs, and above all, support for extensions such as Plugins and Applets.

A plugin is a software extension written to enhance the features of a browser for a specific application or to make it perform a custom task. Plugins are typically in binary format and are therefore platform dependent. Similar to plugins, applets are downloaded from a server to perform a specific task on the local computer. However, unlike the plugins, applets are written in JavaScript or other languages that the browser itself recognizes and can execute and are therefore platform independent.

The support for both plugins and applets are already available in browsers so in the future, if we decide to display an animated boat to render the location of the boat, such animation can be done with relative ease and no software upgrade for the users will be required. The next time they access the server, they will be interacting with the new animated version.

5.4.2 Applet

An applet, downloaded from the server upon connection to the web server, was used for ASV to enhance the services of a browser.

Although browsers have a great list of features, already configured in, this application needed support for Joystick events for guiding the robot. Furthermore, the Keyboard events are generally configured for entering text, or, in case of using the arrow keys, to scroll the screen up and down, or to the left and right.

A different configuration was needed to

- Understand and handle JoyStick events

- To handle the Keyboard events differently and similar to a joystick, treat them as navigational commands and not for scrolling the screen.

A Java applet was developed to do exactly that. The greatest challenge was because the applet needed to intercept the local device events, it was in violation of the inherent security features of a browser: The browser was not allowing for the applet to eavesdrop of these events to handle them. This is in fact a common problem: Applets are always downloaded to the browser from a website and a web server. An applet, can then be the agent for that site. In most cases, applets are safe and perform the function they are designed to do such as graphical renditions, or access to a remote database for dynamic pages, etc. Unfortunately, applets can also be harmful either as spyware and reporting local computer statistics to a remote computer, or, a virus and causing irreparable damage, even a Trojan horse for releasing access to specific ports for a later attack. From the early on, all loaded browser components, plugins and applets as well as JavaScripts, were restricted from having access to local computer resources. In other words, an applet either has all the information it needs to perform its task, or if it needs local information, it needs to be a privileged (e.g. Signed) application to be able to even read a local file.

In this design we encountered this difficulty and consistent with all other applet designs, we solved the problem by having the applet to self sign a certificate with the local system to have the applet sign in.

In the future, the applet can interact with the user in a dialog box, asking the user if the applet is allowed to access the local resource, namely the Joystick buffer and events. If the user clicks on OK, then the applet will be allowed to inspect the Joystick events, otherwise, the applet will be restricted from accessing the joystick and any other events generated on that system.

5.4.3 Keyboard and Joystick Device Drivers

A standard Keyboard driver was used to communicate keyboard strokes to the applet, which runs in the browser. These events had to be intercepted to prevent the browser from handling them.

Similarly, the joystick driver is a complex driver and handles all the Joystick events. However, a java layer had to be written to connect to the DLL not only as a software liaison to the applet, but also to support some of the requirements such as the range of numbers (e.g. joystick positions) that were required for this project.

Additionally, special attention had to be paid to when the joystick is an extreme limit (e.g. All the way to the left, or all the way to the top) and at the end of a range of figure, and how the applet was to handle it.

We developed a Java layer that intercepts all joystick events. This is the layer that the applet knows how to connect to and receive the joystick events.

5.4.4 Web Server

When a request arrives from the browser, a web server is needed to honor the request. For this project Apache Web Server was used.

Aside from high availability for installation on any system, Apache is the most widely used web server in the world. Although a great number of features of this server were not needed for this application, they did not increase the footprint of the application and did not affect the resource requirements. In other words, only features that are needed for this application which is basic web servicing and servlet connectivity were used. Another reason to use Apache was because it is the most mature web server that is available and the easiest to install. It is also available for free download from any open source site with no constraints, commercial quality features and services, global and cost free support through different forums, BLOGS, and a variety of distributions lists made this server the leading candidate for this project.

5.4.5 Servlet/JSP

Servlets are server side extensions – A typical web server knows how to execute static commands by just looking up a file and sending it across the web to the requesting browser, or, executes local programs and transmits the results of the execution of these programs back to the browser. A highly popular extension to Web Servers is CGI (Common Gateway Interface) which

is still in wide use and still suffers performance issues and scalability. We used Servlet extensions, a more modern variation of the CGI which not only solves the basic CGI problems, it also offers a great degree of flexibility and freedom to generate dynamic pages, the back bone of the ASV software system.

The servlets in this application are all written in JSP, Java Server Pages. When a request arrives, the appropriate JSP is invoked. Most requests from the user are commands to direct the robot, or, a request to refresh the collected data.

The request for collected data is resolved locally: The JSP connects to the database and fetches the requested range of data, formats it, and sends it to the user.

The request for sending a command arrives at the server and the responsible JSP, first writes the command to the serial port using the Java Serial Port API, as well as writing that same data to the database for future references.

Also using the Java Serial Port API, the agent is listening to the port, awaiting any data from the robot. As soon the data is arrived, it is recorded to the SQL database.

5.4.6 SQL Database

This application uses MySQL, currently the most widely used database in the world. Although some of the features of MySQL, particularly transaction handling, are lagging behind other databases such as Oracle or SQL Server, this application did not need those features anyway.

Aside from the price (e.g. MySQL is open source and free), global support (e.g. a great number of BLOGS, distribution lists, discussion groups, etc.), MySQL is extremely easy to install, and similar to the web server and the servlet engine, available on a large number of platforms from Windows and MACs, to Solaris and Linux.

The database is data repository to store all the commands that are issued to the robot, as well as recording all the collected data that is received from the robot.

The intention behind using a SQL DBMS was reporting and selective data capability, which we thought was needed for this project. Since all the data is stored in this relational database, the user can directly access the database at a later time to query specific range of data, specific range of values, or specific days and times, etc.

A side affect of using the database is data availability. Because the data is stored in a relational database, any application can connect to it using JDBC, and after authorization, the application has access to the data. Given that databases handle multiple user access to the data at the same time, the data integrity remains intact and guarded by the database itself and we did not have to provision for that.

5.4.7 Serial Port Listeners

A listening agent, using Java Serial port API, is listening to the serial port at all time, awaiting data from the robot. As soon as the data is detected, it is written to the database.

For writing the data to the serial port, the JSP code utilizes the Java Serial port API and writes to the serial port directly.

As soon as a requests arrives from the user, the Web Server receives it and invokes the appropriate JSP. The JSP immediately writes the received data to the serial port and then writes that same data to the database.

The responsibility to parse and scan the received commands from the user, or the received data from the robot, is left to the robot and the user respectively. In other words these agents make no effort to alter these commands and they send them across, as they arrive**.**

5.5 Results

At the very beginning, we were greatly concerned if our design would meet the real time performance requirement for this system. We were planning to use a full fledge relational

database and a webserver, none of which are known to be lightweight or have a small footprint in either the memory or the resources they use.

Early experimentation with a standalone system running a minimal benchmarking test program under Tomcat and MySQL performed well beyond our expectations, encouraging the design and the architecture.

Although the discussed response time requirement was set at 300ms from the time a command is issued to the time it is received, on either side (e.g. from the robot to the user or from the user to the robot), we clocked at 8ms in our benchmark test environment and 90ms in the actual system performance (e.g. Please see section 5.1.1 Real Time System Performance for more detail).

Having satisfied the real time performance requirement, we were able to focus our efforts on simplifying the design and standardizing:

- Instead of using a home grown file format, we were able to use a table in a relational database, complete with update, delete, and select features.
- Instead of using files, which may have been different under various Operating Systems, we were able to use JDBC to connect to the database, a data format for which was the DBMS's responsibility and hidden from our application.
- Instead of using a home grown handshake or protocol over TCP/IP we were able to use standard HTTP to connect to the server.
- Instead of using home grown rendering, even text, we were able to utilize the feature provided by the browser.

It was interesting to note that although we used resource consuming components and applications such as a relational DBMS, or a Webserver to persist the data or to broker requests, they performed well and within our real time requirements tolerance.

Using these existing components and services, we were also able to design an application that future expansions for any additional command will simply be an HTML development to

construct any command to the server side Servlet. All the rest will be taken care of automatically and will require no changes to any other part of the system.

Finally, inline with the above mentioned ubiquity of access, the new design is also multi-user. As many users as the webserver can handle can access the system (e.g. there are currently no limits but similar to all other systems, the performance will degrade as the number of simultaneous users increase. We did not notice any performance degradation with only three users connected at the same time).

5.5.1 Future Plans

The only feature we did not complete was support for video streaming. By implementing Video Streaming, we will be able to see what the robot sees from the comfort of an office across the world.

The video streaming requires a streaming engine support with the webserver, and some HTML page modifications (e.g. the JSP) to use the embedded video feature of the browser. The video streaming also requires a USB video feed into the server from the robot. In other words the robot is required to be able to capture and send the video to the server, preferably on the server USB port, to be picked up for streaming to the browser clients.

We believe the client side will be 2-3 days of work, and incorporating a video streaming engine with the server will be 10-15 days of server development.

6.0 COMPUTER SUBSYSTEMS – Mapping Tool Description

The map user interface was designed in order for the user to specify a pre-set mission plan for the boat. In other words, the interface enabled the user to plot points on the map continuously, which in turn allowed them to draw a path that the boat would use to navigate. Moreover, through this interface the user could request GPS data from the boat. The GPS data would then be accessed through the interface, stored on the on-shore computer, and be plotted on the map. The following figures explain how the Map Interface works with other parts of the system. The interface, which is located on the computer, receives GPS data that it needs from the data modem through a connection with the vessel (via data modem).

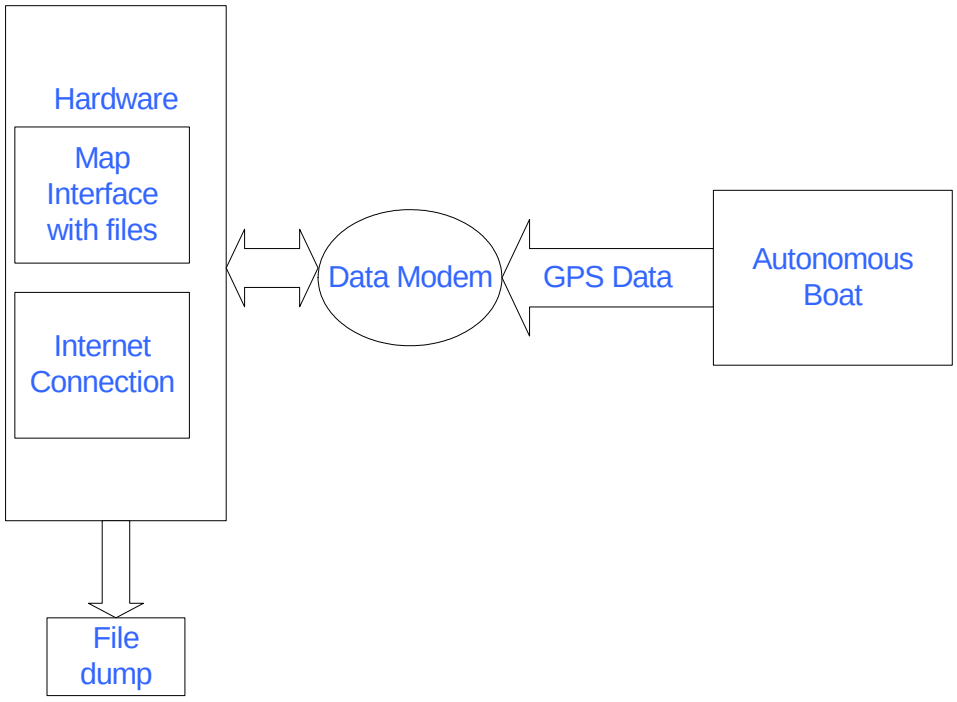


Figure 6.1 Map Interacting with Overall System Structure

In order to implement the mapping tool, there were a couple of requirements that needed to be met:

1) The map should be able to subscribe to an RBNB channel in order to receive real-time GPS data updates and plot them on the map during the mission.
2) The user should be able to toggle on or off historical track of GPS data, that is either a full trail of GPS data or the last position navigated during the mission.

3) The user should be able to also superimpose desired waypoints as they are defined in the mission file and toggle these on or off.
4) It should be possible to define new maps and new coordinate frames for them, so the application can be used for any new location – as opposed to a hard-coded map image.
5) Landmark locations should be pulled from a file that stores these locations and plotted on the map.
6) The map should include basic zooming and manipulation functions.

From the above requirements, the ones that were fully achieved were requirements 2, 3, 4 and 5 that are explained on sections 6.2 and 6.3 in this chapter. Also, requirement 1 was partly accomplished – this is explained more in section 6.1.

6.1 Data Access

We considered two methods to receive GPS data from the boat at the on-shore computer. These two methods included subscription to an RBNB channel and connecting to MySQL database. Although, their full functionality was not implemented, we decided that subscribing to an RBNB channel is a better solution, since connecting to MySQL database carried more overheads. Explanation of how these methods would or did work, are explained below.

Design Solutions

Subscription to an RBNB channel- The way this would have worked was that, when the operator on the on-shore computer made (subscribed) a request for GPS data being transferred from the controller on the boat, the boat created a channel. This channel was transferred to the Ring-Buffered Bus Network (RBNB) located between the on-shore computer and the channel (via wireless connection). The data was then passed to the mapping interface on the computer. Also, it's worthwhile to notice that the RBNB channel was implemented on another software architecture called NETROL. This software was implemented by a couple of SCU students a few years ago and was a rather convenient way of receiving GPS data. The following elaborates on this:

The advantage of subscribing to an RBNB channel was that random access to data was enabled due to the RBNB Ring Buffered nature, whereas an ordinary network bus provided only in-order access of data through the bus and did not allow unwanted data to be discarded. In other words, everything had to pass through the bus to get into the computer.

However, with a RBNB, random data could have been "picked" from any point on the ring structure, which made it faster and more efficient to grab the needed data and made it easier to discard information that was not helpful – in terms of establishing the waypoints.
What the mapping tool managed to do was to take strings in the format that they *would* be received by the GPS (and sent to RBNB channel) and would write the values to a file. They would then be checked for validity or warning signs (these signs are implemented in the string). If the string is proved to be valid, they were plotted on the map. If not, they were simply left in the file without being plotted.
Access data through MySQL server connection- Connection to the database was made through implementing Java code through Java Database Connectivity (JDBC) as the standard SQL access interface. Access required the database name, the table name and type of data that needed to be received, and the Internet Provider (IP) address of the remote computer that contained the database. Only when all of the parameters' names were correct was the connection made. However, it was almost impossible to obtain a single IP address, because the database would be generated all the time connecting to different computers that had different IP addresses.
An Advantage of receiving data from a MySQL database would be the server's speed and small size. Also, there would be very little chance of the system crashing due to MySQL capability to handle large databases on any operating system with enough Random Access Memory (RAM) and sufficient disk space. In other words, the database would work well with a wide variety of computers and did not require a dedicated system.
However, a disadvantage would be that despite its speed and efficiency it would store all the data on a MySQL table. This would not allow the user to throw away unwanted data and only receive the needed data. Disposal of unwanted data would only be possible through explicitly deleting an element on the database table, a programmer-level action and not something the user would have been advised to do.

Final Design:

On the whole, the content of the data received had a higher priority than how fast we received it. A speedy data transfer would make it easier for the user and decreased the time it took for the mission to finish. However, if the right format of data – relevant to waypoint structure – wouldn't be obtained (latitude and longitude coordinates) there would be more overheads in trying to discard them through a *remote* database.

Therefore, we decided that subscription to an RBNB channel would be more efficient, as it would allow data picking from anywhere in the network bus and would remove unwanted data confrontation by the interface. As a result, we now want to have an RBNB channel running simultaneously on the server. The map interface that listens to the Data Turbine at the moment and has the required Data Turbine Listener functions implemented in it. What needs to be achieved is to make the Data Turbine on NETROL "know" that the map interface is listening to it. This can be done by having the Data Turbine Listener have the relevant map functions implemented in it, so that it knows where the request is coming from, so it will in turn respond to it.

6.2 Data Analysis

The issue of analyzing data came into play once the data had been transferred from the boat to the map interface. Data analysis dealt with how the computer handled the received data. Once the GPS data was received, the interface needed to display it on the interface and also store it on the computer. After the user subscribed a request and the GPS channel was formed, data was stored on a GPS mission file that could be accessed by the map interface. The design solutions for the data handling and storage were Single-file Data Storage and Multiple-Files Data Storage.

Design Solutions:

Single-file data storage-Once the mission-gathered data was displayed through a mission file, it was stored on one log file to keep track of the mission history. At the end of the mission, this file contained coordinates of a combination of ordinary points and interesting landmarks. Advantages from the Operating System's perspective: The file was stored on contiguous memory bytes, meaning high access speed and small memory usage. Also, there was little overhead in performing file functions such as "open" and "close", because there was only one file. One disadvantage of having one storage file was that the file was increasing in size as the mission proceeded, making it harder to browse through the file in order to edit or remove a certain point. When the mission was finished it would have been hard to look through the file and tell which waypoints were passed, for example, without an in-depth analysis.

Multiple-files data storage- This approach was very similar to the previous one, only multiple files were generated to organize data by different types of points. For instance, one file was generated to store data on all GPS data stream, and another file was used to save data on all the

waypoints that included interesting landmarks (this file was preferred to be the map file which contained information on map size as well.) An advantage with multiple storage files was easier access of data by the map interface due to the organization system. Therefore, retrieving points in specific files can be performed quickly. Also, this approach makes it easier for the user to toggle “on” or “off” landmark points. A disadvantage of this method was increased CPU usage.

Final Design:

Looking at these two approaches and considering the interface’s functionality, it was concluded that a speedy and well-functioned operating system was not as important as the user convenience. At the same time, using a huge text file containing information about all points might have been too large and refused by the computer. Since the flexibility of the program and the convenience in using it was important, second approach was chosen.

Implementation:

The entire stream of data coming from the boat was saved on a text file, the landmarks on a separate “map info” file, and the latitude and longitude scales and origins for each map on the “mission” file. The data was stored on multiple files, which were renewed (edited) every time a new mission took place. Also, a new map info file was generated each time a new map was used for the application.

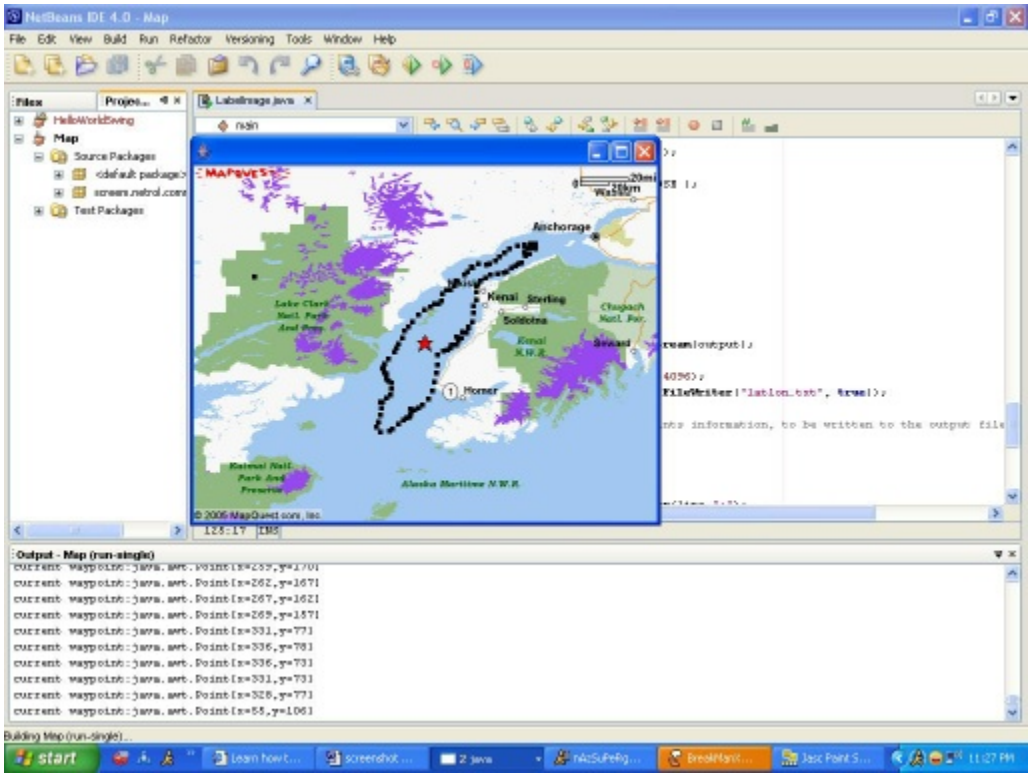


Figure 6.2 Screen Shot of Navigation Map

6.3 Map Structure

The map interface was used to define the pre-set mission for the boat. When the user clicked on the map, the mission plan appeared, graphically, as well as the real-time GPS update from the vessel. In order to have the map interface up and running, there were a few requirements that needed to be met.

In order to receive GPS data, the user needed to subscribe to an RBNB channel, as described in the Chapter 5.4. In order to subscribe to a specific RBNB channel, the user needed to make a specific request to the channel; in this case it was a GPS request. The channel then connected to the boat via wireless connection, which enabled the boat to send back data streams containing latitude and longitude values to the user. Once these streams reached the on-shore computer, they were stored in a mission file. The map interface read these values from the mission file as they came in and plotted them on the map. Therefore, during a mission the user was able to see a full trail of the GPS data stream plotted on the map and stored in a file.

The landmarks of the mission were stored in a separate file containing other information about the map. The specified landmarks were filed after the map information. The interface could then have read the landmark values from the file and displayed them on the map, if the user specified, right before the map GUI was launched. The file containing the generic map information was generated every time a new map (image) was used by the application. The user was also able to define new maps to the interface each time they decided to launch the program. The coordinate frames for each new map were specified in a mission file and read when calculating latitude and longitude values for a pixel point on the map. In this way, the map interface could be used for other locations. Also, the information regarding the previously defined maps was archived and could be referenced. By setting up the map interface this way, the user could view a previously used navigation plan and coordinate the new navigation plan around this information.

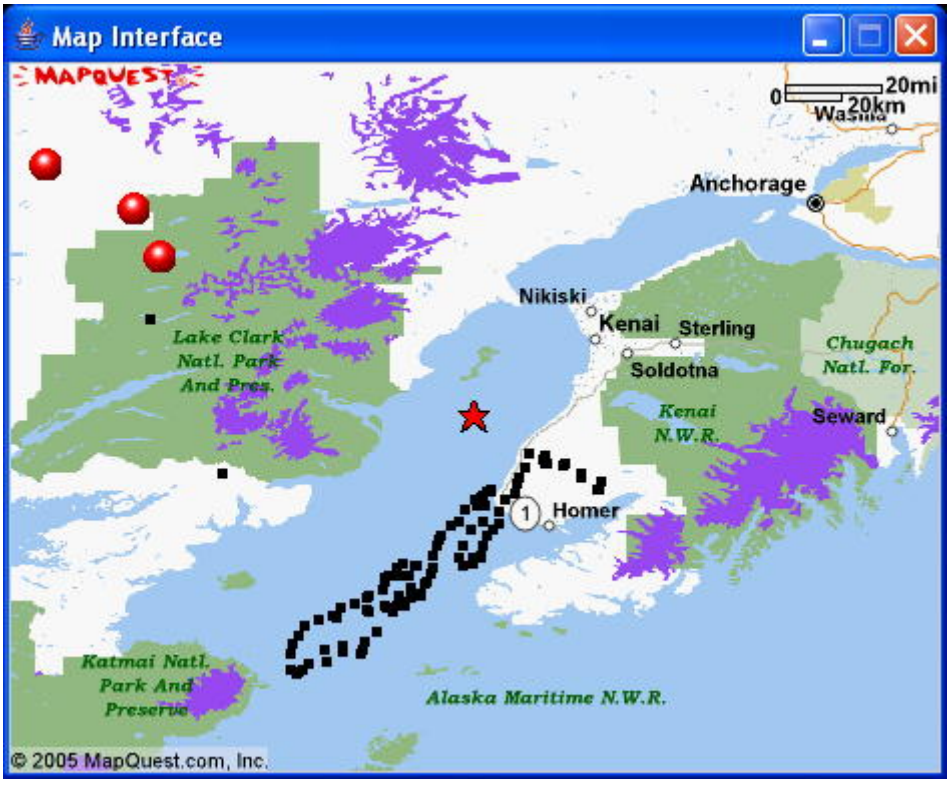


Figure 6.3 Navigation Map for GUI

6.4 Map Architecture Diagram

The following diagram shows how the mapping interface would interact with NETROL software to receive GPS data. Once the request is made to the RBNB channel, the RBNB would connect

to a GPS channel, in order to have data transferred to it. This data will then be coming to a data stream text file associated with the map interface. The mapping tool will then plot this data on the map image. Also, the landmark points for each mission are saved in the map information file, which is generated every time a new map image is given to the interface. The map interface reads these landmarks and displays them on the map. The figure is displayed on the following page.

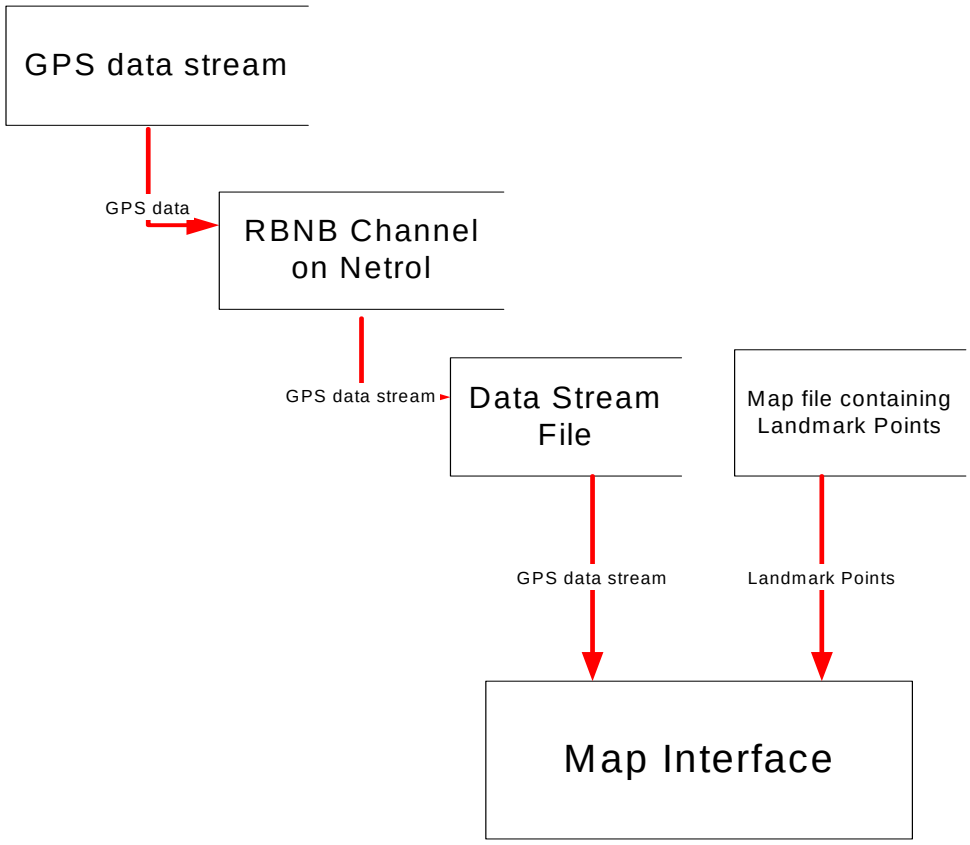


Figure 6.4 Map obtaining data from RBNB channel (How it'll be when fully implemented)

7.0 PATENT SEARCH

In order to maintain the proper draft and stay level in rough seas it is necessary to develop active ballast and trim system. This will allow the boat to compensate to any changes in mid-mission without returning to the dock for manual ballast or trim adjustments

7.1 Motivation

The current SWATH design relies on the two submerged hulls to displace all the water required to create enough buoyancy to support the vessel. While the vertical struts can also displace some water the relative percentage for The WASP is relatively low (about 6% or 50 lbs when completely submerged).

The reason an active ballast system is needed is primarily to compensate for the deployment and retrieval of the instrument package. In air, the package weighs about 44 pounds, however in water it only weighs about 36 pounds. This alone is enough to change the draft by 2.8 inches. The instrument cable weighs about 45 pounds in on the deck, but when fully deployed it only weights about 13 pounds in the water. The total difference when deployed is 40 pounds, enough to change the draft almost the full length of the struts.

If the boat were riding with the main deck only one inch above the surface, assume a very calm day, when fully deployed, the pontoons would be less than an inch below the surface. While this will not cause the boat to sink, it does defeat the purpose of the SWATH design, which is intended to reduce boat motion by decoupling the vessel from the wave motion. If the hulls ride this close to the surface it will be highly effected by the waves especially so when the instruments are deployed.

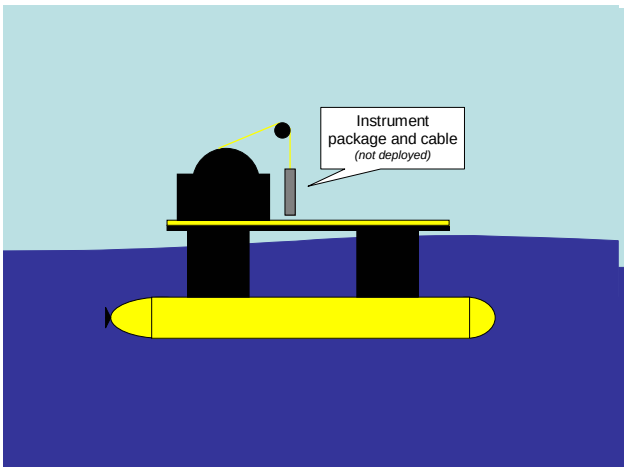


Figure 7.1 Ride Height (no active ballast)
(Instruments not deployed)

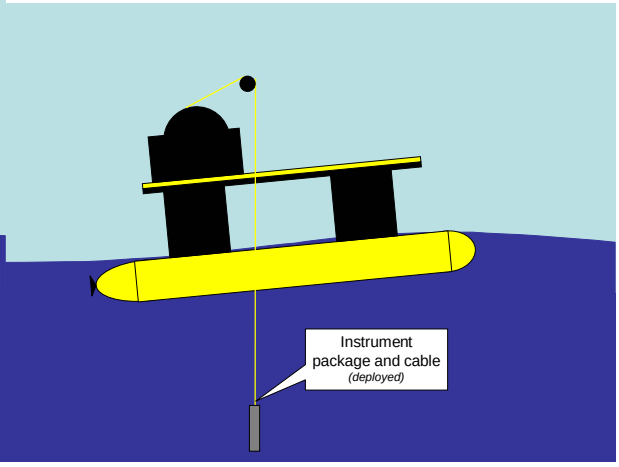


Figure 7.2 Ride Height (no active ballast)
(Instruments deployed)

While this will not cause the boat to sink, it does defeat the purpose of the SWATH design, which is intended to reduce boat motion by decoupling the vessel from the wave motion. If the hulls ride this close to the surface it will be highly effected by the waves especially so when the instruments are deployed.

By controlling the flow of water in and out of compartments within the submerged hulls, weight can be added in small increments to adjust the ride height or draft of the vessel. This will allow for continuous adjustment, as the instrument package is deployed and retrieved.

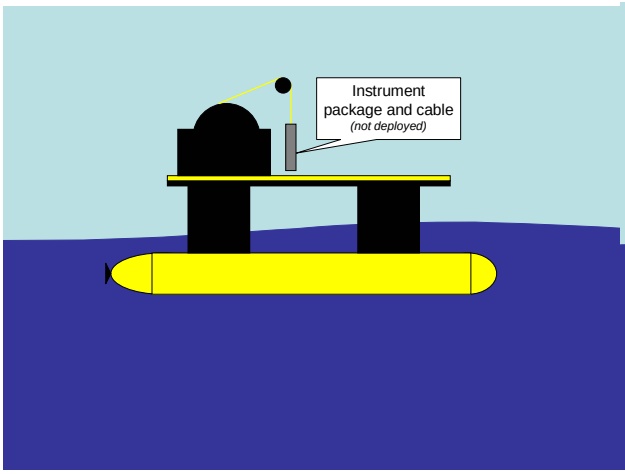


Figure 7.3 Ride Height (with active ballast)
(Instruments not deployed)

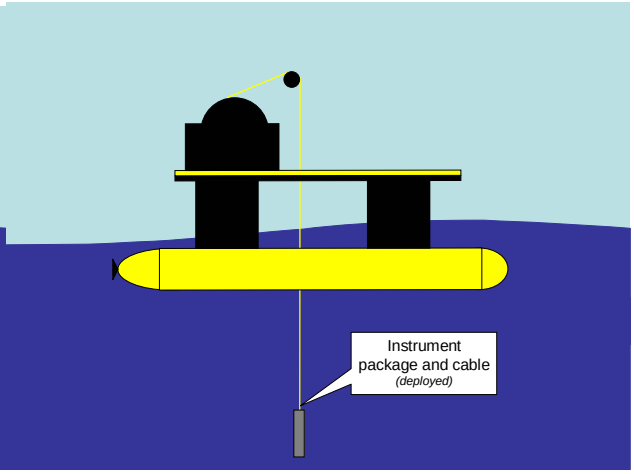


Figure 7.4 Ride Height (with active ballast)
(Instruments deployed)

The trim is an important part of the system. Many times a strong wind or pounding waves can cause a boat to tilt. By placing the ballast compartment in the extreme corners of the vessel (i.e.

the front and back of each of the hulls) and running each independently, ballast can be added to one side or even one corner of the boat to compensate for any forces that may cause it to tilt

7.2 System Overview

The compartments are located in the aft and stern of each hull, creating the most effective trim compensation. They will be water tight except for three holes; one to let water in, one to pump water out, and a valve to let air in or out. There are two main subsystems of the active ballast and trim system; letting water in and pumping water out.

In order to let the water in an electronically controlled solenoid valve is used. By opening the valve, the outside pressure will force water into the chamber displacing the air, which will exit through the air valve. This will decrease the hull's displacement, lowering the buoyant force, and increasing the draft. Each of the valves will be operated independently to adjust trim.

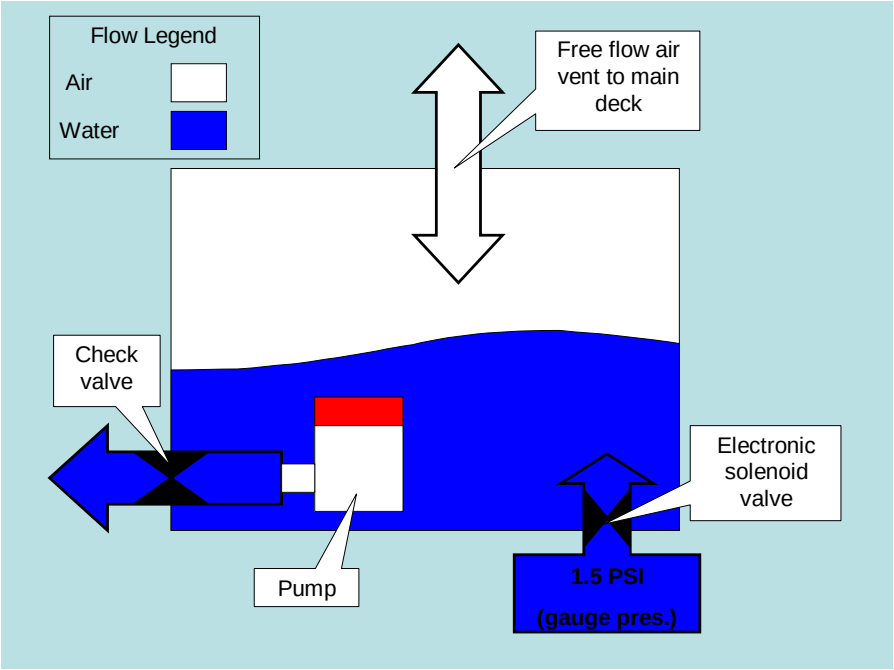


Figure 7.5 Active Ballast Container Block Diagram

In order to pump water out, a small bilge pump is utilized. By pushing the water out through a check valve the water can not flow back in through the pump. Air will enter back into the chamber through the air valve.

7.3 Patent Classification

Class 114 Ships
Sub-class 125 Water tanks – Devices in which ballasting is secured by water tanks adapted to be filled or emptied, thereby distributing weight, which steadies and trims the vessel in sailing, etc.

7.4 Prior Art

Much of the prior art is focused on mono-hull and standard type boats.

The First prior art is the SWAP (Small Water-plane Area Platform) type floating platforms US patent 4,864,958 by Belinsky on September 12, 1989. The object of the invention is to improve stability of floating platforms (ships) during the high seas and to enable them to automatically keep the platform (ship) from listing. Application of this invention were seen for floating drilling platforms, passenger ships, crane ships, floating terminals, SWATH-type ships, etc. Belinsky's invention relied on a cylinder and piston device to change the actual volume of water displaced.

On April 13, 1976, in United States Patent 3,949,693 Bauer designed a Partially submerged floating platform utilizing a polygonal, submerged body. Constructed from concentric pipe sections, the space between each two concentric pipes is could be used for storage as well as ballast tanks. Columns also constructed as upright concentric pipes extend from the submerged float to carry the main platform e.g. a drilling derrick.

Both US Patent 4,174,671 (November 20, 1979, Seidl) and US Patent 6,260,502 (July 17, 2001, Kratz) describe designs for Semi-submersible Vessels. The former describes a marine vessel, which comprises a platform member with two parallel hulls submerged below the water when the ship is in operation. Each of the hulls consists of an after section, a forward section having a smaller cross-sectional area than the after section and a central section having a cross-sectional area substantially smaller than that of the forward section. Both forward sections and both after sections are joined to the platform member by strut-like members which are tapered so that their

water-plane areas increase rapidly between the waterline and their junctions with the platform member.

The latter of the two goes into more detail, including improved elliptically shaped columns, haunch supports and reinforcing members provide stability and resist transverse forces. A skirt design increases stability at the transition point. Improved deck designs reduce environmental impacts and provide additional support against transverse forces.

Other related patents and complete patent files can be found in Appendix M.

7.5 Conclusion

Due to the extremely large quantity of prior art in the topic area (over 1400 pieces) it is difficult to say if this particular design is patentable. While there are similar types of ballast systems for other ships, most of the ballast systems for SWATH type ships had deformable displacement tanks. It is almost certain that this has never been done for a boat of this scale, as it is by far the smallest ever built. It would seam that the idea is too similar to devices used on mono-hull and other ships to be patentable.

8.0 ENGINEERING STANDARDS AND CONSTRAINTS

8.1 Sustainability

The word “sustainable” is used in two ways in the Engineering Handbook. The first way it is used refers to “sustainable communities,” which is a community that uses the resources available to it to meet the needs of the community while at the same time ensuring that future generations will be able to meet their needs as well. The second way it is used refers to “sustainable engineering;” this refers to the process of developing engineering devices, products, and systems that will be used in sustainable communities. Although the team has made use of materials that use non-sustainable resources, the materials that have been used should not create waste in any way such that they contaminate the environment. By limiting pollution the team has ensured that they are not going to harm the environment for future generations. And although they could have used solar power to power their vehicle, in order to ensure that they used sustainable resources to the best of their abilities, it was not a practical application for their endeavor because they simply did not have the funds to make use of solar energy; nor is solar technology advanced enough to make it a practical source of energy (technology has not yet sufficiently harnessed the capabilities of solar power to justify the investment). Because of these reasons the team used batteries, but they left the door open for the addition of a generator. In designing the ASV the team tried to minimize the amount of materials used, minimize energy consumed, pollute very little (if at all), and make sure that it failed very rarely (if at all). They designed the ASV so that it could be updated and upgraded easily. By doing these things they worked with the idea of “sustainable engineering” in mind.

8.2 Usability

The ASV should be easy to learn how to use, effective and efficient in its tasks, relatively free of failures, satisfying to use, and sustainable. The team understood that the customer, University of Alaska, wanted the ASV to take very technical readings of the environment, both above and below the surface of the water, and relay them back to an onshore computer. The ASV had to be very easy to use and maintain; they also assumed that the user will have limited engineering training, if any, so they designed the ASV with these things in mind. An example of this is seen in the Graphical User Interface (GUI), it has been designed with simple “point – and – click” operation in mind.

8.3 Health and Safety

A product may fail because of any one of five reasons – material failure, poor design, environmental effects, human error, or a combination of the other four. One way to minimize material failure is to adequately research all material specs as they pertain to the customer's environment. In addition to in depth research, extensive testing must also be completed. Unfortunately, environmental effects are nearly impossible to test for especially because of the drastic differences between Alaska and California, and because environmental affects will only come to light over extended periods of time. The best the team could offer was to make sure that the ASV was suitable for salt water use, and that the materials and components used were rated for the temperature ranges that exists in the Kasitsna Bay area. By designing a robust ASV, designing an easy-to-use and understand GUI, by documenting the entire project thoroughly, by authoring a concise user's manual, and with thorough user training, the team hoped to combat most foreseeable human error. It is obvious that in order for the ASV to perform reliably and safely over time the team must develop and implement thorough and effective methods of testing.

One of the largest safety concerns is the fact that the end goal is to have an autonomous vessel operating in public waterways. This of course presents many obstacles to overcome. They include incorporating an obstacle avoidance system, and ensuring that the ASV is clearly visible to other boats. A strobe will be attached to a mast to aid in visibility. Eventually an automated obstacle avoidance system needs to be implemented; but for the short term, the idea is that the user will be able to watch a live video feed of the boat's position, and, should the need arise, manually interrupt the autonomous drive, and manually drive around or avoid obstacles in the water, including other boat traffic.

8.4 Environmental Impact:

The first thing that the team had to keep in mind when designing and manufacturing the ASV was that its purpose was to take environmental readings of a protected bay in an isolated part of Alaska. Kasitsna Bay, and the surrounding area, contains many aspects of life that are very important, very interesting, and very rare. Marine biologists at the University of Alaska have realized the importance of the Kasitsna Bay area and have invested large amounts of money to

the study of the area, including a multi-million dollar research center and the funding for the ASV. Because of these reasons, the ASV needed to collect data from the environment without altering the environment through pollution, or any other ways.

8.5 Social

Engineers exist for the purpose of bettering life. One way that life can be made better is through education. This is precisely the reasoning behind the ASV. In building the ASV with the capability of taking environmental readings via certain instrumentation packages (i.e. CTD – Conductivity, Temperature, and Depth), we are aiding in the education of marine biologists studying the Kasitsna Bay area. Without this ASV, or a comparable vehicle, it would be extremely difficult and time consuming to take the same measurements over such a wide area as desired by the University of Alaska.

8.6 Manufacturability

When designing a product for a customer, or a given market of customers, engineers must design for manufacturability (commonly referred to as DFM). This means that the product must be designed in such a way that it can be manufactured efficiently, reliably and within acceptable costs. In the case of the ASV, DFM was not a concern. The ASV is a custom product, not intended for mass production. However, when designing the ASV, the team did have to keep in mind the eventual transport of the product to the customer in remote Alaska. Because of that, the team designed a modular product. All electrical components have been distributed in wafer boats. The wafer boats provided both a waterproof enclosure for the components, made them very accessible, and allowed for secure transport. The platform of the ASV could also be dismantled for relatively easy transport – this includes removing such things as the winch and battery charger. Keeping with the theme of modularity, the pontoons have been designed so that the main portion, 8 feet in length, can fit into a 757 cargo container and the end caps can be easily attached. In addition to designing the ASV around modularity, in order to increase reliability and ensure good connections, the team had all their circuit boards fabricated.

8.7 Ethics

Everyone faces a question of ethics whenever faced with an issue of right and wrong. The ASV team, however, had to worry very little about ethics when designing the ASV. The team's only encounter with the idea of ethics was noticeable in the research phase. When researching sources of energy, to power the ASV, the team had a multitude of options. They settled on four, six-volt 370AH batteries, and decided to build a custom enclosure for the batteries. The custom enclosure allowed air to get to the batteries to allow the batteries to vent during charging, and also prevented them from accidentally spilling and polluting the water. The team could have chosen cheaper and more cost effective sources of energy, but ethically they knew that it was wrong to employ such sources of energy because of the risk of pollution.

8.8 Political Matters

Engineers often are the first to recognize design problems, and they carry an obligation to inform the community of the situation. The obligation arises from the inherent moral right of an individual or community to knowledge of matters that are of concern to its health, safety, and well-being. These issues relate back with ethics. If the team at any point ever thought that their ASV was at risk for problems that would infringe on public health, safety, or well-being they would have been obligated to inform the customer, and/or the public, or simply change their design; this was never an issue, however.

8.9 Economic

Immediate thoughts towards economics drift towards two aspects: build cost, and cost of ownership/operation. The team was fortunate in that we had $2950 donated to the project, this was primarily in the form of free products such as the PVC pipe that the pontoons were constructed from and the honeycomb aluminum that the structure was made of. Out of pocket expenses, which were paid from a variety of sources totaled $14,820. See Appendix I for a complete budget and the Acknowledgments for a list of donors and contributors to the ASV. Cost of ownership of the ASV will include obvious maintenance costs, such as new batteries, fuel for the future generator, upkeep of an adequate on-shore computer, and costs such as storage, transportation, and docking/deploying related expenses.

8.10 Compassion

Compassion happens when we observe that people are suffering, and when we do what we can to alleviate that suffering. If the second part is not there we might call it sympathy or perhaps pity, but not compassion. The ASV was designed for the purpose of aiding in the collection of scientific data, not to alleviate suffering; so compassion was not an aspect that drove the design of the ASV.

8.11 Lifelong Learning

To be a successful lifelong learner you must have a motivation to learn and the tools or methods to facilitate that learning. The ASV was designed to be one of those tools to facilitate that learning; a tool that was new and innovative.

9.0 CONCLUSION

9.1 Conclusion

This year, the ASV team set out to develop an autonomous vessel to sample oceanographic data at specific stations and depths. Our specific goal for this year was to develop a prototype with certain subsystems such as flotation, propulsion, power, vertical deployment, structure, navigation, science instrument controller, wireless communication, and a graphical user interface. We were able to accomplish most of our goal including building the structure: pontoons, vertical struts, platform, and electrical circuitry and purchasing the winch, batteries, and protective boxes. Integration and testing will be performed during Summer of 2005 as part of an SCU-MBARI project.

9.2 Evaluation of Design

The design changed throughout the year and we're confident in our final design: the SWATH. Because the flotation system is submerged, the change in water height does not affect the stability of the boat as drastically. The amplitude of the oscillation of the depth of the instruments is also dampened and the thrusters are constantly submerged. The use of the honeycomb aluminum kept the structure strong and light. Also, by mounting all components onto plastic grating, there is a large amount of adjustability in placement. The use of Atmels will prove easy to learn and continue working with. By using basic languages like Java, the software is easy to adjust if any project specifications are changed and it is compatible with most computers.

9.3 Future Work

Although we accomplished most of the objectives set forth, there is still more to do to meet the complete requirements of the customer. For the mechanical subsystems, a dynamic ballast system should be added. Our idea is that there are sections in the pontoons that contain several pumps that can control the amount of water contained in that section. This allows the ASV to remain more stable in rougher sea conditions. Also, testing of the system in an ocean environment or a simulated environment is needed to determine the reaction of the vessel to waves. There have been numerous miscalculations with resonance and a similar reaction to waves would prove detrimental.

Although we estimate the batteries on board will give the required amount of hours of operation, a generator system should be considered. We anticipate that the scope of the project will increase and with it, an increase in the length of time the ASV will be operational. The generator will add more hours and enable that requirement to be fulfilled without changing the power subsystem.

In order to deploy instruments, three conditions must be fulfilled. First, the wheel that guides the tether from the level wind to the instruments and keeps a safe radius of curvature needs to be fabricated and mounted. This wheel will need to drop the instruments through the center of the ASV and ensure that the instruments rise completely out of the water. The second wheel needs to be attached to the level wind to keep the tether from rubbing against rough edges. Because the edge of the winch is close to the entrance of the instruments, the first wheel needs to be mounted at a steeper angle than optimal. A problem arises when the tether tends to rub against the locking mechanism of the level wind. The second condition that must be fulfilled is a protective cage for multiple instruments. This cage should be light yet strong in order to protect instruments in case they come in contact with a hard object, yet light enough to not create an unbearable load on the tether. The cage must support the various instruments that the scientists require and should be designed for easy removal and installation of new instruments in the case of repairs or upgrades. Finally, a guide for the instruments is desired for periods when the instruments are retracted and the boat is in motion. The instruments will tend to sway in windy conditions, with swells or waves, and with general motion. This guide will reduce stressing the tether and reduce error in the deployment of instruments.

APPENDIX A: WORKS CITED

[1] http://www.westnurc.uaf.edu/kbay/academics.html. Kasitsna Bay Laboratory Academics. Modified May 2005.

[2] www.mbari.org. Monterey Bay Aquarium Research Institute.

[3] http://rsl.engr.scu.edu. Santa Clara University's Robotic Systems Laboratory Webpage.

[4] http://kachemakkayakfest.com/map.html. Sponsored by Homer Chamber of Commerce. Updated 2005.

[5] Survey response from Ted Otis, Alaska Fisheries…

[6] www.westnurc.uaf.edu/kbay.html. Kasitsna Bay Laboratory homepage. Modified 2005.

[7] www.asv.uk.org. ASV Limited,

[8] www.auvlab.mit.edu. Massachussetes Institute of Technology's AUV Lab.

[9] www.nuwc.navy.mil. NAVSEA Underwater Warfare Center

[10] Bluefin Robotics Corp. 237 Putnam Avenue, Cambridge MA, 02139; 617-498-0067; http://www.bluefinrobotics.com.

[11] Western Michigan University. www.geology.wmich.edu/Kominz/windwater.html. "Sea State Photographs for Determining Wind Speed: The Beaufort Wind Force Scale (From NOAA)." Retrieved 5/5/2005.

[12] www.honda.com Honda Generators

[13] www.Warn.com Warn Winch

[14] http://hubbard.engr.scu.edu/embedded/ describes the bootloader.

[15] www.atmel.com AVR processors. Atmel Corporation 2005

[16] www.robot-electronics.co.uk Compass. Devantech

[17] www.garmin.com GPS. Garmin LTD

[18] www.roboteq.com Motor driver. Roboteq Inc.

[19] www.maxstream.net wireless modem. MaxStream, Inc.

R D Instruments USA
9855 Businesspark Avenue, San Diego CA, 92131-1101; 858-693-1178; fax:858-695-1459; rdi@rdinstruments.com; www.rdinstruments.com

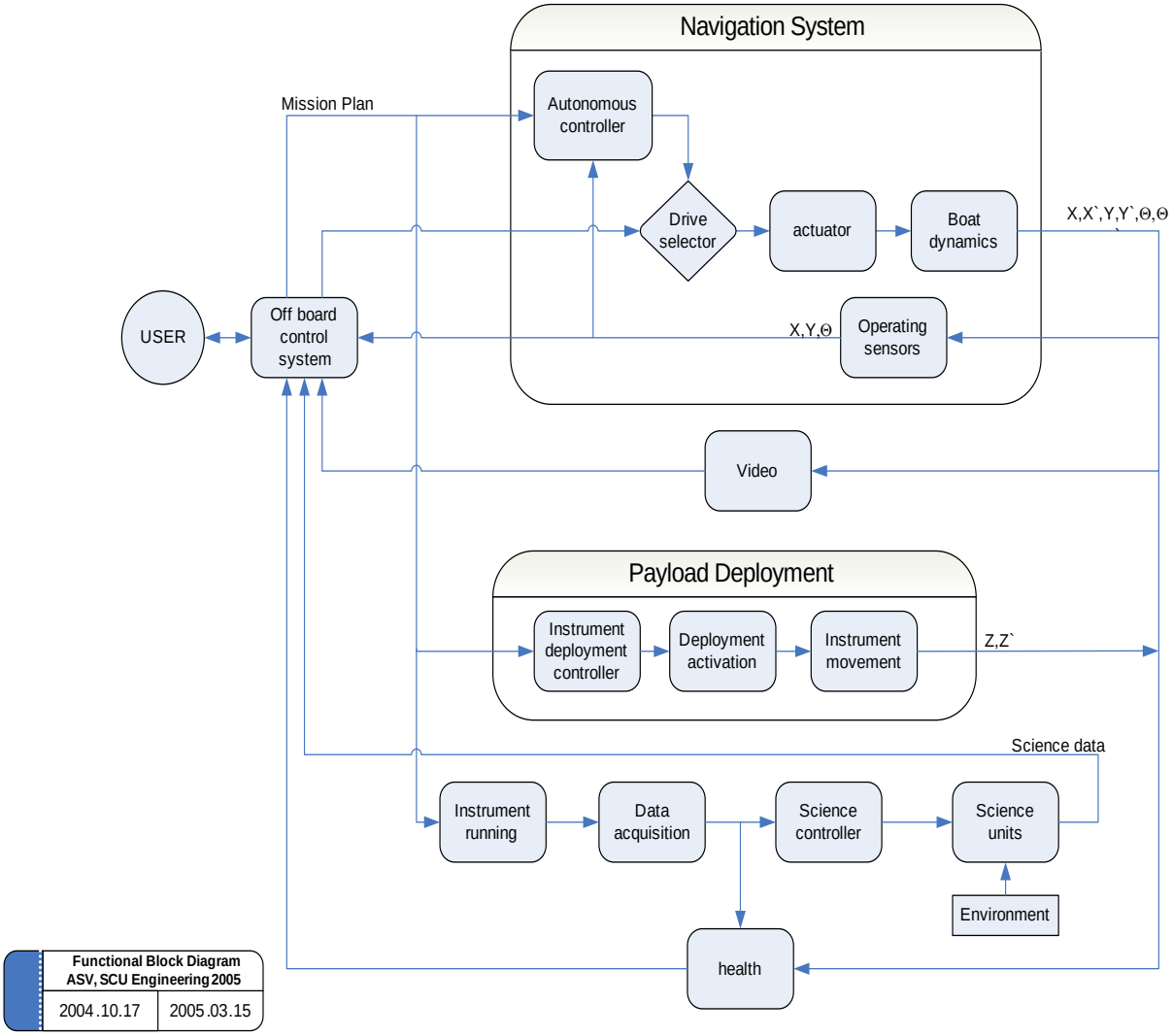
Sea-Bird Electronics, Inc.
1808 136th Place NE, Bellevue, WA 98005; 425-643-9866; fax:425-643-9954; seabird@seabird.com; www.seabird.com

West marine products Inc. (2004). *West Marine: Master Catalog.* USA:Author
Info also at 1-800-BOATING or west marine.com or local stores

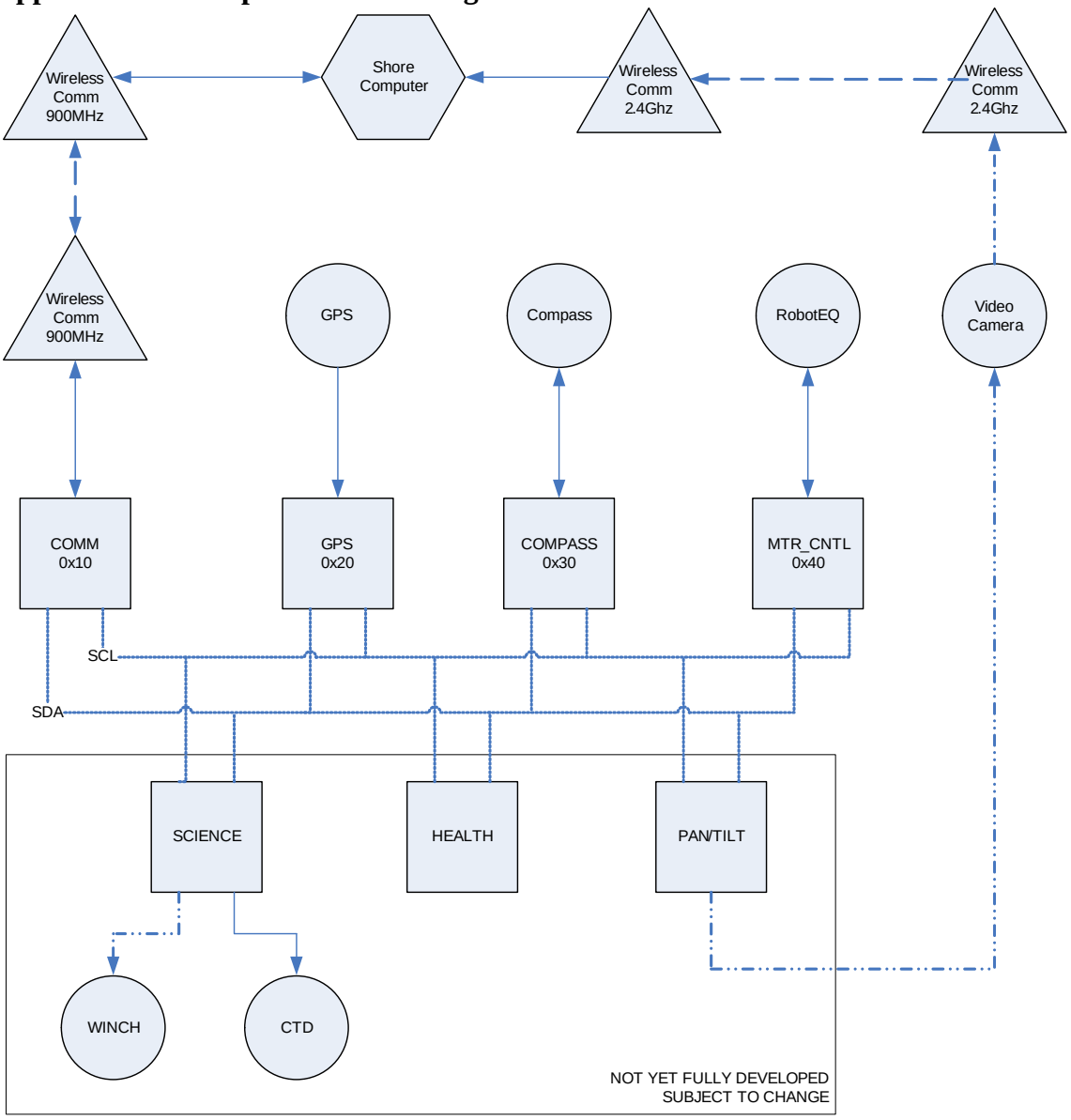
WET Labs
PO Box 518/620 Applegate St., Philomath OR, 97370; 541-929-5650; fax:541-929-5277; wetlabs@wetlabs.com; www.wetlabs.com

APPENDIX B – BLOCK DIAGRAMS

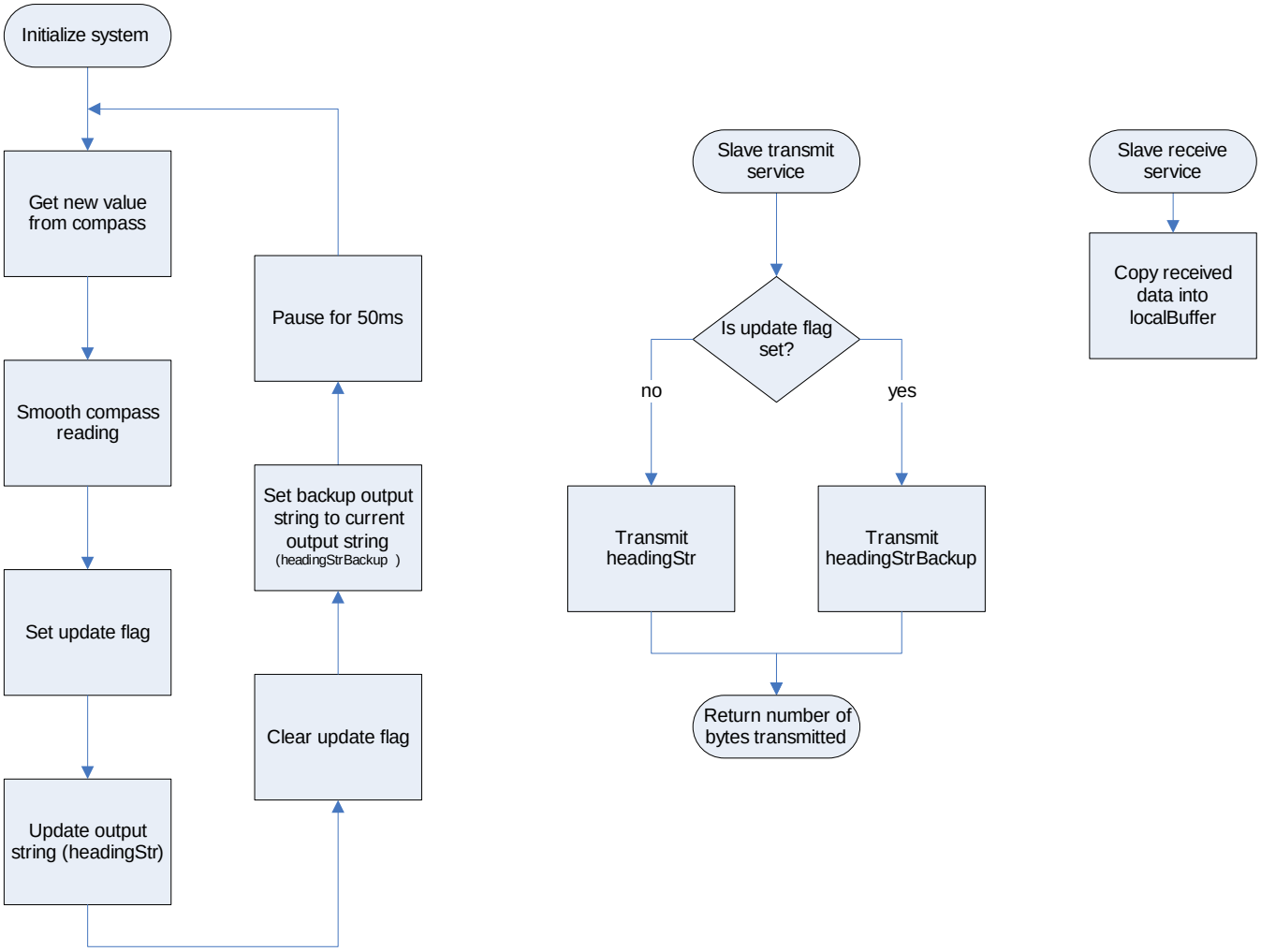
Appendix B-1 Functional Block Diagram



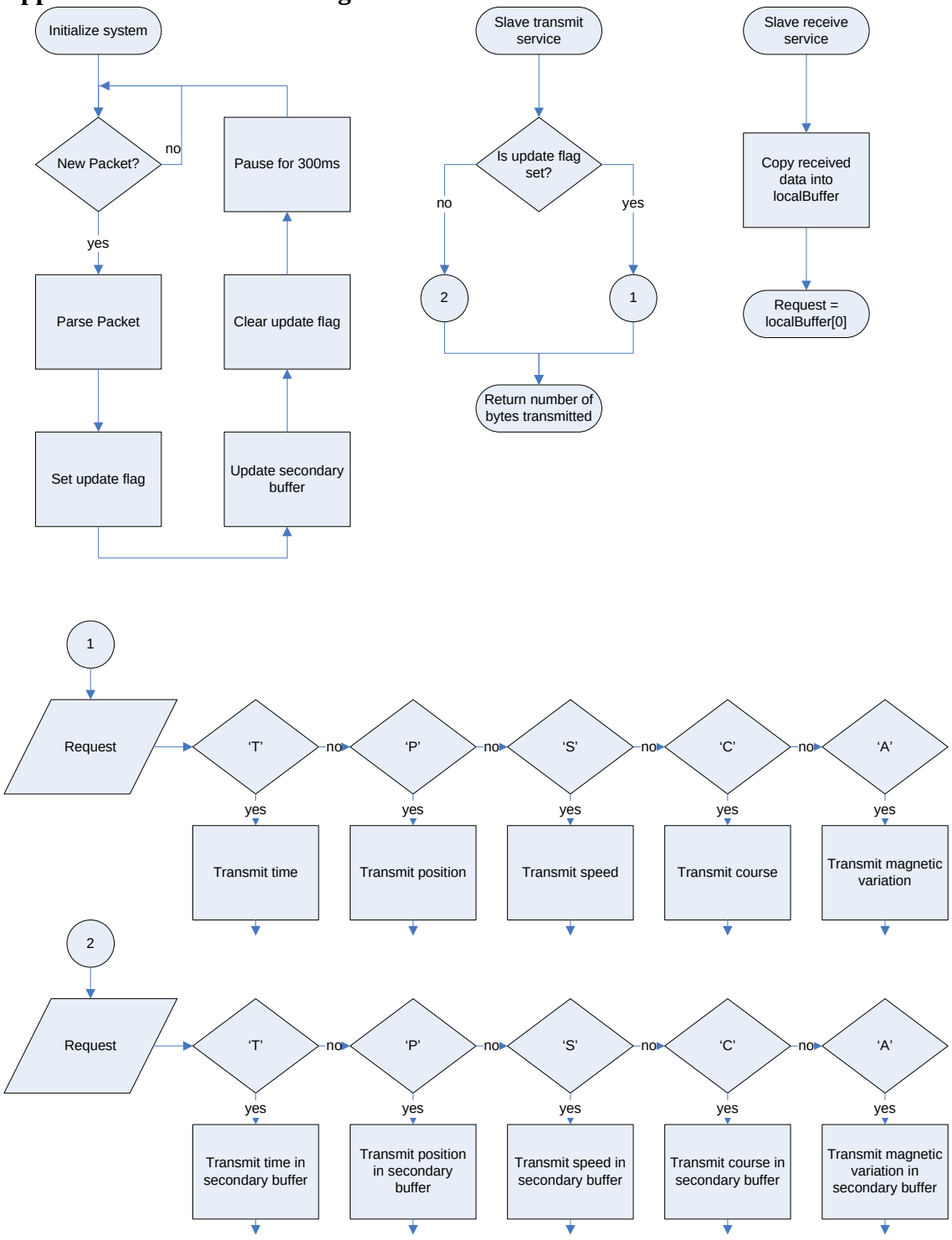
Appendix B-2 Component Block Diagram



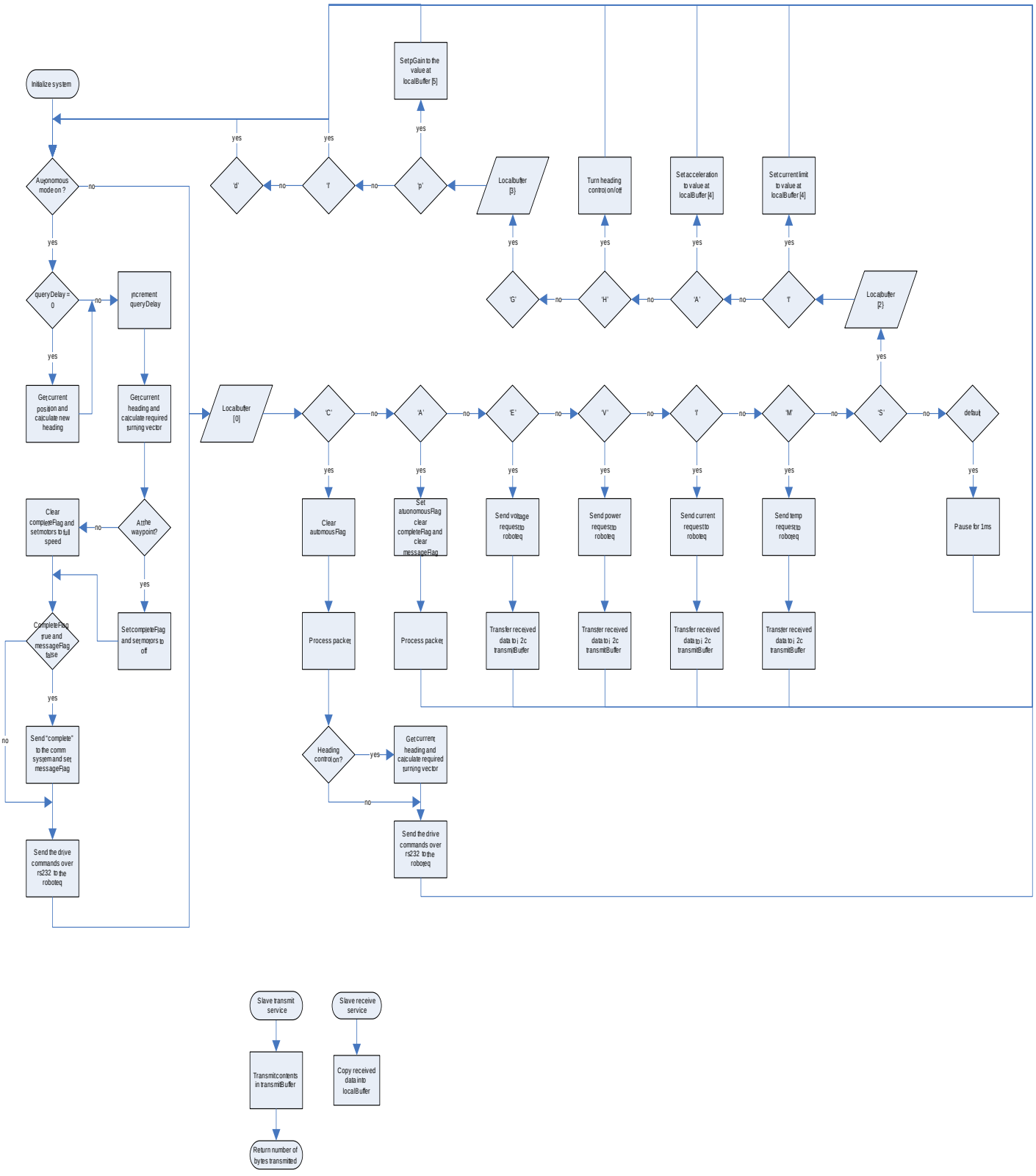
Appendix B-3 Compass Block Diagram



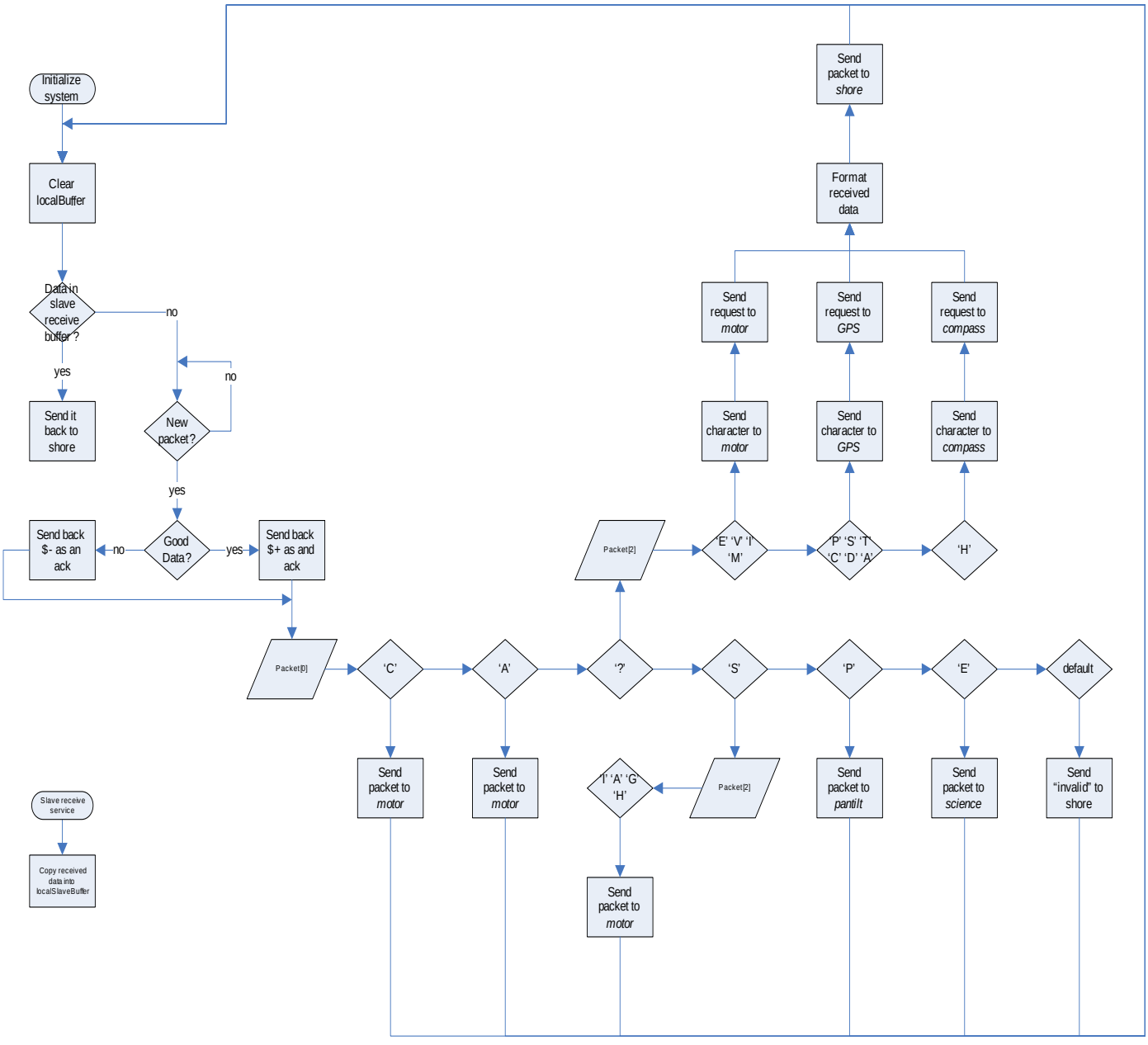
Appendix B-4 GPS Block Diagram



Appendix B-5 Motor Controller Block Diagram



Appendix B-6 Communications Controller Block Diagram



APPENDIX C—CUSTOMER NEEDS SURVEY RESPONSES

Appendix C-1

Response from Geoff Wheat

October 2, 2004

Question/Prompt	Customer Statement	Interpreted Need
Typical Use	I want the ASV to service a 1/4 mile bay	The ASV travels in a 1/4 mile bay
	I want the ASV to be more efficient than scuba diving	The ASV takes measurements faster than a human scuba diver
	I want the ASV to explore areas that are unsafe for divers	The ASV takes measurements in places that are too dangerous for humans
	It should make 12 stops	The ASV should stop at 12 stations
	I want to be able to change the mission at any time.	The program has an override function
	I want the positioning to be fairly accurate	The ASV reaches and keeps a position within an acceptable tolerance
	I don't want to have to go out to retrieve the ASV every time it goes on a mission	The ASV should return to the dock when it is finished with its mission
	This thing is going to be heavy, so I want to be able to take it out of the water using a hook and dockside winch.	The ASV should have built in loading and unloading features
	Maybe take some topside weather data.	The ASV should be able to support topside instruments
Conditions	I'd send it out in 2' waves	The ASV should withstand 2' waves
	The temperature will drop to about low 20s Fahrenheit	The ASV should withstand 20 degree temperature
	The current and the winds will most likely be in different directions	The ASV should be powerful enough to navigate through the current and wind
	The tides change pretty drastically	The ASV should have depth sensing
	The current is pretty strong	The ASV should be able to hold positions within an acceptable tolerance
Attachments	For scientific instruments, an Acoustic Doplar Current Profiler, a Conductivity, Temperature and Depth sensor, a fluorometer, a dissolved oxygen sensor, and an altimeter.	The ASV should be versatile enough to support all instruments: wet mateable and topside equipment.
	I want to be able to purchase batteries from the area	The ASV should use standard parts, including batteries

Appendix C-2.
Response from Jeff Ota
regarding Ph.D Research
December 2, 2004

Question/Prompt	Customer Statement	Interpreted Need
Typical Use	I need the benthos of a body of water (seafloor or lake floor)	System must be able to support instruments to capture benthos of a body of water
	SCU infrastructure is setup so students can maintain and operate ROVs	Set-up infrastructure so that students can maintain and operate system
	Current ROV is a bit bulky to transport	System must be easy to transport
Instrumentation	Main issue when choosing a product-quality of data collected (video), ongoing and operation cost (both financial and time)	System should support high quality instruments, be inexpensive and time efficient in maintenance and operation
	I need stereo video	System must support stereo video
	I would like to collect data twice a year	System will be used twice a year
	Data is collected underwater on the seafloor or lake floor	Data will be collected from the sea/lake floor
	Measurements will be taken less than 300 feet deep	Maximum depth will be 300ft
	Data will be taken instantaneously at the video frame rate	System will not need to stop at each location
Conditions	Possibly marine life, hopefully not	Design safety implementations for marine animals
	No nesting areas	There will be no nesting sites in the area
	Lots of boating traffic in Tahoe, little boating traffic in Monterey	System must be visible to other boaters
	Safety precautions previously taken-tender boat, training of students, and experienced captains	System must be tender. Personnel must be trained and experienced
	No time constraints	There is no time constraint for each mission
	Strong currents/large waves in Monterey, None in Tahoe.	Design for strong currents and large waves
Interface	I prefer a graphical interface	System should utilize a graphical user interface
	I want to have an update at least every second. 5Hz would be optimal	System must have a real-time update with minimal delay
	The mission does not need to be adjusted in real time	The system does not need a manual override
Storage/Maintenance	There are piers at both Tahoe and Monterey	System must incorporate a docking mechanism that is compatible with the existing docks at locations
	The unit will be stored in the SCU Trailer	System must fit and be storable in a trailer
	The unit will be maintained in the SCU Machine Shop	System will be maintained utilizing minimal equipment
	An onboard generator will be available for recharging	System will need to refuel generator fuel

Appendix C-3.
Response from Jeff Ota regarding possible project in El Salvador
December 2, 2004

Question/Prompt	Customer Statement	Interpreted Need
Typical Use	I need the vehicle to fly underneath some vegetation to get riverbed heights	The system must be able to manuever through vegetation
	Existing units are too big…requires an operator onboard	The system should be free of an operator, therefore making the unit smaller
	Main issues when choosing a product-ongoing and operation costs (both financial and time) then quality of data	The system should be inexpensive and time efficient in maintenance and operation
Instrumentation	I need hydrographic data (using an echo sounder)	System must support hydrographic instrumentation
	Currently, I collect data once every five years	System should operate at least once every five years
	I would like to collect data for three weeks, once per year	System optimally operates three weeks out of the year
	Data collected from a boat with a pile sticking the sounder underwater to measure the distance to the riverbed	System will have onboard instruments that are submerged just below the surface
	I want to take measurements between 10m-40m	Instruments will take measurements 10m-40m deep
	I want to take echo sounder data	Echo sounder is a required instrument
	it needs to stop at each location to take measurements	System will stop at each location to take data
Conditions	There are possibly fish in the rivers in question	Safety precautions must be taken for marine life
	I don't know of any nesting areas	Design for avoidance of possible nesting areas
	Little to no boating traffic	System must be visible to other boaters
	Don't know what kind of safety precautions have been taken in the past	Research safety precautions and boating practices in the area
	The collection needs to be taken in front of five dams in a 3 month window	System must be operational in the 3 month window
	No strong currents or waves	System will not undergo strong currents or waves
Interface	I prefer a graphical interface	System should utilize a graphical user interface
	I want to have an update at least every second. 5Hz would be optimal	System must have a real-time update with minimal delay
	The mission does not need to be adjusted in real time	The system does not need a manual override
Storage/	The area does not have a dock	System must incorporate a docking mechanism
Maintenance	The unit will be maintained in the SCU machine shop or the UCA machine shop	System has minimal maintenance equipment
	The unit will either be stored in the SCU trailer or the UCA trailer	System must fit in trailer
	Battery power will be available for recharging	System will be recharged by battery

Appendix C-4.
Response from JenniferReynolds and Raymond Highsmith from UAF
November 10, 2004

Question/Prompt	Customer Statement	Interpreted Need
Typical Use	We're concerned that a flat design will easily flip over in the winds	The system must be stable in gusting winds
	The boat is gonna be shipped to Anchorage in a 737 jet	The system must fit in a 737 cargo space
	It's a 4.5 hour drive from Anchorage to Homer	The system must be transportable in/on a trailer
	Need to take a boat from Homer to K-Bay	The system must be transportable in a boat
Instruments	Altimeter on board	The system must support an altimeter onboard
	We want to send the instruments without hitting the bottom	Instrument deployment must have a safety feature to prevent damage to instruments
	A 100 meter cable will cover most of the bay	Maximum depth of measurements will be 100m
	There are car parts stores in Homer	Off the shelf parts are available in Homer
	There is a marine store in Homer	Off the shelf parts are available in Homer
Conditions	The surface currents are different from the underwater currents	The system must be able to manuever and stationkeep in various currents
	The system will operate in salt water	The system must be corrosion resistant
	There are strong winds in the bay	The system must be stable in and be able to manuever through gusting winds
	There is a sandy beach in front of the lab	The system may dock using the beach area
Storage/Maintenance	There are launch and recover skiffs next to the dock	The system may dock using the skiffs next to the dock
	They are building a new 240ft dock on pilings with a boat lift	The system may dock using the boat lift on the new dock
	The dock is wide enough for a 4 wheeler	The system may be loaded onto a 4 wheeler for docking
	The dock has 4ft railings	Investigate the use of the dock for docking
	Will probably be stored in a shed when its not being used for over a week	The system must be storable in an outdoor shed

Appendix C-5.
Response from Ted Otis

Environment

Will there be marine animals in the sampling area? If so what types?

Yes. The following marine mammals seasonally occur in Kachemak Bay: sea otters, harbor seals, steller sea lions, harbor porpoises, Orcas, Humpback whales, gray whales, Minke whales. Many different species of sea ducks, marine birds, and, of course marine fish and shellfish occur in the area.

Are there any nesting areas?

Various sea ducks and marine birds (e.g., murres, kittewakes, cormorants, puffins), nest in the area. Gull Island is a key seabird nesting site. I believe sea ducks may nest in Sadie and Tutka Bays.

How much boat traffic is in the sampling area?

Kachemak Bay receives a considerable amount of vessel traffic ranging from small private craft <20' to large commercial fishing boats (40-80'), ferries (200') and industrial transport vessels (> 200').

What types of safety precautions have been taken previously?

I'm not sure what you're after here, but the larger vessels all stick to the main shipping" lanes. Smaller craft roam everywhere they're capable of navigating.

What are the weather conditions in the area? (Wave heights, wave characteristics, annual rainfall, typical wind conditions.etc)

Weather conditions can vary dramatically in Kachemak Bay. Frequent storms are common in winter, less so in summer. Afternoon "day-breezes" kick up most afternoons from May-September, consisting of 20+ knot westerly winds with 2-4' wind driven chop. Wave heights in the bay are generally < 3', however can reach 15+' during major storms. Waves are primarily wind driven chop, however larger ocean swells can enter the bay during major storms in the Gulf of Alaska. The prevailing wind directions that generate significant waves in Kachemak Bay are easterly (usually bringing precipitation) and westerly (using driven by high pressure systems). Precipitation is common throughout the year.

You can find photos of Kachemak Bay on the following URL. Good luck on your project.

http://www.habitat.adfg.state.ak.us/geninfo/kbrr/index.html (follow the links to images)

APPENDIX D – PRODUCT DESIGN SPECIFICATIONS

April 18 2005 REV E

Element/Requirement	Units	Datum	Target	Best
Performance				
Top Speed	knots	6	8	10
Battery Life	hrs	4	6	8
Precision	ft	16	3	1
Turning	ft	10	5	0
Depth (sensors)	ft	700	300	1000
Safety				
distance visible from	miles	1/2	2	3
$$$				
Product Cost	$	20000	15000	10000
Selling Price	$	50000	30000	25000
Customers	#	1	2	3
Processes	#	50	40	30
Quantity	#	1	1	5
Size	ft^3(LxWxH)	6x4x3	10x5x3	
Weight	lbs	200	1300	200
Max payload	lbs	100	200	
Materials	#	15	10	8
Shipping	ft^3	50	30	20
Envrionment-temp	°C	20	10	sub 0
Waves	ft	1	5	5
Instalation	hrs x ppl	20	15	10
Maintenance	hr/week	5	4	3
Life Span	Years	3	5	10
Life in service	hr/day	2	5	24
Disposal	ft^3	2	1	0
Asthetics	While it is important for the final product to look good the main emphasis will be on functionality and reliability			
quality/ reliability	The unit will measure all stations and return on 95% of the missions			
Storage	The unit will be stored on the dock out of the water to prevent corrosion			
testing	The low number of units will allow us to do a wide variety of tests and will insure the highest quality			
legal	Several legal issues are important when dealing with maritime law. However, at this time the legal status of unmanned vehicles is being decide, but certain issues that deal with safety will be followed as applicable.			
Documentation	An in depth users manual will address troubleshooting and maintenance of the Mechanical, electrical and User interface (computer)			

APPENDIX E – CONCEPT SCORING

ASV

updated 6/4

Prioritizing Matrix

Overall System

	Criteria	1	2	3	4	5	6	7	8	9	10	11	12	13	Total	Factor
1	Cost		1	0.5	0	0	0	0	0	0	0	0.5	0	0.5	2.5	5
2	Time	0		1	0	0	0	0	0	0	0	0.5	0	0.5	2	4
3	# Subsystems	0.5	0		0	0	0	0	0.5	0.5	0	0.5	0	0.5	2.5	5
4	Strength	1	1	1		0	0	0	0	0.5	0.5	1	0	1	6	12
5	Weight	1	1	1	1		0.5	0.5	1	0.5	1	0.5	0	1	9	18
6	Reliablity	1	1	1	1	0.5		1	1	1	1	1	0.5	1	11	22
7	Size	1	1	1	1	0.5	0		1	1	1	1	0	1	9.5	19
8	Integration	1	1	0.5	1	0	0	0		0.5	1	0	0	1	6	12
9	Rigidity	1	1	0.5	0.5	0.5	0	0	0.5		1	0.5	0	0.5	6	12
10	Manufacturability	1	1	1	0.5	0	0	0	0	0		0.5	0	0.5	4.5	9
11	Power Cons	0.5	0.5	0.5	0	0.5	0	0	1	5	0.5		0	0.5	9	18
12	Stability	1	1	1	1	1	0.5	1	1	1	1	1		1	11.5	23
13	Simplicity	0.5	0.5	0.5	0	0	0	0	0	0.5	0.5	0.5	0		3	6

	Ranking	Fact
1	Stability	23
2	Reliability	22
3	Size	19
4	Weight	18
5	Power Cons	18
6	Rigidity	12
7	Integration	12
8	Strength	12
9	Manufacturing	9
10	Simplicity	6
11	# Subsystems	5
	Cost	5
	Time	4

Autonomous Surface Vehicle
Selection Criteria Matrix

last updated 6/5

Overall Design

CRITERIA	Factor	AUV		ROV		ASV		SWATH ASV	
Time-Design	1	200		150		200		250	
Time-Build	1	1000		600		900		900	
Time-Test	1	50		50		50		100	
Time-Total		1250	1.0	800	1.6	1150	1.1	1250	1.0
Time Score	4		**4.0**		**6.3**		**4.3**		**4.0**
Cost	1	20000	1.0	15000	1.3	17000	1.2	15000	1.3
Cost Score	5		**5.0**		**6.7**		**5.9**		**6.7**
Stability	23	7	**161.0**	7	**161.0**	5	**115.0**	6	**138.0**
Reliability	22	5	**110.0**	8	**176.0**	7	**154.0**	7	**154.0**
Size	19	8	**152.0**	6	**114.0**	7	**133.0**	7	**133.0**
Weight	18	6	**108.0**	4	**72.0**	5	**90.0**	5	**90.0**
Power Consumption	18	7	**126.0**	6	**108.0**	7	**126.0**	7	**126.0**
Rigidity	12	8	**96.0**	7	**84.0**	5	**60.0**	7	**84.0**
Integration	12	5	**60.0**	5	**60.0**	5	**60.0**	5	**60.0**
Strength	12	6	**72.0**	5	**60.0**	5	**60.0**	6	**72.0**
Manufacturability	9	4	**36.0**	6	**54.0**	6	**54.0**	7	**63.0**
Simplicity	6	5	**45.0**	7	**63.0**	6	**54.0**	6	**54.0**
# Subsystems	5	4	**36.0**	6	**54.0**	5	**45.0**	5	**45.0**
Total			**1011**		**1019**		**961**		**1030**
Rank			**3**		**2**		**4**		**1**

ASV

updated 12/7

Prioritizing Matrix

Subsystem: Flotation

Criteria	1	2	3	4	5	6	7	8	9	10	11	Total	Factor
1 Cost	-	0.5	1	0	0	0	0	0	0	0	0	1.5	3
2 Time	0.5	-	1	0	0	0	0	0	1	0	0.5	3	6
3 # Parts	0	0	-	0	0	0	0.5	0	0	0	0	0.5	1
4 Strength	1	1	1	-	1	0.5	0.5	0	0.5	0	1	6.5	13
5 Weight	1	1	1	0	-	0.5	0.5	0	0.5	0	1	5.5	11
6 Reliablity	1	1	1	0.5	0.5	-	0.5	0.5	1	0.5	1	7.5	15
7 Size	1	1	0.5	0.5	0.5	0.5	-	0	1	0	1	6	12
8 Buoyancy	1	1	1	1	1	0.5	1	-	1	0.5	1	9	18
9 Rigidity	1	0	1	0.5	0.5	0	0	0	-	0	1	4	8
10 Stability	1	1	1	1	1	0.5	1	0.5	1	-	1	9	18
11 Simplicity	1	0.5	1	0	0	0	0	0	0	0	-	2.5	5

Ranking	Fact
1 Stability	18
2 Buoyancy	18
3 Reliability	15
4 Strength	13
5 Size	12
6 Weight	11
7 Rigidity	8
8 Time	6
9 Simplicity	5
10 Cost	3
11 # Parts	1

Autonomous Surface Vehicle
Selection Criteria Matrix

last updated 4/27

Flotation

CRITERIA	Factor	Inflatables		Fishing Boat		Foam		Semi-Submergible		PVC w endcapes	
Time-Design	1	1		2		5		10		10	
Time-Build	1	1		3		30		50		40	
Time-Test	1	5		1		2		5		5	
Time-Total		7	1.0	6	1.2	37	0.2	65	0.1	55	0.1
Time Score	6		**6.0**		**7.0**		**1.1**		**0.6**		**0.8**
Cost	1	800	1.0	800	1.0	500	1.6	600	1.3	400	2.0
Cost Score	3		**3.0**		**3.0**		**4.8**		**4.0**		**6.0**
Stability	18	5	**90.0**	8	**144.0**	8	**144.0**	5	**90.0**	8	**144.0**
Density	18	9	**162.0**	6	**108.0**	8	**144.0**	6	**108.0**	9	**162.0**
Reliability	15	5	**75.0**	8	**120.0**	6	**90.0**	5	**75.0**	8	**120.0**
Strength	13	4	**52.0**	9	**117.0**	6	**78.0**	7	**91.0**	7	**91.0**
Size	12	9	**108.0**	4	**48.0**	8	**96.0**	7	**84.0**	7	**84.0**
Weight	11	8	**88.0**	4	**44.0**	7	**77.0**	5	**55.0**	5	**55.0**
Rigidity	8	3	**24.0**	8	**64.0**	5	**40.0**	7	**56.0**	8	**64.0**
Simplicity	5	7	**35.0**	8	**40.0**	5	**25.0**	1	**5.0**	7	**35.0**
Parts	1	9	**9.0**	9	**9.0**	7	**7.0**	7	**7.0**	8	**8.0**
Total			**652**		**704**		**707**		**576**		**770**
Rank			**4**		**3**		**2**		**5**		**1**

ASV

updated 12/7

Prioritizing Matrix

Subsystem: Propulsion

Criteria	1	2	3	4	5	6	7	8	Total	Factor
1 Cost		0	0	0	0	0.5	0	0	0.5	1
2 Time	1		0	0.5	0	0.5	0.5	0.5	3	6
3 Thrust	1	1		1	1	0.5	0.5	1	6	12
4 Weight	1	0.5	0		0	1	0	1	3.5	7
5 Reliability	1	1	0	1		1	1	0.5	5.5	11
6 Mountability	0.5	0.5	0.5	0	0		0	0.5	2	4
7 Power Consumption	1	0.5	0.5	1	0	1		1	5	10
8 Simplicity	1	0.5	0	0	0.5	0.5	0		2.5	5

Ranking		Fact
1	Thrust	12
2	Reliability	11
3	Power Consumption	10
4	Weight	7
5	Time	6
6	Simplicity	5
7	Mountability	4
8	Cost	1

Autonomous Surface Vehicle
Selection Criteria Matrix

last updated 4/27

Propulsion

CRITERIA	Factor	Pontoon mounted electric motors (2)		Structure mounted gasoline motor		Structure mounted electric motors (2)	
Time-Design	1	2		2		4	
Time--Build	1	6		2		4	
Time-Test	1	10		10		10	
Time-Total		18	0.8	14	1.0	18	0.8
Time Score	6		**4.7**		**6.0**		**4.7**
Cost	1	1600	2	600	0.8	1600	2
Cost Score	1		**2**		**0.8**		**2.0**
Thrust	12	9	**108**	8	**96**	9	**108**
Reliability	11	8	**88**	9	**99**	8	**88**
Power Consumption	10	7	**70**	7	**70**	6	**60**
Weight	7	8	**56**	6	**42**	7	**49**
Simplicity	5	8	**40**	8	**40**	9	**45**
Mountability	4	7	**28**	8	**32**	8	**32**
Total			**397**		**386**		**389**
Rank			**1**		**3**		**2**

ASV

updated 12/7

Prioritizing Matrix

Subsystem: Power

Criteria	1	2	3	4	5	6	7	8	Total	Factor
1 Operating Life	-	0.5	1	1	0.5	0.5	0	1	4.5	9
2 Cost	0.5	-	1	1	0	0	0.5	0	3	6
3 Time	0	0	-	0	0	0	0	1	1	2
4 #Parts	0	0	1	-	0	0	0.5	0	1.5	3
5 Weight	0.5	1	1	1	-	0.5	1	0.5	5.5	11
6 Size	0.5	1	1	1	0.5	-	1	0.5	5.5	11
7 Simplicity	1	0.5	1	0.5	0	0	-	0	3	6
8 Availability	0	1	0	1	0.5	0.5	1	-	4	8

Ranking	Fact
1 Weight	11
2 Size	11
3 Operating Life	9
4 Availability	8
5 Simplicity	6
6 Cost	6
7 #Parts	3
8 Time	2

Autonomous Surface Vehicle
Selection Criteria Matrix

Power

CRITERIA	Factor	Batteries		Solar		Hybrid	
Time-Design	1	1		2		4	
Time-Build	1	3		5		5	
Time-Test	1	2		2		2	
Time-Total		6	1.0	9	0.7	11	0.5
Time Score	2		**2.0**		**1.3**		**1.1**
Cost	1	500	1.0	10000	0.1	1200	0.4
Cost Score	6		**6.0**		**0.3**		**2.5**
Weight	18	3	**54.0**	4	**72.0**	9	**162.0**
Size	18	6	**108.0**	2	**36.0**	8	**144.0**
Operating Life	15	7	**105.0**	7	**105.0**	9	**135.0**
Availability	13	9	**117.0**	4	**52.0**	9	**117.0**
Simplicity	12	10	**120.0**	8	**96.0**	8	**96.0**
#Parts	8	10	**80.0**	8	**64.0**	8	**64.0**
Total			**592**		**427**		**722**
Rank			**3**		**4**		**1**

last updated 4/27

Gasoline	
4	
5	
2	
11	0.5
	1.1
2000	0.3
	1.5
9	**162.0**
8	**144.0**
9	**135.0**
8	**104.0**
7	**84.0**
7	**56.0**
	688
	2

ASV

updated 12/7

Prioritizing Matrix

Subsystem: Vertical Deployment

Criteria	1	2	3	4	5	6	7	Total	Factor
1 Time	-	1	0	0	1	0.5	1	3.5	7
2 Cost	0	-	0	0	0	1	1	2	4
3 Weight	1	1	-	1	1	1	1	6	12
4 Size	1	1	0	-	0	1	1	4	8
5 Strength	0	1	0	1	-	0	1	3	6
6 Simplicity	0.5	0	0	0	1	-	0.5	2	4
7 Maximum S	0	0	0	0	0	0.5	-	0.5	1

Ranking	Fact
1 Weight	12
2 Size	8
3 Time	7
4 Strength	6
5 Simplicity	4
6 Cost	4
7 Maximum S	1

Autonomous Surface Vehicle
Selection Criteria Matrix

last updated 4/27

Vertical Deployment

CRITERIA	Factor	Cable Handling System		Telescope		ROV	
Time-Design	1	5		5		20	
Time-Build	1	15		10		50	
Time-Test	1	2		5		10	
Time-Total		22	1.0	20	1.1	80	0.3
Time Score	7		**7.0**		**7.7**		**1.9**
Cost	1	1000	1.0	200	5.0	2000	0.5
Cost Score	4		**4.0**		**20.0**		**2.0**
Weight	18	6	**108.0**	7	**126.0**	5	**90.0**
Size	18	7	**126.0**	7	**126.0**	4	**72.0**
Strength	15	9	**135.0**	3	**45.0**	6	**90.0**
Simplicity	13	6	**78.0**	7	**91.0**	2	**26.0**
Maximum Speed	12	9	**108.0**	6	**72.0**	5	**60.0**
Total			**566**		**488**		**342**
Rank			**1**		**3**		**4**

ASV updated 12/7

Prioritizing Matrix

Subsystem: Micorcontrollers

Criteria		1	2	3	4	Total	Factor
1	Cost		0.5	0	0	0.5	1
2	Capabilities	0.5		0.5	0.5	1.5	3
3	Tech Support	1	0.5		0.5	2	4
4	Familiarity	1	0.5	0.5		2	4

Ranking		Factor
1	Cost	1
2	Capabilites	3
3	Tech Support	4
4	Familiarity	4

Autonomous Surface Vehicle
Selection Criteria Matrix

last updated 12/7

Microcontrollers

CRITERIA	Factor	Basic-X		Atmel ATMega		OKI	
Cost	1	3	**3.0**	7	**7.0**	10	**10.0**
Capabilites	3	2	**6.0**	8	**24.0**	10	**30.0**
Tech Support	4	1	**4.0**	6	**24.0**	10	**40.0**
Familiarity	4	4	**16.0**	8	**32.0**	3	**12.0**
Total			**29**		**87**		**92**
Rank			**3**		**2**		**1**

APPENDIX F – CALCULATIONS

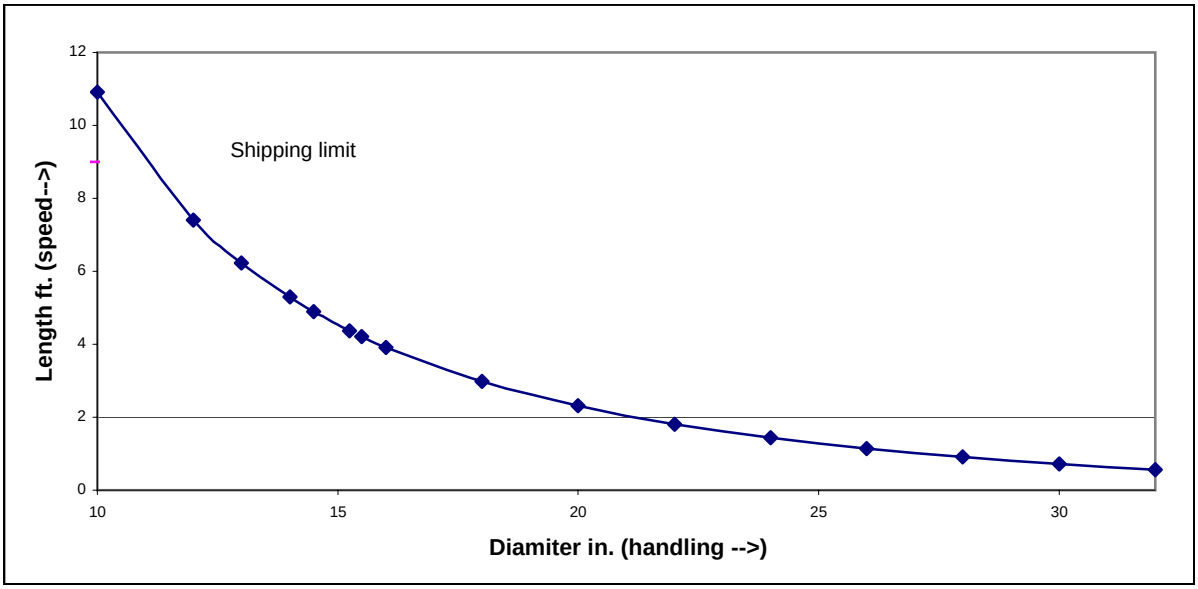
Weight Budget
originated 10/19
Rev -D 6/6

Item	**Quantity**	**Weight (lbs)**		Confidence
		Lbs in air	Lbs in wtr	
Pontoons (net)	2	60	20	spec
Structure	1	45	*na*	spec
Thruster	2	11	*na*	spec
Winch	1	45	*na*	spec
Tether	100ft	7	2	
	200m	45	13	
Batteries	4	113	*na*	spec
Generator 1000W	1	29	*na*	spec
Generator 2000W	1	46.3	*na*	spec
Charger	1	25	*na*	spec
GPS	1	2	*na*	spec
Wireless	1	2	*na*	spec
Compass	1	1	*na*	spec
Processor	1	2	*na*	spec
Radio Modem	1	5	*na*	spec
On-board Weather Instruments	1	5	*na*	spec
Sensors				
Dissolved Oxygen	1	1.5	1.25	spec
Fluorometer	1	0.9	0.75	spec
CTD	1	37	26	spec
ADCP	1	5	3.5	spec

Total	on trailer (*w/o generator*)	***816***
	in water (*w/o generator*)	***736***
	in water (*w/ 2000W generator*)	***782***

Pontoon & Ballast Sizing
originated 10/19
Rev -E 6/7

Diamiter	volume/ linear ft	lb disp/ linear ft	end cap volume	lb disp in end cap	min length	length/ side	Max length (shipping)	Total craft
in	ft^3/ft	lbs/ft	ft^3	lbs	ft	ft	ft	lbs
10	0.55	35	0.6	38.79	21.8	10.9	9	***800***
12	0.79	50	0.9	55.85	14.8	7.4	9	^
13	0.92	59	1.0	65.55	12.5	6.2	9	
14	1.07	68	1.2	76.02	10.6	5.3	9	
14.5	**1.15**	**73**	1.3	**81.55**	**9.8**	**4.9**	9	**Density**
15.25	**1.27**	**81**	1.4	**90.20**	**8.7**	**4.4**	9	lb/ft^3
15.5	**1.31**	**84**	1.5	**93.18**	**8.4**	**4.2**	9	***64***
16	1.40	89	1.6	99.29	7.8	3.9	9	^
18	1.77	113	2.0	125.66	6.0	3.0	9	
20	2.18	140	2.4	155.14	4.6	2.3	9	
22	2.64	169	2.9	187.72	3.6	1.8	9	**length of**
24	3.14	201	3.5	223.40	2.9	1.4	9	**end cap**
26	3.69	236	4.1	262.19	2.3	1.1	9	in
28	4.28	274	4.8	304.08	1.8	0.9	9	**10**
30	4.91	314	5.5	349.07	1.4	0.7	9	^
32	5.59	357	6.2	397.16	1.1	0.6	9	



length of area to flood	Pontoon Length	Total volume	Water Displaced	target loading @ length	loading range @ length	
in	*ft*	*ft^3*	*lb*	*lb*	*lb*	
8	***8***	3.06	196	800	max	*898*
^	^	*Gal*			min	*702*
		8	68	800	max	*834*
		^			min	*766*

Speed Calculations
originated 11/15/04
Rev -E 6/6/05

Standard Hull Vessel Calculations

volt	amp	watt	hp	thrusters	weight		sl ratio	WL	WL effet	MPH	Knots
12	**50**	600	0.80	**2**	**850**	528	1.5	**9**	4.50	6.75	5.87
12	**50**	600	0.80	**2**	**850**	528	1.5	**10**	4.74	7.12	6.19
12	**50**	600	0.80	**2**	**850**	528	1.5	**9**	4.50	6.75	5.87
12	**50**	600	0.80	**2**	**850**	528	1.5	**10**	4.74	7.12	6.19

Submerged Craft Calculation (Better approximation)

	in				out		
	Thrust (lbs.)	**water**	**drag coef**	**area**	**ft/s**	**knots**	**mph**
pontoon	**55**	**0.995**	**0.25**	**1.31**	**12.99**	**7.70**	**8.86**
pontoon + strut	**55**	**0.995**	**0.3**	**1.55**	**10.89**	**6.45**	**7.43**

Power Calculations

originated 11/15/04

Rev -E 6/6/05

Battery Selection

	Ah	# / unit	# / 12 v	Ah/# @12V	$/unit	$/Ah @12V
6V	370	113	226	1.637	210	1.14
12V	245	158	158	1.551	430	1.76
12V	200	129	129	1.550	380	1.90

Aditional Generator Power

Total hours	hours added by generator	amp hours added
5.9	0.0	237
8.3	2.4	426
10.2	4.3	578
11.7	5.8	699
12.9	7.0	796
13.9	8.0	874
14.7	8.7	936
15.3	9.4	985
15.8	9.9	1025
16.2	10.3	1057
16.5	10.6	1082
16.7	10.8	1103
16.9	11.0	1119
17.1	11.2	1132
17.2	11.3	1142
17.3	11.4	1151
17.4	11.5	1157

Ballast Pump Calculations

originated 11/15/04
Rev -E 6/6/05

Pump Volume Selection

GPH	GPM	lb/m	lb/m *(x4 pumps)*	time to adjust to weight change (s)		Cost	
				100 lbs.	200 lbs.	1 pump	4 pumps
300	5	45	180	33.3	66.7	$15.46	$61.84
500	8.3	75	300	20.0	40.0	$20.49	$81.96
800	13.3	120	480	12.5	25.0	$31.99	$127.96
1000	16.7	150	600	10.0	20.0	$36.46	$145.84
1100	18.3	165	660	9.1	18.2	$39.99	$159.96
1500	25.0	225	900	6.7	13.3	$64.99	$259.96
2000	33.3	300	1200	5.0	10.0	$89.99	$359.96

Winch Motor Calculations
originated 11/15/04
Rev -E 6/6/05

Winch Size Selection

Model	**1700**	**3700**		**1700**	**3700**
Cost	*$159.99*	*$349.99*		*$159.99*	*$349.99*
	MAX			**AVG**	
drum	2.0	2.8		2.0	2.8
reel	11.0	11.0		15.0	15.0
ratio	5.5	4.0		7.5	5.5
winch speed (no load)	4.5	6.2		4.5	6.2
winch speed (50lbs*ratio <500lbs.)	3.2	4.8		3.2	4.8
speed (max)	24.8	24.8		33.8	33.8
Speed 50lbs)	17.6	19.2		24.0	26.2
time down (min)	8.1	8.1		5.9	5.9
time up (min)	11.4	10.4		8.3	7.6
Time (per Station)	**19.4**	**18.5**		**14.3**	**13.6**
amps (station avg)	33.0	42.0		50.0	53.0
amp hours (per Station)	**10.7**	**12.9**		**11.9**	**12.0**

ASV Test plan
April 18 2005

Evaluation	Location/Time	Equipment	Accuracy	Trials	Projected Outcome	Assumptions	people hrs
Performance							
Top Speed	Leavey/TBA	stop watch, tape measure	.5 knots	5	8 Knots	distance/time constant velocity	0.5
Battery Life	Leavey/TBA	stop watch	.5 hrs	2	6 Hrs	average power draw	0.5
Precision	Leavey/TBA	tape measure	.5 ft	5	3 ft	center to center distance while stationkeeping	0.5
Turning	Leavey/TBA	tape measure	.5 ft	5	5 ft	center to center radius, low speed	0.5
Depth (sensors)	Shop/ April 25	tape measure	.1 ft	1	300 ft	really long tape	1
distance visible from	On campus/April 25	car, odometer	.01 mi	2	2 mi	clear day, no fog	1
Weight	shop/April 28	scale	5 lbs	1	800 lbs	weigh parts seperately	2
Max payload	shop/April 28	scale, tape measure	5 lbs	1	50 lbs	volume equals weight of water displaced	5
Waves	Monterey/Summer	wave tank, acelerometers	.5 ft	50	5 ft	to be done this summer	200
Assembly	Shop/May 4	stop watch	.5 hrs	1	15 Hrs	by very tired people at 1 am	15

APPENDIX G – MATERIAL PROPERTIES

MatWeb Data Sheet

Overview - High Density Polyethylene (HDPE), Ultra High Molecular Weight

KeyWords:
UHMWPE; Plastics, Polymers
SubCat: Polyethylene, Thermoplastic, HDPE, Polymer

Properties	Value	Min	Max	Comment
Physical				
Density, g/cc	--	0.934	0.954	Average = 0.945 g/cc; Grade Count = 5
Water Absorption, %	0.01	--	--	Grade Count = 1
Environmental Stress Crack Resistance, hour	500	--	--	Grade Count = 1
Melt Flow, g/10 min	--	2	12	Average = 8 g/10 min; Grade Count = 3
Mechanical				
Hardness, Shore D	--	63	68	Average = 65.2; Grade Count = 5
Tensile Strength, Ultimate, MPa	40	--	--	Grade Count = 1
Tensile Strength, Yield, MPa	--	20	27.6	Average = 23.4 MPa; Grade Count = 5
Elongation at Break, %	--	300	850	Average = 590%; Grade Count = 5
Tensile Modulus, GPa	--	0.5	0.85	Average = 0.68 GPa; Grade Count = 2
Flexural Modulus, GPa	--	0.7	1.45	Average = 0.996 GPa; Grade Count = 5
Izod Impact, Notched, J/cm	--	4	NB	Average = 9.5 J/cm; Grade Count = 5
Tensile Impact Strength, kJ/m²	--	710	1900	Average = 1300 kJ/m²; Grade Count = 2
Coefficient of Friction	0.15	--	--	Grade Count=1
Electrical				
Electrical Resistivity, ohm-cm	1.00E+16	--	--	Grade Count = 2
Surface Resistance, ohm	1.00E+16	--	--	Grade Count = 1
Dielectric Constant	2.3	--	--	Grade Count = 1
Dielectric Strength, kV/mm	28	--	--	Grade Count = 1
Dissipation Factor	0.0002	--	--	Grade Count = 1
Dissipation Factor, Low Frequency	0.0002	--	--	Grade Count = 1
Thermal				
CTE, linear 20°C, µm/m-°C	--	125	200	Average = 160 µm/m-°C; Grade Count=2
Melting Point, °C	130	--	--	Grade Count = 1
Maximum Service Temperature, Air, °C	--	67	82	Average = 74°C; Grade Count = 4
Deflection Temperature at 0.46 MPa (66 psi), °C	--	67	79	Average = 73.2°C; Grade Count=4
Deflection Temperature at 1.8 MPa (264 psi), °C	46	--	--	Grade Count=1
Vicat Softening Point, °C	--	129	139	Average = 130°C; Grade Count = 4
Minimum Service Temperature, Air, °C	-30	--	--	Grade Count = 1
Brittleness Temperature, °C	-75	--	--	Grade Count=3

Vendors
Matweb.com

HexWeb® ACG® Honeycomb

Aluminum Commercial Grade

Product Data

Description

HexWeb® Aluminum Commercial Grade honeycomb is made from 3000 series aluminum alloy foil. An organic coating is applied to the foil which provides excellent protection against corrosive atmospheres. Five cell sizes and densities are available. The honeycomb is manufactured by bonding together sheets of aluminum foil which are expanded to form a cellular honeycomb configuration. The node bond adhesive is a thermosetting type, cured under heat and pressure with the honeycomb in the unexpanded or HOBE® (HOneycomb Before Expansion) condition. Slices are cut from the unexpanded material to specified thickness, or cell depth, and then expanded to final configuration.

The honeycomb can be supplied either expanded or in HOBE slices. Any sheet thickness between 0.125 and 20 inches can be provided. The aluminum substrate can be perforated so that all cells will be vented in a slice as thin as 0.187 inches. The perforations of the HexWeb® ACG foil allow outgassing of the adhesives used to bond panel facings. Nonperforated honeycomb is also available.

Features

- Low Cost
- High Structural Strength/Low Weight
- Corrosion Resistant

Applications

HexWeb® ACG is a commercial grade honeycomb core offering industrial designers, at relatively low cost, the advantages of an all-metal honeycomb with long service life and resistance to fungus, moisture, and temperature. Uses include industrial tooling panels, architectural panels, shelving, storage tank covers, building walls, table and countertops.

Special Products/Custom Processing

HexWeb® Aluminum Commercial Grade honeycomb can be provided machined or formed into various shapes. This can include edge chamfering, simple and complex taper cuts, and other special machined configurations. HexWeb® ACG can be bonded to a variety of facing materials to form flat sandwich panels. Contact plant for honeycomb that is required to meet energy absorption strength specifications.

Standard Dimensions

Hexcel's HexWeb® Aluminum Commercial Grade materials are available in the following standard size:

Unexpanded L (HOBE)	Expanded Dimensions	Nominal Sq. Ft. Per Sheet
66" ± 1"	48" + 2" – 0" L x 96" + 4" – 0" W	32

Type Designation

HexWeb® ACG honeycomb is designated as follows:

Material – Cell Size – Density – Perforated

Example:

ACG – 3/8 – 3.3P

Where:

ACG – designates Aluminum Commercial Grade honeycomb type

3/8 – is the cell size in inches

3.3 – is the nominal density in pounds per cubic foot

P – indicates cell walls are perforated

HEXCEL®

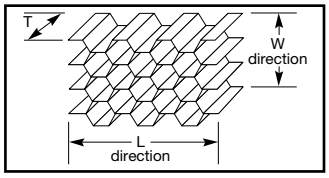
HexWeb® ACG® *Product Data*

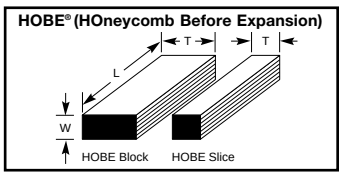
Dimensional Nomenclature

T = Thickness, or cell depth

L = Ribbon direction, or longitudinal direction

W = Transverse direction, or direction perpendicular to the ribbon





Availability

HexWeb® ACG honeycomb will be shipped F.O.B. Casa Grande, Arizona.

One of the major advantages of the HexWeb® ACG product line is the availability of a structural metallic honeycomb at low cost. The product is standard in one sheet size and is shipped untrimmed as expanded. Variations in T are available, while the L and W dimensions will be supplied in the standard expanded dimensions as shown. Custom L and W requirements or pieces cut to size can be supplied, but may carry a premium charge, depending on volume.

Thickness Tolerances

Tolerances on cut thickness are as shown at right:

Sheet Thickness	Standard Tolerance
0.125" to 3.999"	± 0.008"
4.000" and over	± 0.062"

Density Tolerance

The standard density tolerance of the nominal density is ± 17%.

Mechanical Properties

HexWeb® ACG honeycomb has been tested per MIL-STD-401. The following typical properties apply:

Hexcel Honeycomb Designation Material – Cell	Nominal Density pcf	Compressive			Crush Strength psi	Plate Shear			
		Bare	Stabilized			L Direction		W Direction	
		Strength psi	Strength psi	Modulus ksi		Strength psi	Modulus ksi	Strength psi	Modulus ksi
		typ	typ	typ	typ	typ	typ	typ	typ
ACG – 1/4	4.8	630	660	148	245	365	70	215	38
ACG – 3/8	3.3	340	370	92	120	230	45	130	22
ACG – 1/2	2.3	190	205	40	60	140	28	80	14
ACG – 3/4	1.8	120	130	24	45	100	20	65	11
ACG – 1	1.3	80	85	16p	25	65	14	45	7

Tested at 0.625 inch thickness.

Important

Hexcel Corporation believes, in good faith, that the technical data and other information provided herein is materially accurate as of the date this document is prepared. Hexcel reserves the right to modify such information at any time. The performance values in this data sheet are considered representative but do not and should not constitute specification minima. The only obligations of Hexcel, including warranties, if any, will be set forth in a contract signed by Hexcel or in Hexcel's then current standard Terms and Conditions of Sale as set forth on the back of Hexcel's Order Acknowledgement.

For further information, please contact your nearest sales office, or visit our website at **www.hexcel.com**

Australia
Suite 2, 86 Grimshaw Street
Greensborough, Victoria 3088
Tel: **61** 3 9432 7100
Fax: **61** 3 9432 7200

China
Room B707, Yin Hai Bldg.
250 Cao Xi Rd
Shanghai 200233
Tel: **86** 21 6483 6741/2
Fax: **86** 21 6483 6744

Japan - Joint Venture
DIC - Hexcel Limited
Room 603, Santsu-Mori Bldg.
2-22-1 Nishi - Shimbashi
Minato-Ku, Tokyo 105
Tel: **81** 3 5401 0271
Fax: **81** 3 5401 0270

USA
11711 Dublin Blvd.
Dublin, CA 94568-2832
Tel: **1** 925 551 4900
Fax: **1** 925 828 9202

USA
2350 Airport Freeway, Suite 5
Bedford, TX 76022-6027
Tel: **1** 817 315 3939
Fax: **1** 817 571 8629

Austria
Industriestrasse 1
A-4061, Pasching
Tel: **43** (0)7229 7720
Fax: **43** (0)7229 772299

France
Zl La Plaine, B.P.27 Dagneux
01121 Montluel CEDEX
Tel: **33** (0)4 72 25 26 27
Fax: **33** (0)4 72 25 27 30

Spain
Bruselas, 10 - 16
Polig. Ind. "Ciudad de Parla"
28980 Parla, Madrid
Tel: **34** 91 664 4900
Fax: **34** 91 698 4914

USA
900 Main Street South
Building 1, Suite 104
Southbury, CT 06488
Tel: **1** 203 267 1414
Fax: **1** 203 267 1561

USA
PO Box 18748
Salt Lake City, UT 84118-074
Tel: **1** 800 987 0658
Fax: **1** 801 508 8103

Belgium
Rue Trois Bourdons, 54
B-4840 Welkenraedt
Tel: **32** 87 307 411
Fax: **32** 87 882 895

Germany
Postfach 1560
21655 Stade
Tel: **49** 4141 7879-00
Fax: **49** 4141 7879-01

United Kingdom
Duxford, Cambridge
CB2 4QD
Tel: **44** (0)1223 833141
Fax: **44** (0)1223 838808

USA
42705 Grand River, Suite 201
Novi, MI 48375
Tel: **1** 248 344 8688
Fax: **1** 248 305 9760

USA
16310 NE 80th Street, Suite 1
Redmond, WA 98052
Tel: **1** 425 558 4400
Fax: **1** 425 861 5847

Brazil
Av. J. Guilhermino, 474/72
S.J.Campos, SP 12210-130
Tel: **55** 12 3941 2242
Fax: **55** 12 3923 1186

Italy
Via San Cristoforo, 44
21047 Saronno (VA)
Tel: **39** 02 96709082
Fax: **39** 02 9600809

The honeycomb sandwich construction is one of the most valued structural engineering innovations developed by the composites industry.

Used extensively in aerospace and many other industries, the honeycomb sandwich provides the following key benefits over conventional materials:

- Very low weight
- High stiffness
- Durability
- Production cost savings

Hexcel began developing honeycomb over 40 years ago, and now supplies a range of high performance honeycombs, prepregs and Redux® film adhesives - all ideally suited to the manufacture of honeycomb sandwich constructions. Hexcel is also the leading supplier of lightweight honeycomb sandwich panels.

This guide explains how to design and manufacture honeycomb sandwich panels, from materials selection and analysis of mechanical properties, through to production methods, and includes basic sample calculations for simple constructions.

More complex calculations may require computer modelling which, although mentioned briefly, is beyond the scope of this publication.

December 2000
Publication No. AGU 075b

Contents

BENEFITS OF HONEYCOMB SANDWICH CONSTRUCTIONS

The facing skins of a sandwich panel can be compared to the flanges of an I-beam, as they carry the bending stresses to which the beam is subjected. With one facing skin in compression, the other is in tension. Similarly the honeycomb core corresponds to the web of the I-beam. The core resists the shear loads, increases the stiffness of the structure by holding the facing skins apart, and improving on the I-beam, it gives continuous support to the flanges or facing skins to produce a uniformly stiffened panel. The core-to-skin adhesive rigidly joins the sandwich components and allows them to act as one unit with a high torsional and bending rigidity.

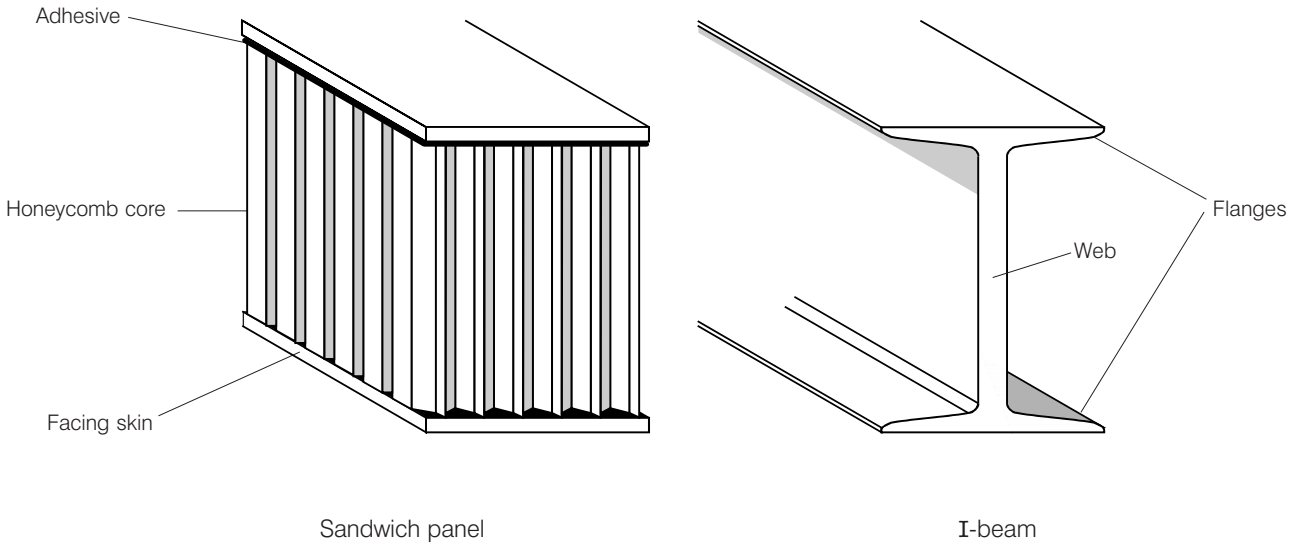


Figure 1 shows the construction of a sandwich panel compared to an I beam.

	Solid Material	Core Thickness t	Core Thickness 3t
	t	2t	4t
Stiffness	1.0	7.0	37.0
Flexural Strength	1.0	3.5	9.2
Weight	1.0	1.03	1.06

Figure 2 shows the relative stiffness and weight of sandwich panels compared to solid panels.

MATERIALS SELECTION

Honeycomb Sandwich Materials

The honeycomb sandwich construction can comprise an unlimited variety of materials and panel configurations. The composite structure provides great versatility as a wide range of core and facing material combinations can be selected. The following criteria should be considered in the routine selection of core, facing, and adhesive.

Structural Considerations

Strength:
Honeycomb cores and some facing materials are directional with regard to mechanical properties and care must be taken to ensure that the materials are orientated in the panel to take the best advantage of this attribute.

Stiffness:
Sandwich structures are frequently used to maximise stiffness at very low weights. Because of the relatively low shear modulus of most core materials, however, the deflection calculations must allow for shear deflection of the structure in addition to the bending deflections usually considered.

Adhesive Performance:
The adhesive must rigidly attach the facings to the core material in order for loads to be transmitted from one facing to the other. Suitable adhesives include high modulus, high strength materials available as liquids, pastes or dry films. As a general rule, a low peel-strength, or relatively brittle adhesive should never be used with very light sandwich structures which may be subjected to abuse or damage in storage, handling or service.

Economic Considerations:
Composite sandwich panels can provide a cost effective solution. Value analysis should include assessment of production and assembly costs; and installation costs including supporting structure.

Environmental Considerations

Temperature:
As in any materials system the thermal environment will play an important role in the selection of materials.
All systems are basically operational at Room Temperature and materials are readily available to give performance up from -55°C to 170°C.
Material selection should also take account of available manufacturing facilities, especially cure temperature capability.

Flammability:
Materials used in bonded sandwich construction are usually classified into three categories:

1) Non-burning - which means that the product will not burn.

2) Self-extinguishing - which means that the material will burn while held in a flame but will extinguish when the flame is removed.

3) Flammable. Flammable materials are sometimes further defined by determining the flame spread rate under specified conditions.

Heat Transfer:
The transfer of heat through a sandwich panel is dependent upon the basic principles of convection, conduction and radiation. Metallic cores with metallic facings maximise heat flow characteristics.

Moisture/Humidity:
Some core and facing materials offer excellent resistance to degradation due to moisture and humidity.

Adhesive Solvents and Outgassing:
Some adhesives give off gases or solvent vapours during cure which can interact with resin systems in some non-metallic cores, or with the node adhesive in some metallic honeycombs. The entire bonding process must be checked to ensure that no reduction in mechanical properties has occurred due to incompatibility of the materials or process actually used. All of Hexcel's Redux® film adhesives are compatible with this type of construction.

Honeycomb Materials

HexWeb honeycomb is available in a wide range of materials including:-

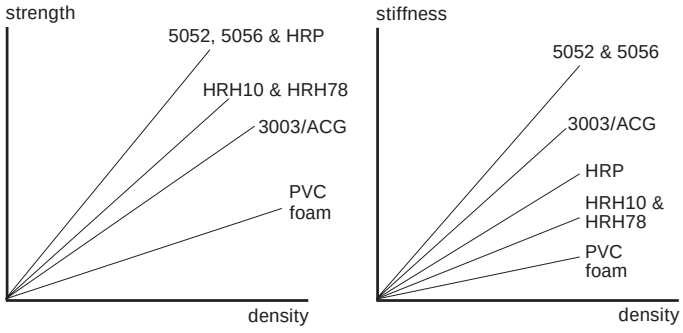
Aluminium, Nomex (Aramid), Korex, Kevlar, Fibreglass, Carbon.

For details please consult The HexWeb Honeycomb Selector Guide and the HexWeb Honeycomb Attributes and Properties Manual.

Selected mechanical properties for Aluminium and Nomex honeycombs are shown in Appendix I.

Mechanical Performance

Honeycomb strength and stiffness (compression and shear) is proportional to density. Relative performance of the material types is shown in comparison to PVC foam.



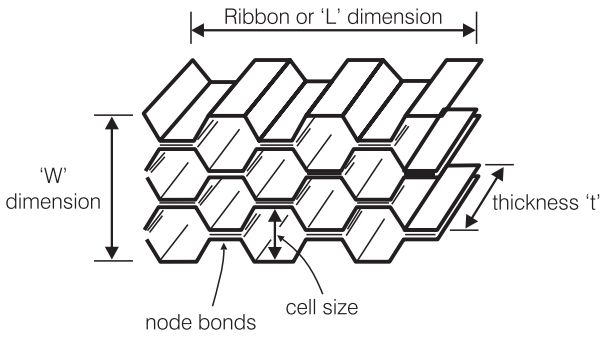
Key: Aluminium - 3003/ACG; 5052; 5056
Nomex - HRH10; HRH78;
Fibreglass - HRP

Cell Size

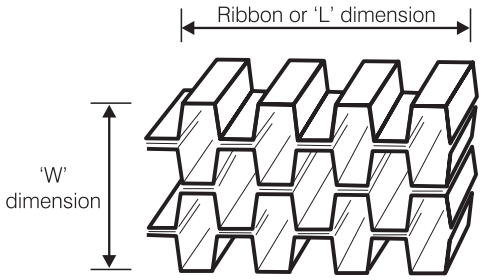
A large cell size is the lower cost option, but in combination with thin skins may result in telegraphing, i.e. a 'dimpled' outer surface of the sandwich. A small cell size will give an improved surface appearance, and provides a greater bonding area, but at higher cost.

Cell Shape

Normally supplied with hexagonal cell shapes, a few honeycomb types can be supplied with rectangular cell shapes (W:L approximately 2:1), and designated OX.



hexagonal cell size
= measured across flats



OX cell size
= nominal hexagonal cell size before expansion to rectangular shape.

Hexagonal cells give minimum density for a given amount of material.
Rectangular cells give easier forming in the W direction (with less anticlastic curvature than is exhibited by hexagonal cell honeycomb).

Skin Materials

The table in Appendix II shows properties of typical facing materials for sandwich panel construction.

Skin considerations include the weight targets, possible abuses and local (denting) loads, corrosion or decorative constraints, and costs.

Facing material thickness directly affects both the skin stress and panel deflection.

Hexcel Composites offers a wide range of prepreg materials. Refer to the Prepreg Matrix Selector Guide to identify systems most likely to suit your application, where fibre reinforced composites are thought appropriate.

Adhesive Materials

For honeycomb sandwich bonding, the following criteria are important:

1. Fillet Forming

To achieve a good attachment to an open cell core such as honeycomb, the adhesive should flow sufficiently to form a fillet without running away from the skin to core joint.

2. Bond Line Control

Every endeavour should be made to ensure intimate contact between the parts during bonding, as the adhesive needs to fill any gaps between the bonding surfaces.

Adhesives are often supplied supported by a carrier cloth, for the purpose of helping them to remain in place where the parts are squeezed particularly tightly together.

Hexcel Composites offers a wide range of film adhesives. Refer to the REDUX® Film Adhesive Selector Guide to identify the most suitable material for your application.

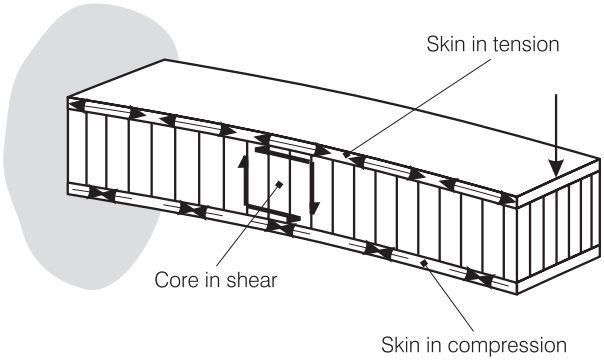
SANDWICH DESIGN

How a Sandwich Beam Works

Loads

Consider a cantilever beam with a load applied at the free end. The applied load creates a bending moment which is a maximum at the fixed end, and a shear force along the length of the beam.

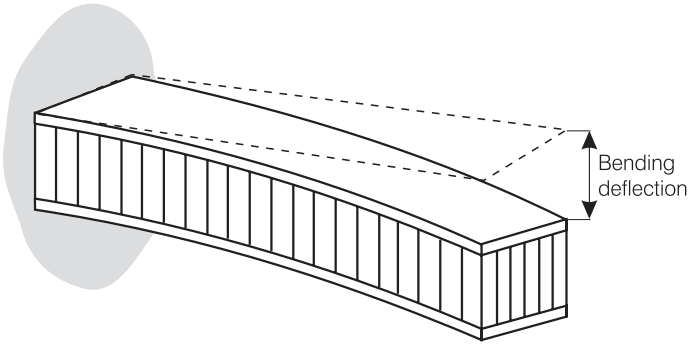
In a sandwich panel these forces create tension in the upper skin and compression in the lower skin. The core spaces the facing skins and transfers shear between them to make the composite panel work as a homogeneous structure.



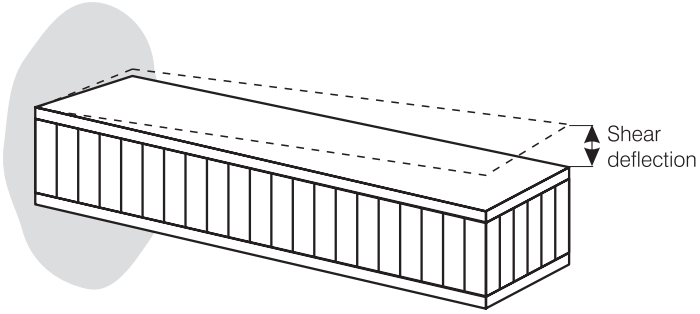
Deflections

The deflection of a sandwich panel is made up from bending and shear components.

The bending deflection is dependant on the relative tensile and compressive moduli of the skin materials.



The shear deflection is dependant on the shear modulus of the core.



Total Deflection = Bending Deflection + Shear Deflection.

Under different sets of applied loads and supporting conditions, the material stresses and deflections can be calculated as shown on page 9 onwards.

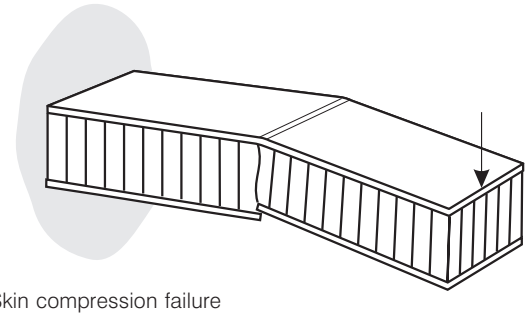
Failure modes

Designers of sandwich panels must ensure that all potential failure modes are considered in their analysis. A summary of the key failure modes is shown below:

1. Strength

The skin and core materials should be able to withstand the tensile, compressive and shear stresses induced by the design load.

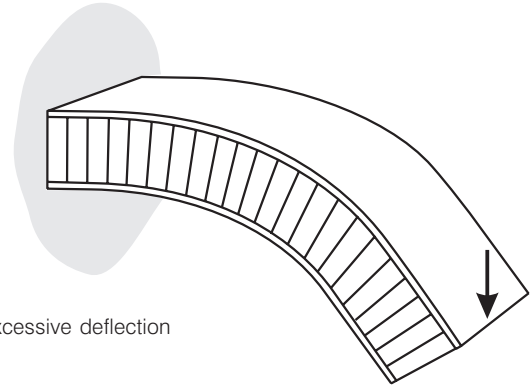
The skin to core adhesive must be capable of transferring the shear stresses between skin and core.



Skin compression failure

2. Stiffness

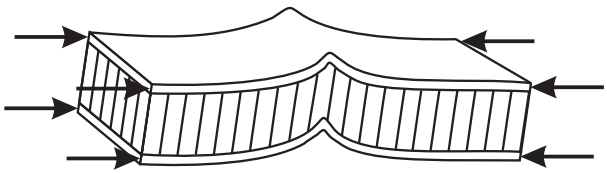
The sandwich panel should have sufficient bending and shear stiffness to prevent excessive deflection.



Excessive deflection

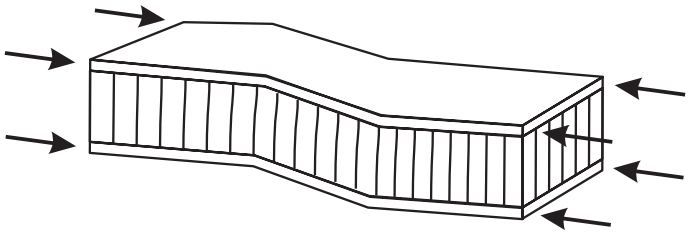
3. Panel buckling

The core thickness and shear modulus must be adequate to prevent the panel from buckling under end compression loads.



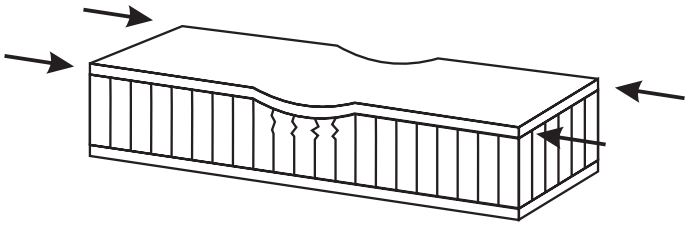
4. Shear crimping

The core thickness and shear modulus must be adequate to prevent the core from prematurely failing in shear under end compression loads.



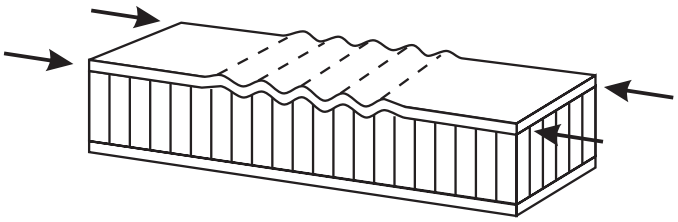
5. Skin wrinkling

The compressive modulus of the facing skin and the core compression strength must both be high enough to prevent a skin wrinkling failure.



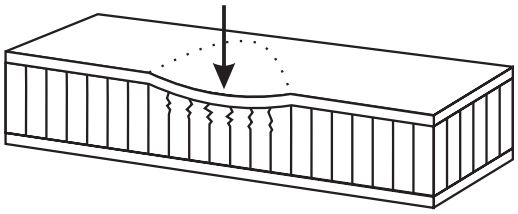
6. Intra cell buckling

For a given skin material, the core cell size must be small enough to prevent intra cell buckling.



7. Local compression

The core compressive strength must be adequate to resist local loads on the panel surface.



DESIGN GUIDELINES FOR A HONEYCOMB SANDWICH PANEL

1. Define loading conditions

e.g. Point loading, uniform distributed load, end loads.

Care should be taken to consider all possible loading conditions. For example, see table page 11, or refer to an appropriate 'Industry Standard' for guidance.

2. Define panel type

e.g. Cantilever, simply supported.

This is determined by the type and extent of the panel supports. Fully built in support conditions should only be considered when the supporting structure has adequate stiffness to resist deflection under the applied loads.

3. Define physical/space constraints

This should include an assessment of the requirements including:

- deflection limit
- thickness limit
- weight limit
- factor of safety

Preliminary materials selection should be based on the above criteria in conjunction with the features considered on pages 4 - 5 (and appendices I and II).

4. Preliminary calculations

- Make an assumption about skin material, skin thickness and panel thickness. Ignore the core material at this stage.
- Calculate stiffness.
- Calculate deflection (ignoring shear deflection).
- Calculate facing skin stress.
- Calculate core shear stress.

5. Optimise design

- Modify skin thickness, skin material and panel thickness to achieve acceptable performance.
- Select suitable core to withstand shear stress.

6. Detailed calculations

- Calculate **stiffness.**
- Calculate **deflection**, including shear deflection.
- Calculate **facing skin stress.**
- Calculate **core shear stress**.
- Check for **panel buckling** - where applicable
- Check for **shear crimping**.
- Check for **skin wrinkling**.
- Check for **intracell buckling**.
- Check for **local compression** loads on core.

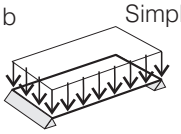
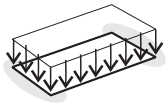
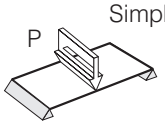
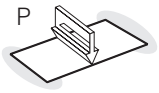
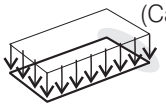
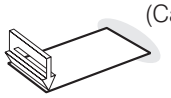
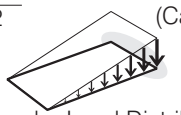
NB.
The formulae used for the sample problems that follow on pages 12 to 19, use simplified terms and give an order of magnitude appreciation.

See also Summary of Formulae in Appendix III.

NOMENCLATURE

- a = Panel length
- A = Area of applied load
- b = Beam width
- D = Panel bending stiffness
- E_C = Compression modulus of core
- E_f = Modulus of elasticity of facing skin
- F = Maximum shear force
- G_C = Core shear modulus - in direction of applied load
- G_L = Core shear modulus - Ribbon direction
- G_W = Core shear modulus - Transverse direction
- h = Distance between facing skin centres
- k_b = Beam - bending deflection coefficient
- k_S = Beam - shear deflection coefficient
- K_1 = Panel parameter (used for simply supported plate)
- K_2 = Panel parameter (used for simply supported plate)
- K_3 = Panel parameter (used for simply supported plate)
- l = Beam span
- M = Maximum bending moment
- P = Applied load
- P_b = Critical buckling load
- q = Uniformly distributed load
- R = Ratio G_L/G_W
- s = Cell size
- S = Panel shear stiffness
- t_C = Thickness of core
- t_f = Thickness of facing skin
- V = Panel parameter (used for simply supported plate)
- δ = Calculated deflection
- σ_C = Core compressive stress
- σ_{CR} = Critical facing skin stress
- σ_f = Calculated facing skin stress
- τ_C = Shear stress in core
- μ = Poissons Ratio of face material
- λ = Bending correction factor for Poissons Ratio effect

Summary of beam coefficients

BEAM TYPE	MAXIMUM SHEAR FORCE **F**	MAXIMUM BENDING MOMENT **M**	BENDING DEFLECTION COEFFICIENT k_b	SHEAR DEFLECTION COEFFICIENT k_s
$P = q\,l\,b$ Simple Support  Uniform Load Distribution	$\frac{P}{2}$	$\frac{Pl}{8}$	$\frac{5}{384}$	$\frac{1}{8}$
$P = q\,l\,b$ Both Ends Fixed  Uniform Load Distribution	$\frac{P}{2}$	$\frac{Pl}{12}$	$\frac{1}{384}$	$\frac{1}{8}$
Simple Support P  Central Load	$\frac{P}{2}$	$\frac{Pl}{4}$	$\frac{1}{48}$	$\frac{1}{4}$
Both Ends Fixed P  Central Load	$\frac{P}{2}$	$\frac{Pl}{8}$	$\frac{1}{192}$	$\frac{1}{4}$
$P = q\,l\,b$ One End Fixed (Cantilever)  Uniform Load Distribution	P	$\frac{Pl}{2}$	$\frac{1}{8}$	$\frac{1}{2}$
One End Fixed (Cantilever) P  Load One End	P	Pl	$\frac{1}{3}$	1
$P = \frac{q\,l\,b}{2}$ One End Fixed (Cantilever)  Triangular Load Distribution	P	$\frac{Pl}{3}$	$\frac{1}{15}$	$\frac{1}{3}$

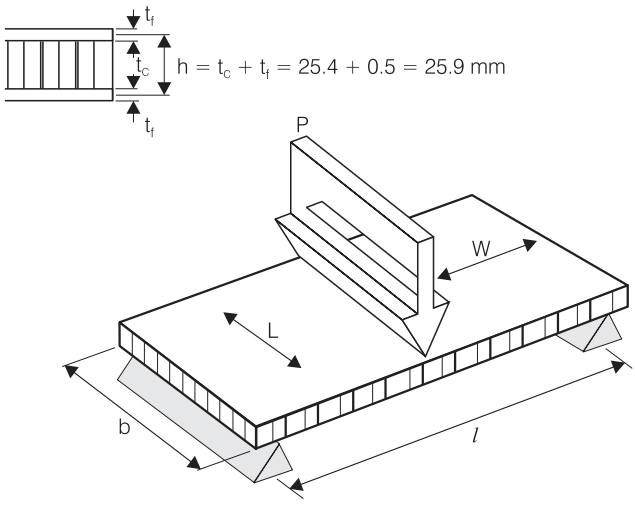
SAMPLE PROBLEMS BASED ON A STANDARD HEXLITE 220 PANEL

Configuration and Data:

Facing Skins	Aluminium 5251 H24
Thickness t_1 and t_2	= 0.50mm
and from Appendix II	
Yield Strength	= 150 MPa
E_f Modulus	= 70 GPa
Poissins Ratio μ	= 0.33
Core	5.2 - 1/4 - 3003
Thickness t_C	= 25.4 mm
and from Appendix I	
E_C Modulus	= 1000 MPa
Longitudinal shear	= 2.4 MPa
G_L Modulus	= 440 MPa
Transverse shear	= 1.5 MPa
G_W Modulus	= 220 MPa
Stabilized Compression	= 4.6 MPa

Simply Supported Beam

- taking a beam as being defined as having width (b) less than 1/3 of span (*l*)



Considering a centre point loaded beam
with b = 0.5m and l = 2m and P = 1500N

Beam

Bending Stiffness

$$D = \frac{E_f t_f h^2 b}{2}$$

Where $h = t_f + t_C$

Shear Stiffness

$$S = b h G_C$$

$$D = \frac{(70 \times 10^9)\ (0.5 \times 10^{-3})\ (25.9 \times 10^{-3})^2\ (0.5)}{2}$$

$$D = 5869.6\ Nm^2$$

As the core shear here will be taken by the weaker transverse direction - take $G_C = G_W$ shear modulus

$$S = (0.5)\ (25.9 \times 10^{-3})\ (220 \times 10^6)$$

$$S = 2849 \times 10^3\ N$$

Beam continued

Deflection

Bending plus Shear

$$\delta = \frac{k_b P l^3}{D} + \frac{k_s P l}{S}$$

Where k_b and k_s are deflection coefficients from page 11.

If doing preliminary calculations, just work out the bending deflection.

If optimising design, calculate for both bending and shear components (as shown opposite).

Bending plus Shear

$$\delta = \frac{1}{48} \times \frac{1500 \times 2^3}{5869.6} + \frac{1}{4} \times \frac{1500 \times 2}{2849 \times 10^3}$$

$$\delta = 0.04259\text{m} + 0.000263\text{m}$$

Total = approx 43mm

If excessive, then the most efficient way to reduce deflection is to increase core thickness, and thus increase the skin separation and the value of h.

Facing Stress

$$\sigma_f = \frac{M}{h\, t_f\, b}$$

Where M is Maximum Bending Moment expression from page 11

and $h = t_f + t_c$

$$M = \frac{Pl}{4} = \frac{1500 \times 2}{4} = 750\text{ Nm}$$

$$\sigma_f = \frac{750}{(25.9 \times 10^{-3})\ (0.5 \times 10^{-3})\ (0.5)}$$

$$\sigma_f = 115.8\text{ MPa}$$

So calculated stress is less than face material typical yield strength of 150 MPa, thus giving a factor of safety.

Core Stress

$$\tau_c = \frac{F}{hb}$$

Where F is Maximum Shear Force expression from page 11

$$F = \frac{P}{2} = \frac{1500}{2} = 750\text{N}$$

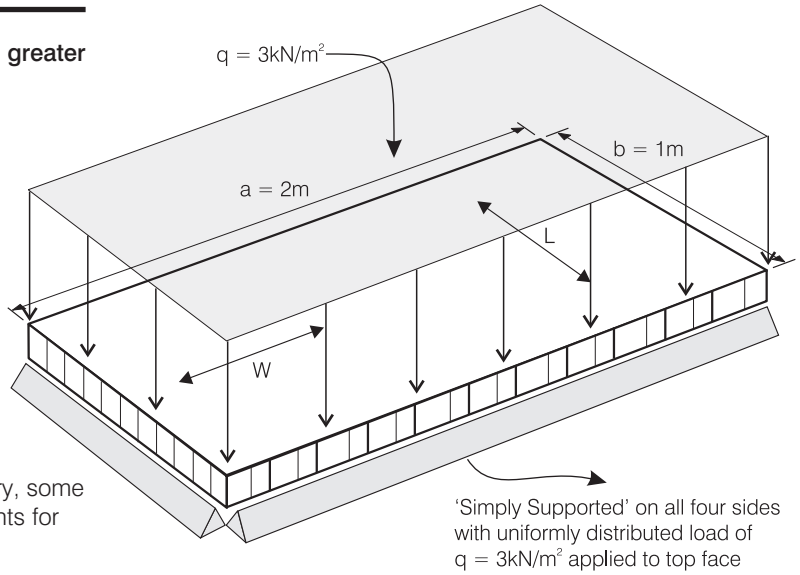
$$\tau = \frac{750}{(25.9 \times 10^{-3})\ (0.5)}$$

$$\tau = 0.06\text{ MPa}$$

So calculated shear is considerably less than core material typical plate shear in the transverse (W) direction of 1.5 MPa, giving a factor of safety, which could allow core density to be reduced.

Simply Supported Plate

- taking a plate as being defined as having width (b) greater than 1/3 of length (a)



Because plate theory is more involved than beam theory, some 'charts' have been provided to give multipliers/coefficients for use with plates simply supported on all four sides.

A further term (λ) is also introduced, to take account of the Poisson's Ratio of the face skin materials. For the plate set up shown, λ is taken as $1-\mu^2$.

NB. For the earlier beams, and the end load conditioning to follow, λ is assumed to be 1, as any affect from Poisson's Ration is small due to the relative narrowness of beams.

Plate

Determine Plate Coefficient

$\frac{b}{a}$

$$R = \frac{G_L}{G_W}$$

$$V = \frac{\pi^2 E t_f h}{2b^2 G_W \lambda}$$

From Fig.1 (page 16) $K_1 = 0.0107$
From Fig.2 (page 17) $K_2 = 0.102$

and for shorter span length b

From Fig.3 (page 17) $K_3 = 0.36$

Deflection

$$\delta = \frac{2K_1 q b^4 \lambda}{E_f t_f h^2}$$

$$\frac{1000}{2000} = 0.5$$

$$\frac{440}{220} = 2$$

Use R = 2.5 in Figs. on pages 16 & 17

$$V = \frac{\pi^2 (70 \times 10^9) (0.5 \times 10^{-3}) (25.9 \times 10^{-3})}{(2) (1^2) (220 \times 10^6) (1 - 0.33^2)} = 0.023$$

Take V = 0 in Figs. on pages 16 & 17

$$\delta = \frac{(2) (0.0107) (3 \times 10^3) (1^4) (1 - 0.33^2)}{(70 \times 10^9) (0.5 \times 10^{-3}) (25.9 \times 10^{-3})^2}$$

$\delta = 0.0024m = 2.4mm$

Plate continued

Facing Stress

$$\sigma_f = \frac{K_2 \, q \, b^2}{ht}$$

$$\sigma_f = \frac{(0.102)\ (3 \times 10^3)\ (1^2)}{(25.9 \times 10^{-3})\ (0.5 \times 10^{-3})}$$

$$\sigma_f = 23.6 \text{ MPa}$$

So calculated stress is considerably less than skin material typical yield strength of 150 MPa, thus giving a factor of safety.

Core Shear

$$\tau_C = \frac{K_3 \, q \, b}{h}$$

$$\tau_C = \frac{(0.36)\ (3 \times 10^3)\ (1)}{(25.9 \times 10^{-3})}$$

$$\tau_C = 0.042 \text{ MPa}$$

So calculated core shear is considerably less than typical core material shear value of 1.5 MPa, thus giving a factor of safety.

Local Compression

$$\sigma_C = \frac{P}{A} = \frac{q \times A}{A}$$

$$\sigma_C = \frac{(3 \times 10^3)\ (2 \times 1)}{(2 \times 1)}$$

$$\sigma_C = 0.003 \text{ MPa}$$

So local compression would not be an issue, being very small in comparison to typical core compression strength of 4.6 MPa.

Charts providing coefficients for plates simply supported on all four sides

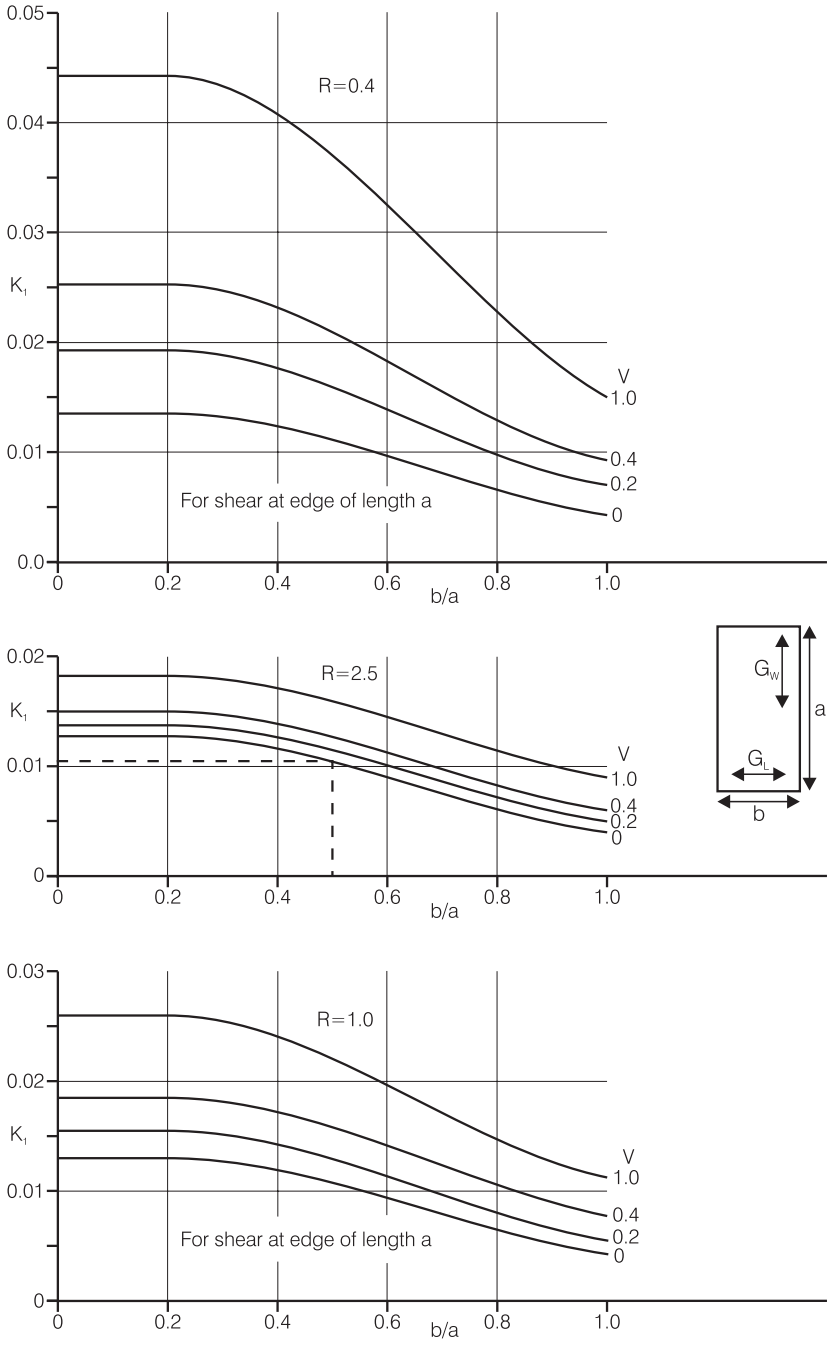


Figure 1 - K_1 for determining maximum deflection δ

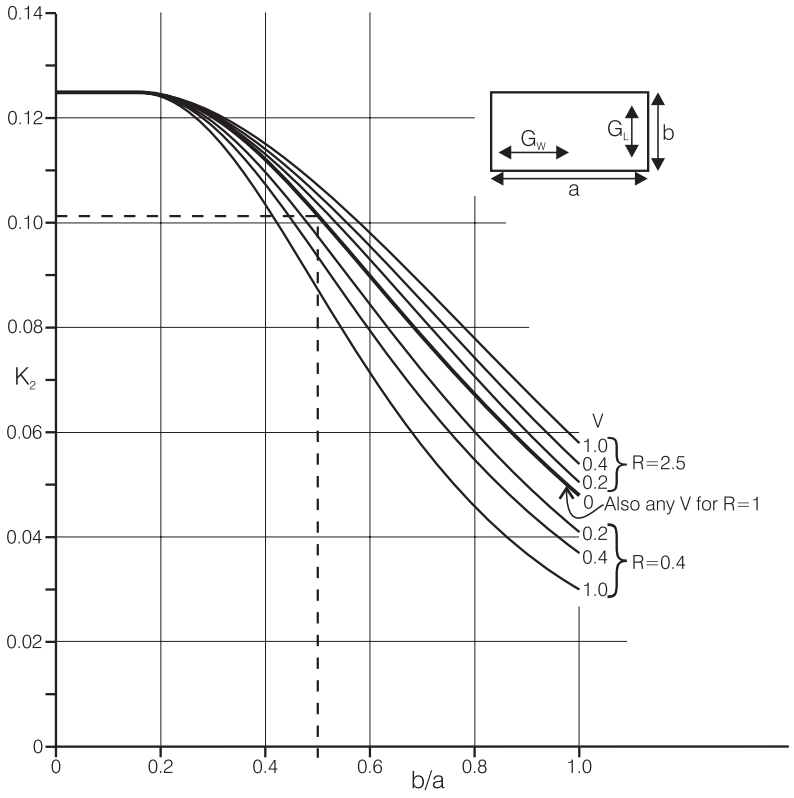


Figure 2 - K_2 for determining facing stress σ

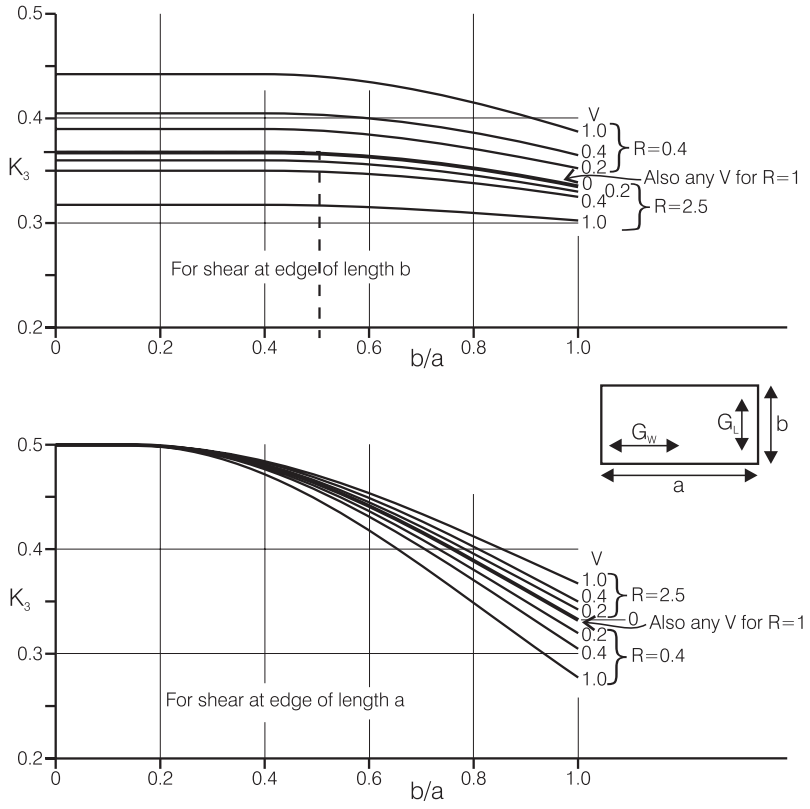
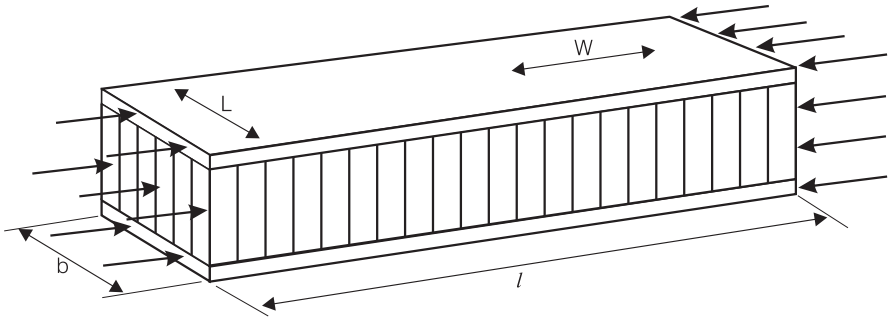


Figure 3 - K_3 for determining maximum core shear stress t_c

END LOAD CONDITIONS

Considering a uniformly distributed end load of
q = 20kN/m length and with b = 0.5m and l = 2m.



q = 20kN/m length

End Loading

Facing Stress

$$\sigma_f = \frac{P}{2\, t_f\, b}$$

assuming end load is taken by both skins, and applied load
P = q x b

Panel Buckling

$$P_b = \frac{\pi^2 D}{l^2 + \pi^2 D \over G_c\, h\, b}$$

Taking D from the beam calculation example.

$$\sigma_f = \frac{(20 \times 10^3)\ (0.5)}{(2)\ (0.5 \times 10^{-3})\ (0.5)}$$

$$\sigma_f = 20 \text{ MPa}$$

This is safe, as it is considerably less than skin material typical yield strength of 150 MPa.

Considering the core shear to be in the weaker transverse direction

So $G_c = G_w$ shear modulus

Then

$$P_b = \frac{\pi^2\ (5869.6)}{(2)^2 + \pi^2\ (5869.6) \over (220 \times 10^6)\ (25.9 \times 10^{-3})\ (0.5)}$$

$$P_b = 14{,}413 \text{ N}$$

So calculated load at which critical buckling would occur is greater than the end load being applied (P) of 10,000 N, thus giving a factor of safety.

End Loading continued

Shear Crimping

$P_b = t_c G_c b$

Taking G_C as G_W

$P_b = (25.4 \times 10^{-3}) (220 \times 10^6) (0.5)$

$P_b = 2.79$ MN

So the calculated load at which shear crimping would occur, is considerably greater than the end load being applied (P) of 10,000 N, thus giving a factor of safety.

Skin Wrinkling

$$\sigma_{CR} = 0.5 [G_C E_C E_f]^{1/3}$$

Taking G_C as G_W

$$\sigma_{CR} = 0.5 [(220 \times 10^6) (1000 \times 10^6) (70 \times 10^9)]^{1/3}$$

$$\sigma_{CR} = 1244 \text{ MPa}$$

So the stress level at which skin wrinkling would occur, is well beyond the skin material typical yield strength of 150 MPa; so skin stress is more critical than skin wrinkling.

Intracell Buckling

$$\sigma_{CR} = 2 E_f \left[\frac{t_f}{s}\right]^2$$

NB: s = cell size

$$\sigma_{CR} = 2 (70 \times 10^9) \left[\frac{(0.5 \times 10^{-3})}{(6.4 \times 10^{-3})}\right]^2$$

$$\sigma_{CR} = 854 \text{ MPa}$$

So stress level at which intracell buckling would occur is well beyond the skin material typical yield strength of 150 MPa; so skin stress is more critical than intracell buckling.

COMPUTER MODELLING OF HONEYCOMB SANDWICH PANELS

For a more sophisticated analysis of a structure, considering the sandwich panel to be subjected to a combination of forces, a technique such as Finite Element Analysis (FEA) might be used.

In general terms, the shear forces normal to the panel will be carried by the honeycomb core. Bending moments and in-plane forces on the panel will be carried as membrane forces in the facing skins.

For many practical cases, where the span of the panel is large compared to its thickness, the shear deflection will be negligible. In these cases, it may be possible to obtain reasonable results by modelling the structure using composite shell elements. It should be noted that the in-plane stiffness of the honeycomb is negligible compared to that of the facing skins.

Where a more detailed model is required it is possible to model the honeycomb core using solid 3D elements. Attempts to model the individual cells of the honeycomb should be avoided for normal engineering analyses.

When defining the properties of honeycomb core the following points should be taken into consideration:-

$E_X \approx E_Y \approx 0$

A very small value may be necessary to avoid singularity.

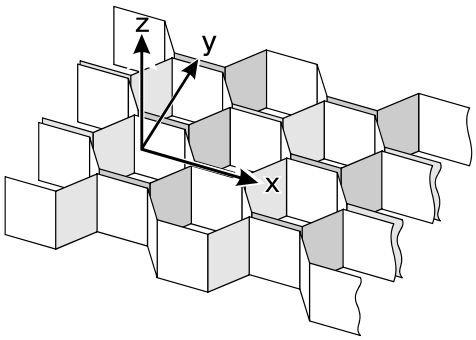
$\mu_{xy} \approx \mu_{xz} \approx \mu_{yz} \approx 0$

$G_{xy} \approx 0$

$G_{xz} = G_L$ = shear modulus in ribbon direction

$G_{yz} = G_W$ = shear modulus in transverse direction

$E_Z = E_C$ = compressive modulus of core material



Before analysing a large structure the modelling technique should be checked by modelling a simple panel with known results.

The above simplistic approach has proven to give reasonable engineering solutions for practical applications.

The actual force/stress distribution within a honeycomb sandwich structure is a complex subject, and is beyond the scope of this publication.

MANUFACTURE

Basic Honeycomb Sandwich Production Methods

Honeycomb sandwich components may be produced using three alternative well-established methods:-

Heated Press, generally used for the production of flat board or simple preformed panels.

Vacuum Bag Processing, used for curved and complex form panels.

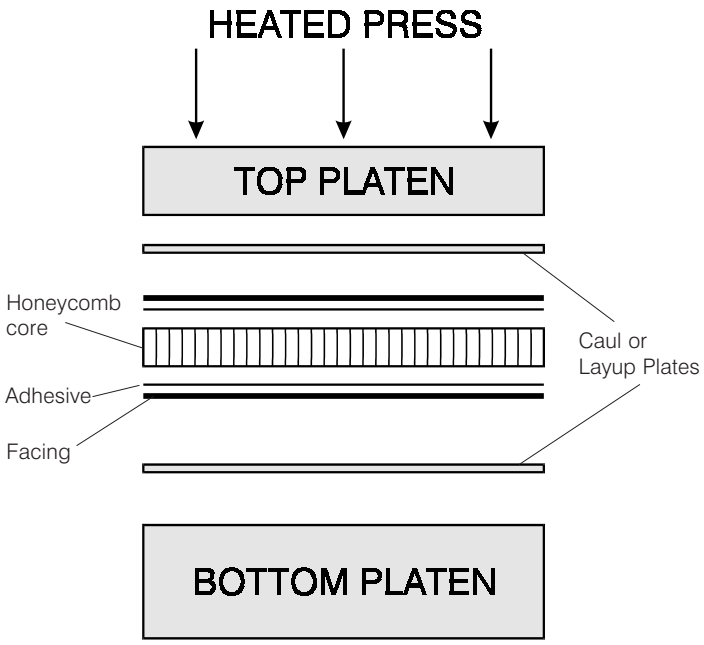
Matched Mould Processing, used generally for batch production of finished panels.

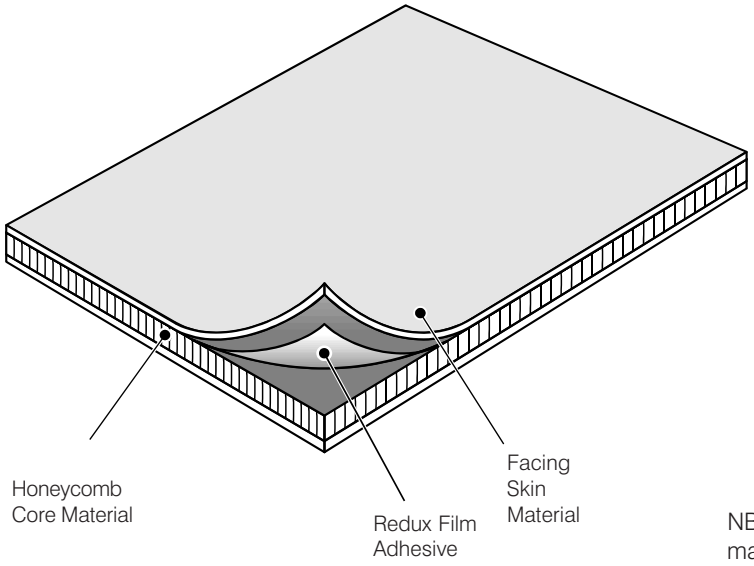
Heated Press

Ideally the panels should be assembled ready for curing as a single shot process. This method is suitable for metallic and prepreg (pre-impregnated) facing skins. Alternatively prepreg facing skin materials may be pre-cured by using a press, and subsequently bonding with a film adhesive layer.

Hexcel's Redux® range of film adhesives is well suited for these production methods.

Integrally bonded items such as extruded bar sections and inserts may be included and located by the honeycomb core or with simple tooling.



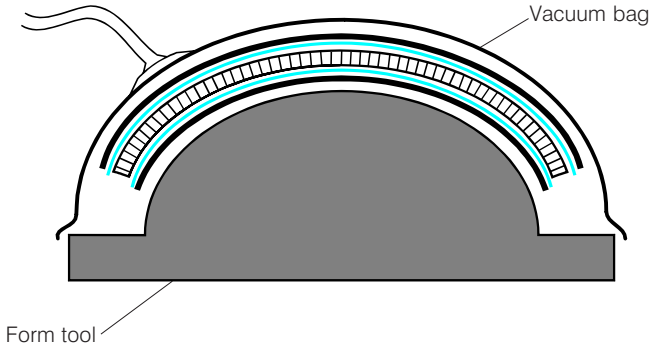


NB. Some further information related to production methods, may be found in Hexcel Composites publications: "Redux Bonding Technology" Ref: RGU 034b and "Prepreg Technology" Ref: FGU 017 available on request or via www.hexcelcomposites.com

Vacuum Bag Processing

The component should be assembled for cure as a single shot process, the necessary consolidation is obtained using a vacuum. This can be cured in an oven, and additional pressure can be applied if an autoclave is used.

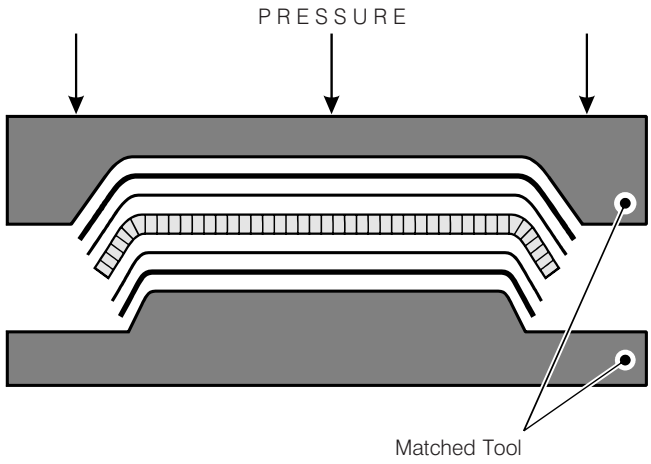
This method is suitable for items with prepreg or preformed composite or metallic facing skins. When flexible or formed honeycomb core and film adhesives are used complex items may be produced.

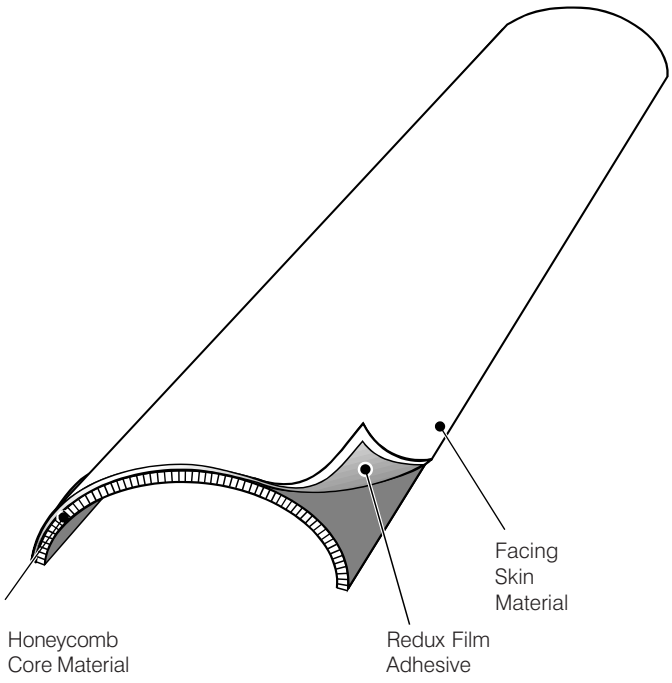


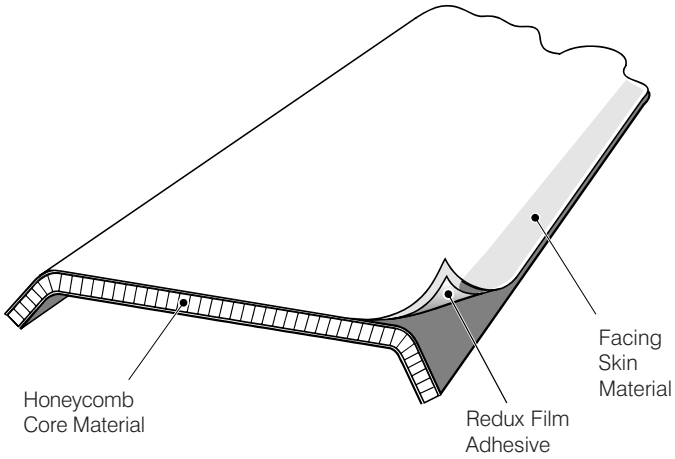
Match Mould Processing

This method is most suited to the single shot cure process where a key objective is to achieve production items with high levels of tolerance and surface finish. The heat and pressure cure cycle in this case is applied using a variety of methods. Typical methods are the use of heated tools with external mechanical pressure or non heated tools placed in a press or oven to achieve the full cycle.

Using a room temperature curing adhesive cold bonding may be considered if the sandwich construction is too large to be processed using the above methods, or if heating equipment is unavilable.







Advice on special methods and applications, plus information on equipment and suppliers can be obtained from Hexcel Composites on request.

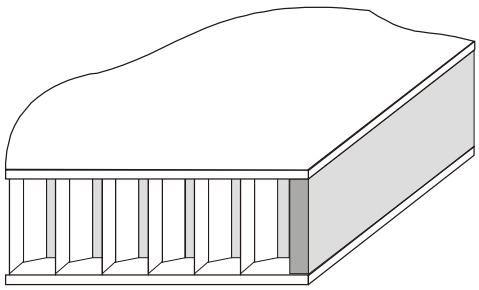
Sandwich Panel Edge Closure Design

When designing of sandwich panels it may be necessary to consider methods of closing or sealing the edges. Exposed edge areas are a potential weakness in the design as they may be susceptible to local impact or environmental damage.

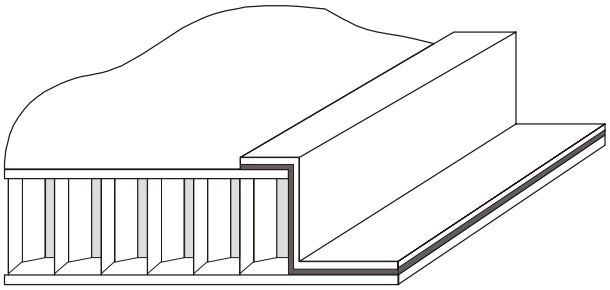
Edge closures may also provide local reinforcements, attachment points, or simply meet aesthetic requirements.

Illustrated are a number of methods commonly used to close sandwich boards:

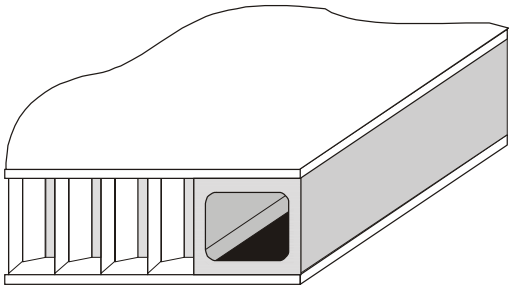
Further information on edge closure, board joining, fabrication and finishing methods is available in the Hexcel Composites publication “Sandwich Board Fabrication Technology” Ref: LTU 018, available on request.



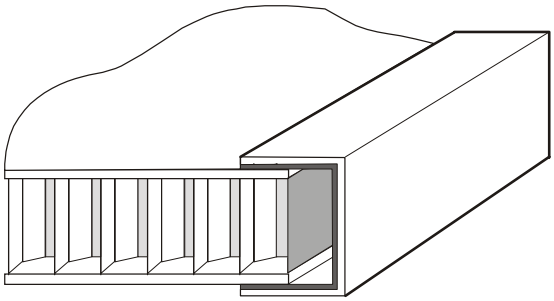
Panel closed with edge filler.



Bonded 'Z' section



Box extrusion



Bonded 'U' section

SAFETY

Handling Precautions

When fabricating from honeycomb sandwich board materials it is advisable to wear disposable clean cotton gloves throughout the entire operation. This helps to keep the panel clean, and affords protection for the operator's hands.

Glass fibre dust is an irritant. Avoid breathing the dust generated by cutting operations, and do not rub the eyes with hands which may be contaminated with the dust.

The usual precautions should be observed while working with synthetic resins.

Product Safety Data Sheets have been prepared for all Hexcel Composites products and are available to company safety officers on request.

APPENDIX I

Mechanical Properties of Honeycomb Materials - Typical Values at Room Temperature

PRODUCT CONSTRUCTION		COMPRESSION		PLATE SHEAR			
Density	Cell Size*	Stabilized		L Direction		W Direction	
kg/m³ (lb/ft³)	mm (in)	Strength MPa	Modulus MPa	Strength MPa	Modulus MPa	Strength MPa	Modulus MPa
3003 Aluminium							
29 (1.8)	19 (3/4)	0.9	165	0.65	110	0.4	55
37 (2.3)	9 (3/8)	1.4	240	0.8	190	0.45	90
42 (2.6)	13 (1/2)	1.5	275	0.9	220	0.5	100
54 (3.4)	6 (1/4)	2.5	540	1.4	260	0.85	130
59 (3.7)	9 (3/8)	2.6	630	1.45	280	0.9	140
83 (5.2)	6 (1/4)	4.6	1000	2.4	440	1.5	220
5052 Aluminium							
37 (2.3)	6 (1/4)	1.35	310	0.96	220	0.58	112
50 (3.1)	5 (3/16)	2.3	517	1.45	310	0.9	152
54 (3.4)	6 (1/4)	2.6	620	1.6	345	1.1	166
72 (4.5)	3 (1/8)	4.2	1034	2.3	483	1.5	214
83 (5.2)	6 (1/4)	5.2	1310	2.8	565	1.8	245
127 (7.9)	6 (1/4)	10.0	2345	4.8	896	2.9	364
130 (8.1)	3 (1/8)	11.0	2414	5.0	930	3.0	372
5056 Aluminium							
37 (2.3)	6 (1/4)	1.8	400	1.2	220	0.7	103
50 (3.1)	3 (1/8)	2.4	669	1.7	310	1.1	138
50 (3.1)	5 (3/16)	2.8	669	1.8	310	1	138
72 (4.5)	3 (1/8)	4.7	1275	3.0	483	1.7	193
HRH10 Nomex (Aramid)							
29 (1.8)	3 (1/8)	0.9	60	0.5	25	0.35	17.0
32 (2.0)	5 (3/16)	1.2	75	0.7	29	0.4	19.0
32 (2.0)	13 (1/2)	1.0	75	0.75	30	0.35	19.0
48 (3.0)	3 (1/8)	2.4	138	1.25	40	0.73	25.0
48 (3.0)	5 (3/16)	2.4	140	1.2	40	0.7	25.0
64 (4.0)	3 (1/8)	3.9	190	2.0	63	1.0	35.0
64 (4.0)	6 (1/4)	5.0	190	1.55	55	0.86	33.0
80 (5.0)	3 (1/8)	5.3	250	2.25	72	1.2	40.0
96 (6.0)	3 (1/8)	7.7	400	2.6	85	1.5	50.0
123 (7.9)	3 (1/8)	11.5	500	3.0	100	1.9	60.0
144 (9.0)	3 (1/8)	15.0	600	3.5	115	1.9	69.0
29 (1.8)	5 OX (3/16)	1.0	50	0.4	14	0.4	21.0
48 (3.0)	5 OX (3/16)	2.9	120	0.8	20	0.85	35.0
HRH78	Typical mechanical properties are similar to HRH10, however, the aramid sheet manufacturing tolerances are wider therefore minimum values may be reduced.						

Other foil thicknesses and cell size are available: see specific data sheet or Selector Guide, obtainable from Hexcel Composites on request.
*Please note that the exact cell sizes for HexWeb core are the imperial measurements. The metric values are provided for reference only.

APPENDIX II

Properties of typical facing materials for sandwich panel construction.

FACING MATERIAL	TYPICAL STRENGTH Tension/Compression MPa	MODULUS OF ELASTICITY Tension/Compression GPa	POISSON'S RATIO μ	TYPICAL CURED PLY THICKNESS mm	TYPICAL WEIGHT PER PLY kg/m^2
Epoxy UD CARBON tape (0°) 60% volume fraction	2000 / 1300	130 / 115	0.25	0.125	0.19
Epoxy UD GLASS tape (0°) 55% volume fraction	1100 / 900	43 / 42	0.28	0.125	0.25
Epoxy WOVEN CARBON (G793-5HS) 55% volume fraction	800 / 700	70 / 60	0.05	0.30	0.45
Epoxy WOVEN ARAMID (285K-4HS 60% volume fraction	500 / 150	30 / 31	0.20	0.20	0.27
Epoxy WOVEN GLASS (7781-8HS) 50% volume fraction	600 / 550	20 / 17	0.13	0.25	0.47
Phenolic WOVEN GLASS (7781-8HS) 55% volume fraction	400 / 360	20 / 17	0.13	0.25	0.47
ALUMINIUM Alloy 2024 T3 5251 H24 6061 T6	Av. Yield 270 150 240	Av. 70	0.33	0.50	1.35
STEEL carbon 1006 1017	Av. Yield 285 340	Av. 205	0.30	0.5	4.15
Exterior PLYWOOD Fir	30 / 35	Av. 9	0.1	12.7	6.3
Tempered HARDWOOD Teak	110 / 40	Av. 12	0.1	12.7	8.5

APPENDIX III

Summary of Formulae

BEAM (pages 12 to 13)

Bending Stiffness = $D = \frac{E_f t_f h^2 b}{2}$ where $h = t_f + t_c$

Shear Stiffness = $S = bhG_c$ where $G_c = G_L$ or G_W

Deflection $= \delta = \frac{K_b P l^3}{D}$ (bending) + $\frac{K_S P l}{S}$ (shear)

Facing Stress $= \sigma_f = \frac{M}{h t_f b}$ where M is from page 11

Core Stress $= \tau_c = \frac{F}{h b}$ where F is from page 11

PLATE (pages 14 to 17)

Plate Coefficient = i) $\frac{b}{a}$; ii) $R = \frac{G_L}{G_W}$; iii) $V = \frac{\pi^2 E_f t_f h}{2b^2 G_W \lambda}$

Deflection $= \delta = \frac{2K_1 q b^4 \lambda}{E_f t_f h^2}$

Facing Stress $= \sigma_f = \frac{K_2 q b^2}{h t}$

Core Shear $= \tau_c = \frac{K_3 q b}{h}$

Local Compression = $\sigma_c = \frac{P}{A} = \frac{q \times A}{A}$ (NB: also applicable to Beams)

END LOADING (pages 18 to 19)

Facing Stress $= \sigma_f = \frac{P}{2 t_f b}$ where P = q x b if applicable

Panel Buckling $= P_b = \frac{\pi^2 D}{l^2 + \frac{\pi^2 D}{G_c h b}}$ where D is as per beam

Shear Crimping $= P_b = t_c G_c b$

Skin Wrinkling $= \sigma_{CR} = 0.5 [G_c E_c E_f]^{1/3}$

Intra Cell Buckling $= \sigma_{CR} = 2 E_f \left[\frac{t_f}{s}\right]^2$

The above formulae assume symetrical items, with thin facings of the same skin material and thickness, and core relatively much less stiff than skins.

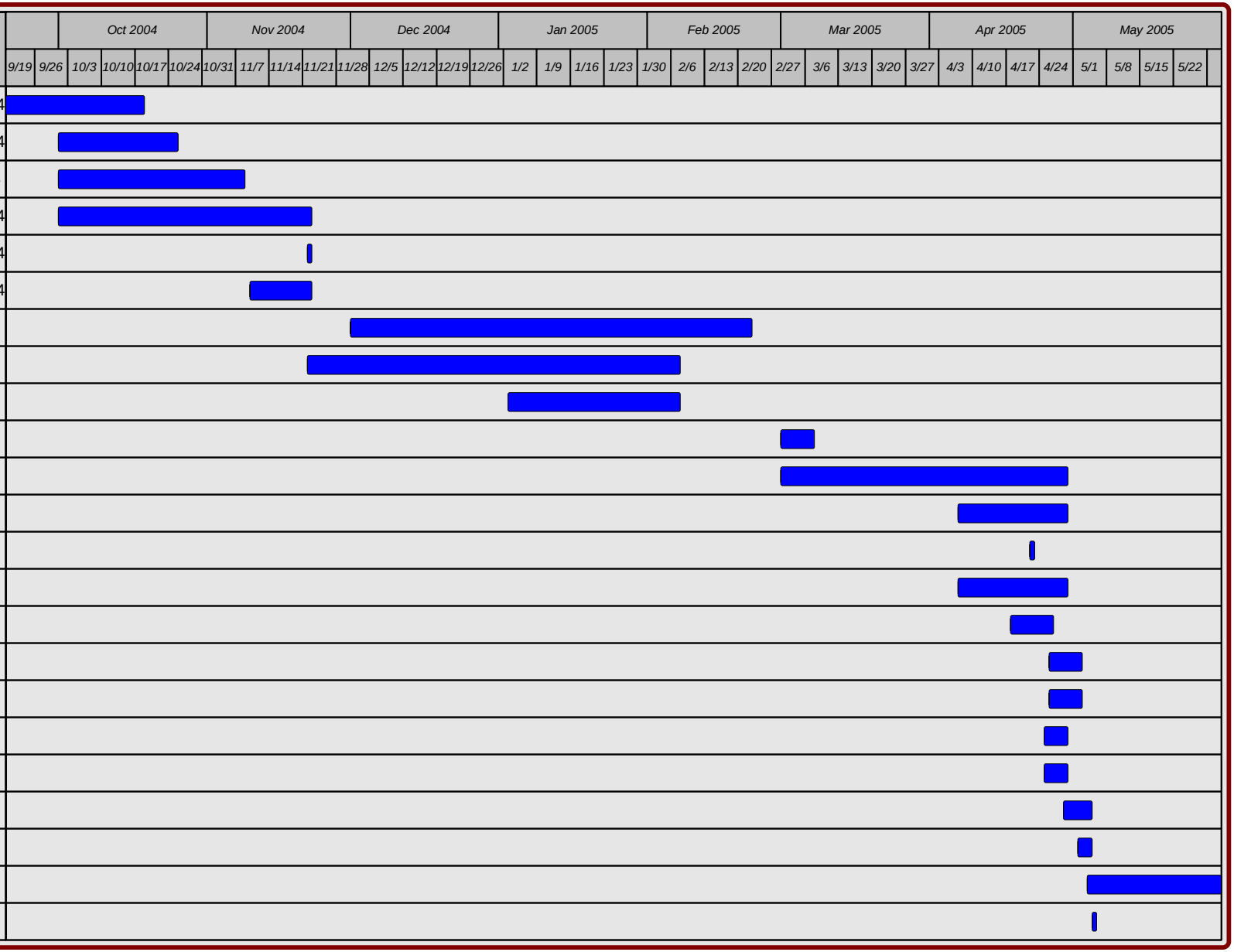
Further reading:

ZENKERT, D. - ***An Introduction to Sandwich Construction***
Emas Publishing, London (1997)

BITZER, T. - ***Honeycomb Technology***
Chapman & Hall, London (1997)

APPENDIX H – TIMELINE

ID	Task Name	Start	Finish
1	Project Description	9/20/2004	10/18/2004
2	Background Information	10/1/2004	10/25/2004
3	Subsystem Design	10/1/2004	11/8/2004
4	Fall Quarter Design	10/1/2004	11/22/2004
5	Fall Design Review	11/22/2004	11/22/2004
6	Physical Mock-up	11/10/2004	11/22/2004
7	Order Parts & Materials	12/1/2004	2/22/2005
8	Winter Design	11/22/2004	2/7/2005
9	Detailed Drawings	1/3/2005	2/7/2005
10	Manufacture Grating	3/1/2005	3/7/2005
11	Manufacture Pontoons	3/1/2005	4/29/2005
12	Manufacture End caps	4/7/2005	4/29/2005
13	Manufacture Charger Box	4/22/2005	4/22/2005
14	Manufacture Battery Box	4/7/2005	4/29/2005
15	Manufacture Winch Interface	4/18/2005	4/26/2005
16	Manufacture Arms	4/26/2005	5/2/2005
17	Manufacture Deck	4/26/2005	5/2/2005
18	Manufacture Clamps	4/25/2005	4/29/2005
19	Manufacture Thruster mounts	4/25/2005	4/29/2005
20	Assemble Parts	4/29/2005	5/4/2005
21	Integrate Electrical and Mechanical Components	5/2/2005	5/4/2005
22	Test	5/4/2005	8/30/2005
23	Senior Design Conference	5/5/2005	5/5/2005



APPENDIX I – BUDGET/EQUIPMENT LIST

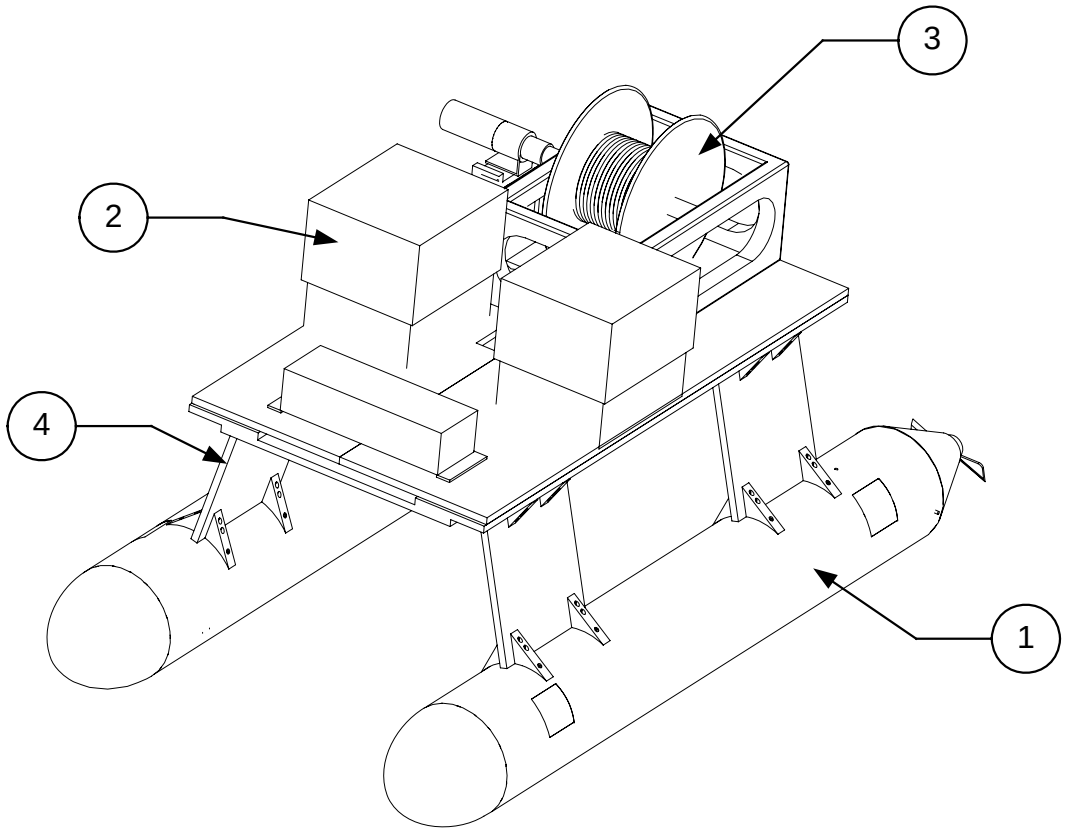
ASV
Budget
rev 6/5/05

Category	Income		
Base Unit	U of Alaska	**$20,000**	
Sensors (add on)	U of Alaska	**$65,000**	
		$85,000	

Category	SubSystem	1st Budget (Gut)	Current Budget	Spent	Donated	Will spend - future years	Spent+ Donated +Will spend	Component description	Part #	# of Items	B/M/O	Vendor
Mechanical	Flotation	*$1,000*	$200	$200	$0	$50	$250	Foam	FL-5	4	B	
			$60	$60	$0	$0	$60	Fiberglass	FL-1,3	1	B	fiberglass supply
			$90	$90	$0	$0	$90	Epoxy	FL-1,3	1	B	
			$0	$0	$150	$0	$150	PVC	FL-2	0	M	Preston pipes
	Deck	*$500*	$300	$0	$200	$0	$200	Honey Comb Aluminum	DK-1,2	10	M	MBARI
			$0	$0	$100	$0	$100	Deck Grating	DK-3	0	M	Ingomar
	Thrusters	*$3,000*	$1,600	$1,600	$0	$0	$1,600	Minn Kota RT55AP	FL-4	2	B	Minn Kota
	Winch	*$2,000*	$1,945	$1,945	$0	$0	$1,945	Frame and Reel	WM-2	1	B	Shark Marine
			$1,662	$1,662	$0	$0	$1,662	Level wind	WM-2	1	B	Shark Marine
			$1,027	$1,027	$0	$0	$1,027	12 cdtr. Slipring	WM-2	1	B	Shark Marine
			$1,200	$0	$0	$1,200	$1,200	Encoder	WM-2	1	B	Shark Marine
			$2,350	$2,350	$0	$0	$2,350	Cable (200m)	WM-2	1	B	Shark Marine
			$300	$350	$0	$0	$350	Motor + Mount	WM-1,3	1	M	Warn
	Docking	*$1,500*	$12	$12	$0	$0	$12	Cleats	DK-6	4	B	West Marine
			$500	$0	$0	$500	$500	Trailer	TR-1	1	B	
							$0					
	Boxes/Conectors	*$500*	$140	$500	$0	$0	$500	Waterproof Containers	BC-1	4	B	Otter Box
	/Waterproofing		$20	$0	$0	$20	$20	wet mateable conductors	BC-2	4		
	Assrt. Parts	*$100*	$100	$100	$0	$200	$300					
	Mechanical	***$8,600***	***$11,506***	***$9,896***	***$450***	***$1,970***	***$12,316***					
Electrical												
	Power	*$1,000*	$836	$836	$0	$0	$836	370Ah 6V Battery	PW-1	4	B	West Marine
			$1,100	$0	$0	$1,100	$1,100	2000w Generator	PW-2	1	B	Honda
		$300	$600	$300	$0	$0	$300	Charger MK440	PW-3	2	B	
	Processor	*$1,000*	$1,000	$1,000	$0	$0	$1,000					
	Circuit boards	*$100*	$100	$0	$1,500	$0	$1,500					
	Radio Modem	*$400*	$400	$400	$0	$0	$400					
	Camera & Video	*$900*	$900	$900	$0	$0	$900					
	Antenna	*$100*	$100	$0	$0	$100	$100					
	Shore PC	*$1,000*	$1,000	$0	$1,000	$0	$1,000					
	Electrical	***$4,800***	***$6,036***	***$3,436***	***$2,500***	***$1,200***	***$7,136***					
	Total (ME, EE)	***$13,400***	***$17,542***	***$13,332***	***$2,950***	***$3,170***	***$19,452***					

Instruments												
	GPS	*$100*	$100	$100	$0	$0	$100					
	Altimiter	*$1,000*	$1,000	$0	$0	$1,000	$1,000					
	Compass	*$50*	$50	$50	$0	$0	$50					
	ADCP	*$40,000*	$40,000	$0	$0	$40,000	$40,000					
	Digital O_2	*$4,000*	$4,000	$0	$0	$4,000	$4,000					
	CTD	*$15,000*	$15,000	$0	$1,500	$0	$1,500					
	Fluorometer	*$5,000*	$5,000	$0	$0	$5,000	$5,000					
	Instruments	***$65,150***	***$65,150***	***$150***	***$1,500***	***$50,000***	***$51,650***					
	Grand Total	***$78,550***	**$82,692**	**$13,482**	**$4,450**	**$53,170**	**$71,102**					

APPENDIX J – DETAIL/ASSEMBLY DRAWINGS



Item No.	Part Number	Qty	Description
1	FL	1	Flotation Assembly
2	PW	1	Power Assembly
3	WM	1	Winch Assembly
4	DK	1	Deck Assembly

REV	DESCRIPTION	BY	DATE
B	as built	PDM	6/6/05

REVISION HISTORY

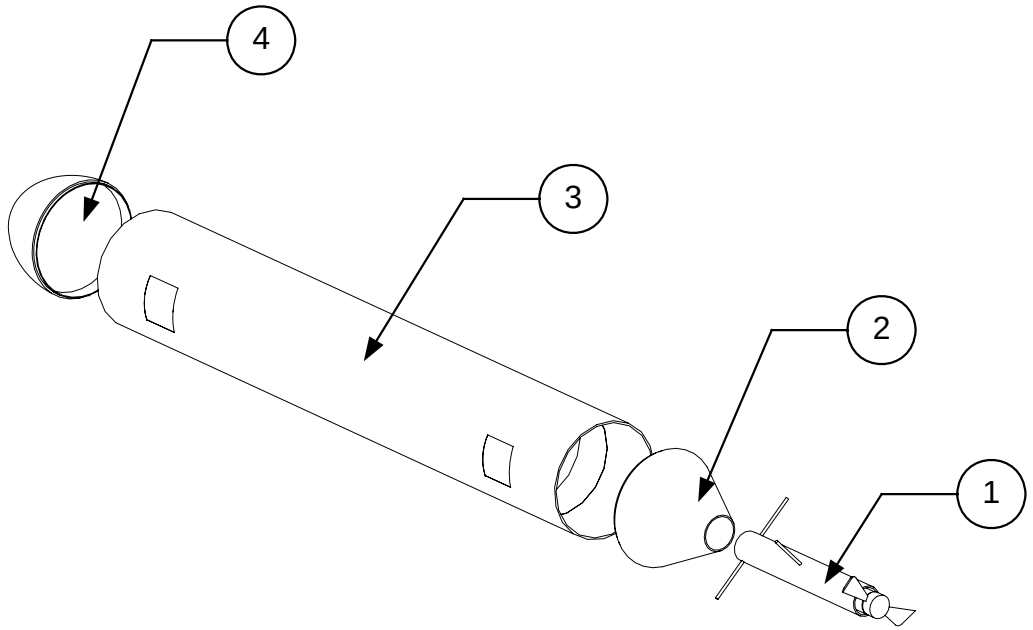
THE WASP

Supplier

TITLE: System Assembly

SIZE	UNITS	DWG NO	REV
A	ft	AS-SYS	B

SCALE NTS | QUNTY 1x | SHEET 1 of 1



Item No.	Part Number	Qty	Description
1	FL-4	1	Thruster mount
2	FL-3	1	Rear End Cap
3	FL-2	1	Main Tube
4	FL-1	1	Front End Cap

REV	DESCRIPTION	BY	DATE
B	as built	PDM	6/6/05

REVISION HISTORY

THE WASP

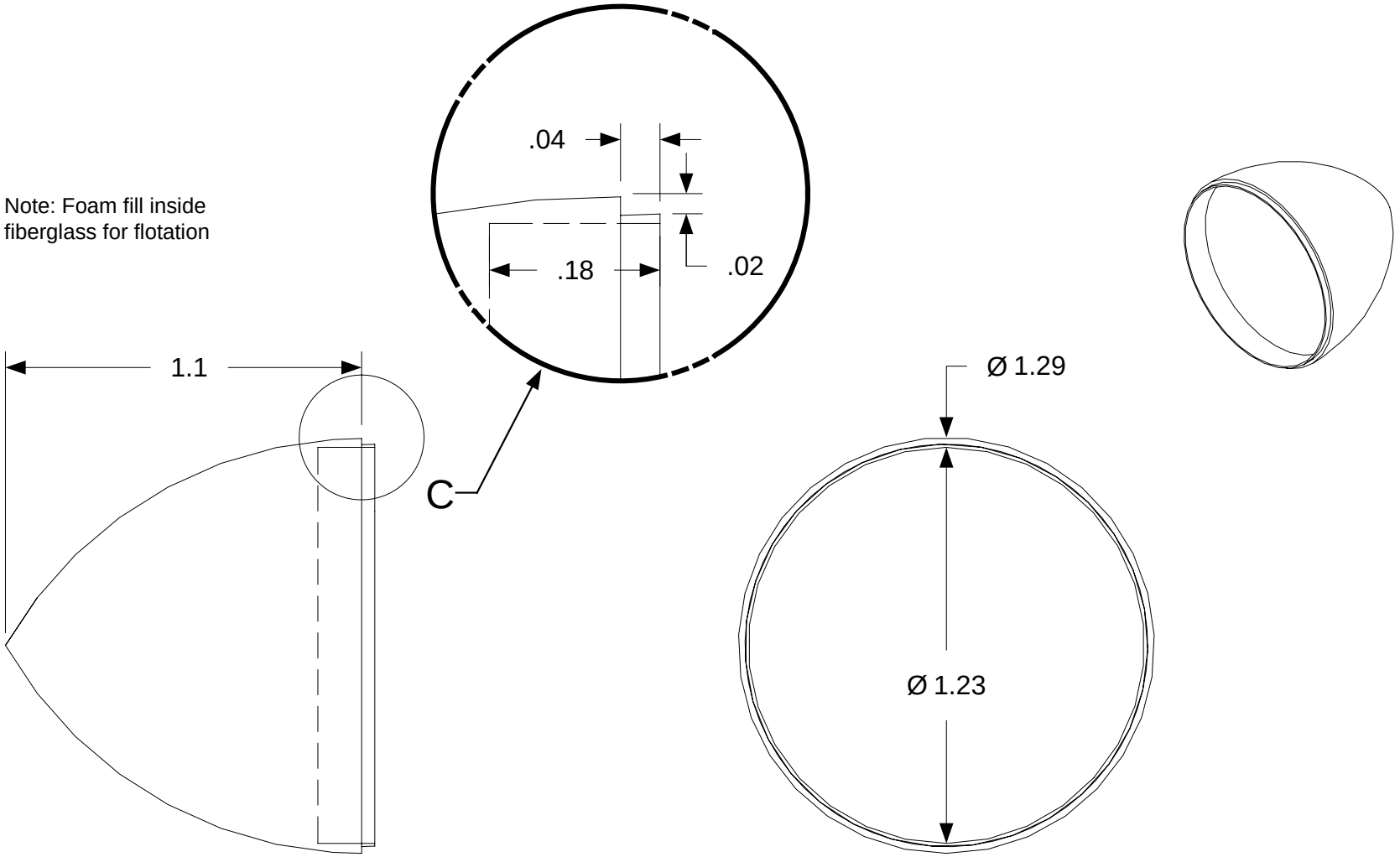
Supplier

TITLE: Flotation Assembly

SIZE	UNITS	DWG NO	REV
A	ft	AS-FL	B

SCALE	QUNTY	SHEET
NTS	1x	1 of 1

Note: Foam fill inside
fiberglass for flotation



REV	DESCRIPTION	BY	DATE
A	Release for notes	PDM	2/2/05
B	Redline	PDM	3/15/05
C	as built	PDM	6/6/05

REVISION HISTORY

TOLERANCE
.x ± .01
.xx ± .001
.xxx ± .0005

THE WASP

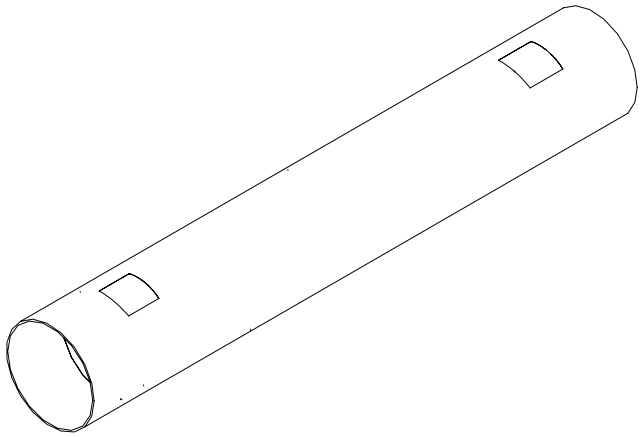
Supplier
Fiberglass Supply

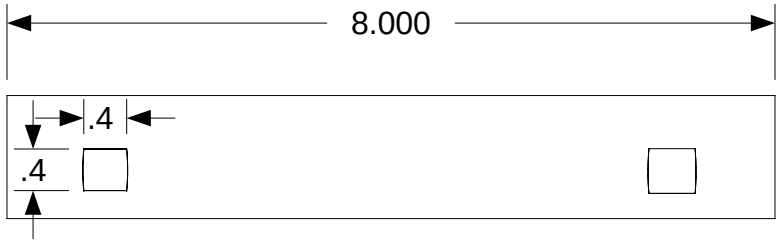
Material
Fiberglass w/ Foam Fill

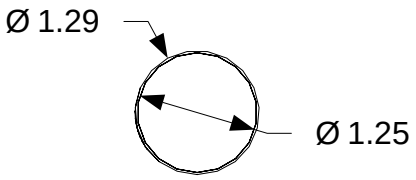
TITLE
Front End Cap

SIZE	UNITS	DWG NO	REV
A	ft	FL-1	C

SCALE	QUNTY	SHEET
	2x	1 of 1







REV	DESCRIPTION	BY	DATE
A	Release for notes	PDM	2/2/05
B	Redline	PDM	3/15/05
C	as built	PDM	6/6/05

REVISION HISTORY

TOLERANCE
.x ± .01
.xx ± .001
.xxx ± .0005

THE WASP

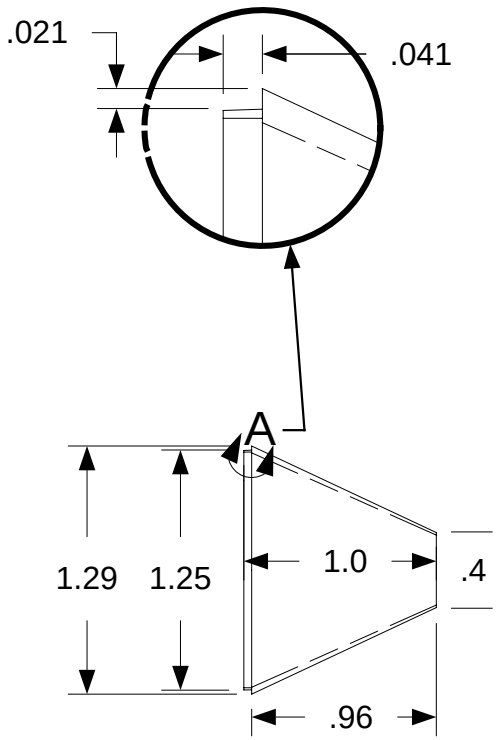
Material: 15in Schedule 40 PVC

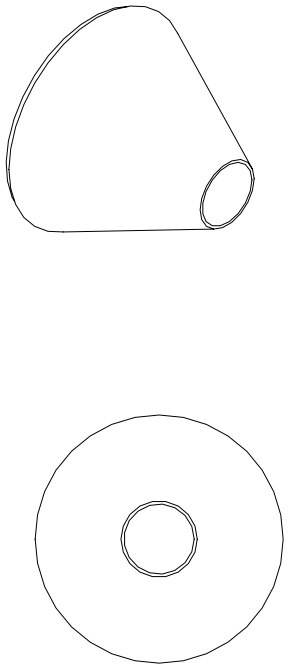
Supplier: Preston Pipes

TITLE: Main Tube

SIZE	UNITS	DWG NO	REV
A	ft	FL-2	C

SCALE	QUNTY	SHEET
	2x	1 of 1





REV	DESCRIPTION	BY	DATE
A	Release for notes	PDM	2/7/05
B	Redline	PDM	3/15/05
C	as built	PDM	6/6/05

REVISION HISTORY

TOLERANCE
.x ± .01
.xx ± .001
.xxx ± .0005

THE WASP

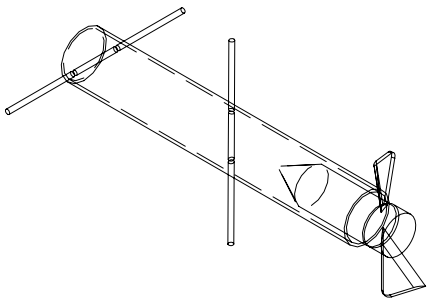
Supplier
Fiberglass Supply

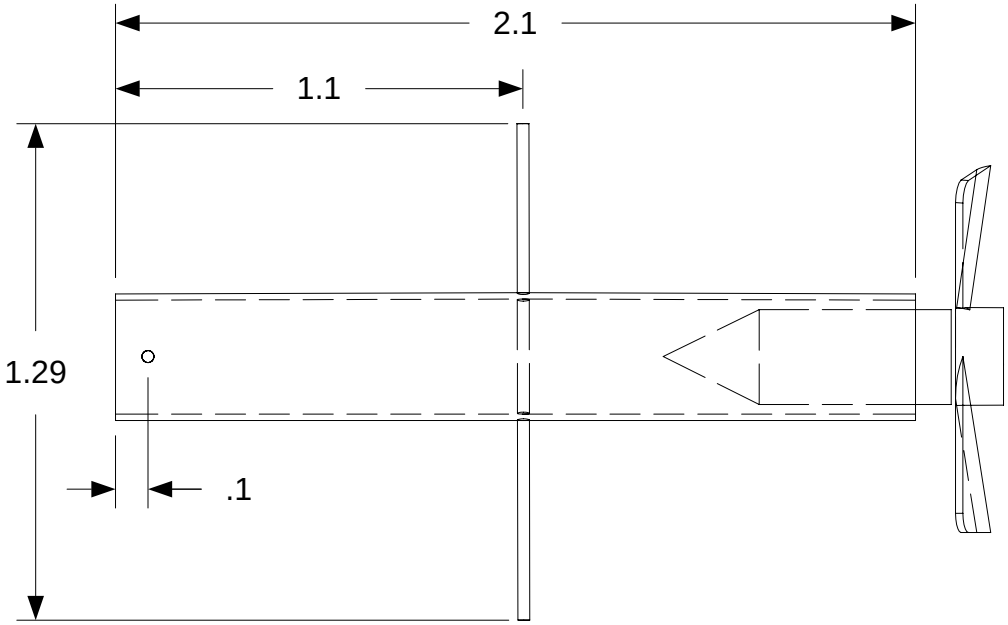
Material
Fiberglass

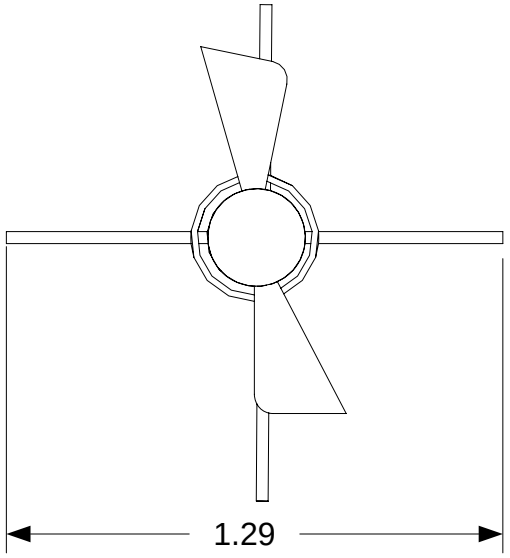
TITLE
Rear End Cap

SIZE	UNITS	DWG NO	REV
A	ft	FL-3	C

SCALE	QUNTY	SHEET
NTS	2x	1 of 1







REV	DESCRIPTION	BY	DATE
A	as built	PDM	6/6/05

REVISION HISTORY

TOLERANCE
.x ± .01
.xx ± .001
.xxx ± .0005

THE WASP

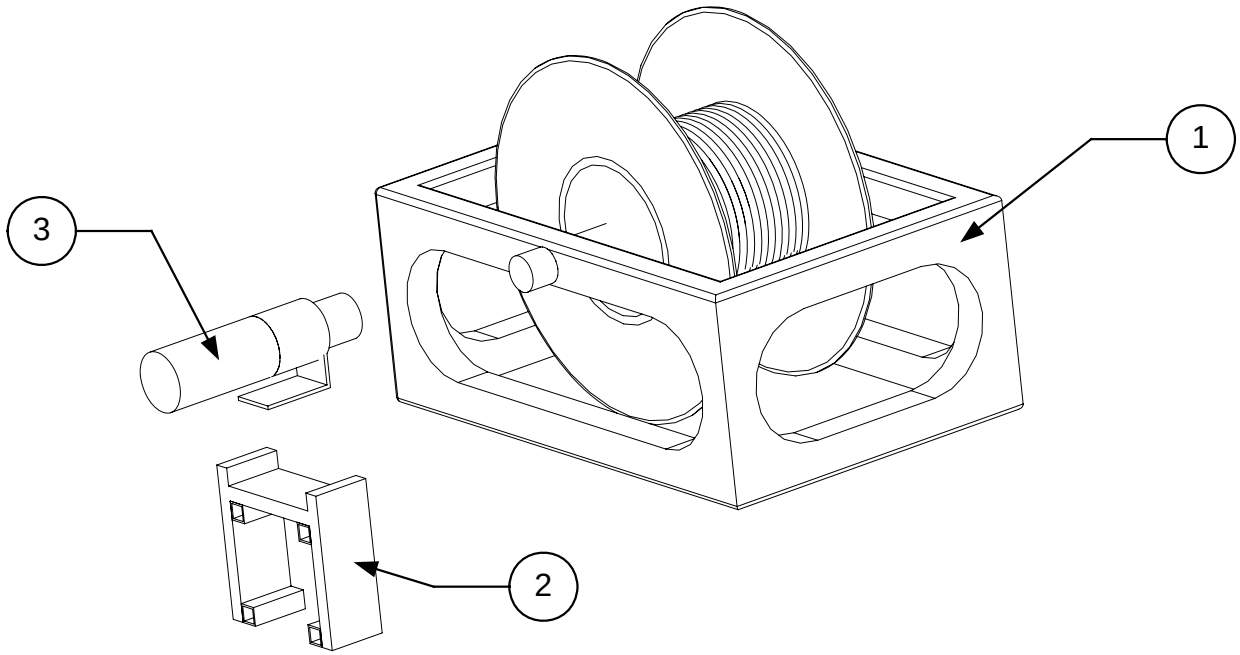
Supplier

Material: 4in ABS plastic / .375in All-thread

TITLE: Thrusters

SIZE	UNITS	DWG NO	REV
A	ft	FL-4	A

SCALE	QUNTY	SHEET
	2x	1 of 1



REV	DESCRIPTION	BY	DATE
B	as built	PDM	6/6/05

REVISION HISTORY

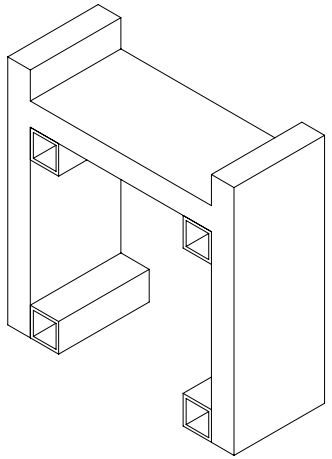
Item No.	Part Number	Qty	Description
1	WM-2	1	Reel and Teather
2	WM-1	1	Winch Motor Mount
3	WM-3	1	Winch Motor

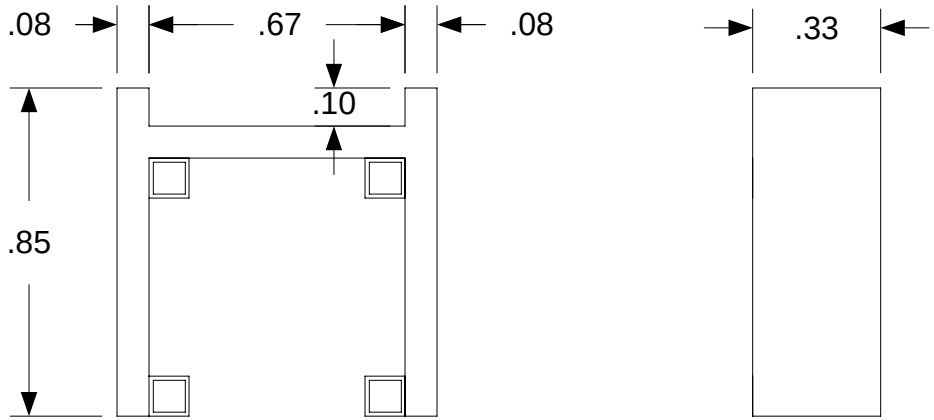
Supplier

TITLE: Winch Assembly

SIZE	UNITS	DWG NO	REV
A	ft	AS-WM	B

SCALE	QUNTY	SHEET
NTS	1x	1 of 1





REV	DESCRIPTION	BY	DATE
A	as built	PDM	6/6/05

REVISION HISTORY

TOLERANCE
.x ± .01
.xx ± .001
.xxx ± .0005

THE WASP

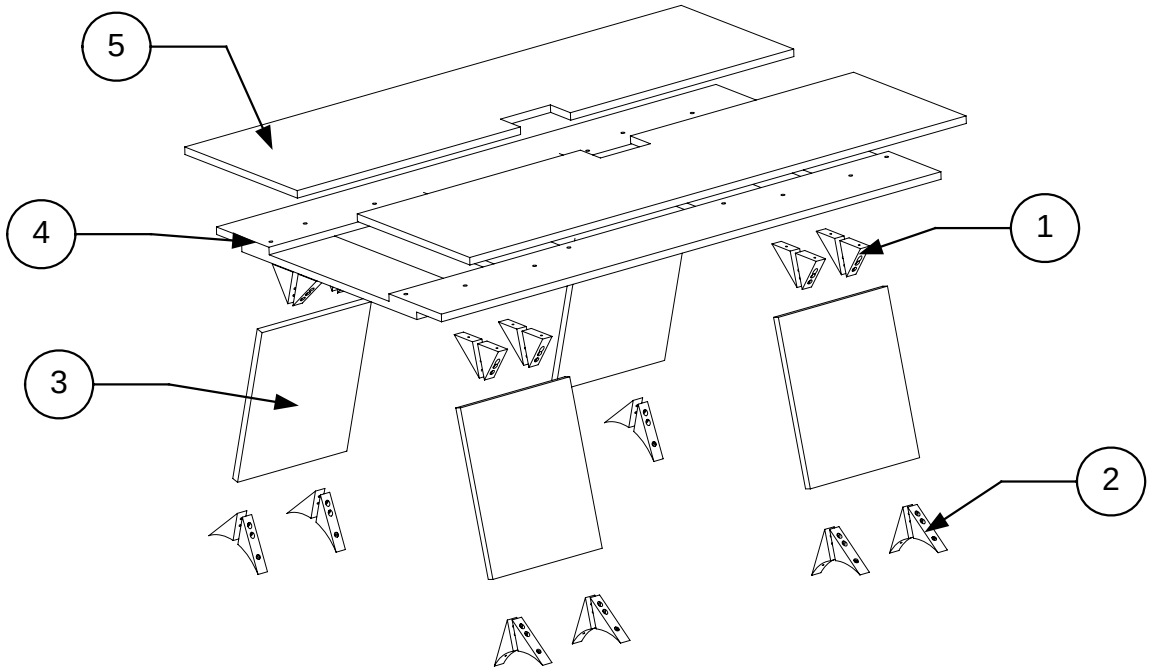
Supplier
MBARI

Material
Honeycomb Aluminium w/ 1in square aluminum

TITLE
Winch Motor Mount

SIZE	UNITS	DWG NO	REV
A	ft	WM-1	A

SCALE	QUANTY	SHEET
NTS	1	1 of 1



Item No.	Part Number	Qty	Description
1	DK-4	8	Top Clamp
2	DK-5	8	Bottom Clamp
3	DK-2	4	Arm
4	DK-1	1	Deck Frame
5	DK-3	2	Deck Grating

REV	DESCRIPTION	BY	DATE
B	as built	PDM	6/6/05

REVISION HISTORY

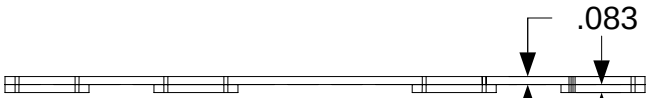
THE WASP

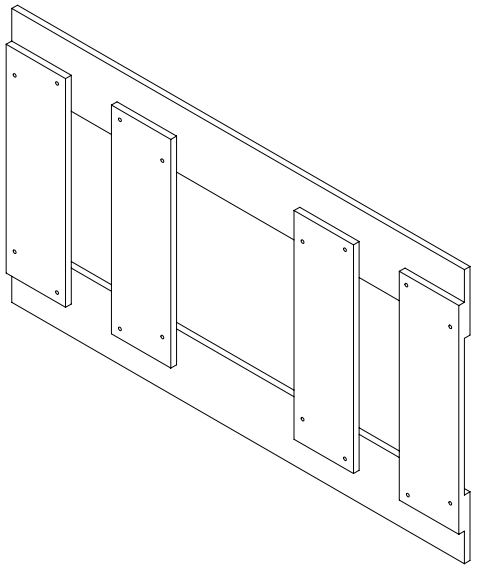
Supplier

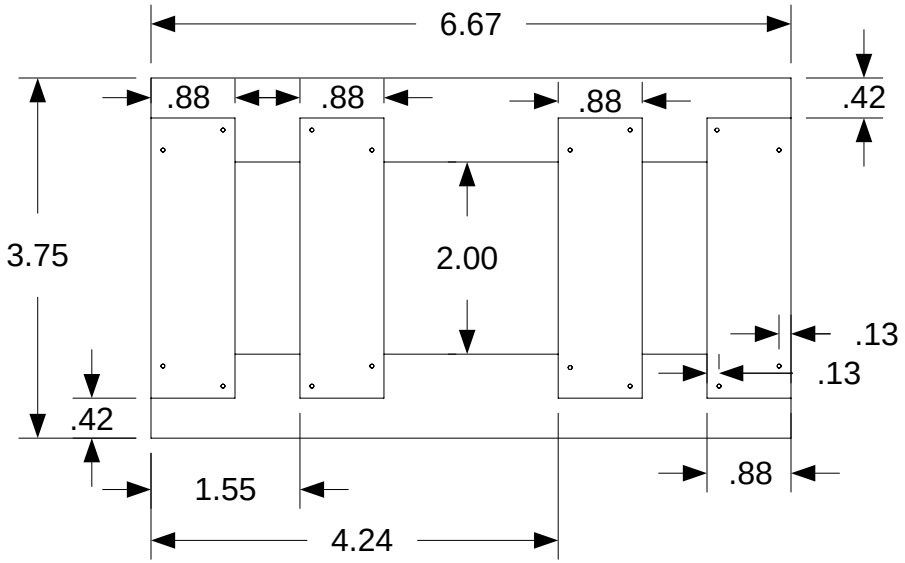
TITLE: Deck Assembly

SIZE	UNITS	DWG NO	REV
A	ft	AS-DK	B

SCALE	QUNTY	SHEET
NTS	1x	1 of 1







REV	DESCRIPTION	BY	DATE
A	Release for notes	PDM	2/2/05
B	Redline	PDM	3/15/05
C	as built	PDM	6/6/05

REVISION HISTORY

TOLERANCE
.x ± .01
.xx ± .001
.xxx ± .0005

THE WASP

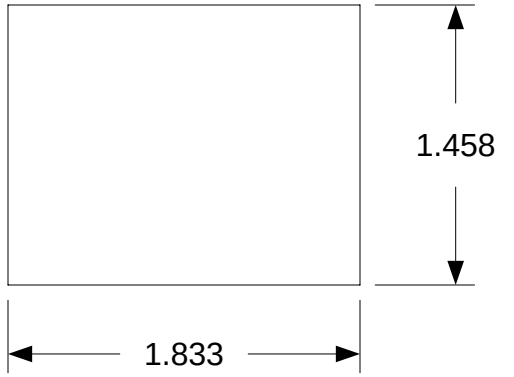
Supplier
MBARI

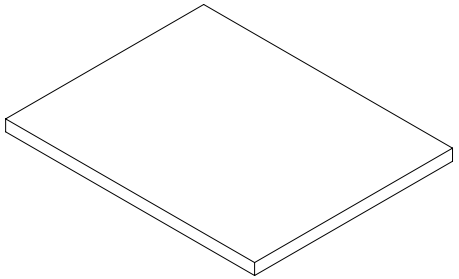
Material
Honeycomb Aluminium

TITLE
Deck Frame

SIZE	UNITS	DWG NO	REV
A	ft	DK-1	C

SCALE	QUANTY	SHEET
NTS	2	1 of 1





Note: 8x .5in holes to be drilled
through arm, alignd w/ clamps

REV	DESCRIPTION	BY	DATE
A	Release for notes	PDM	2/2/05
~~B~~	~~Holes for balast vent~~	PDM	3/15/05
C	Redline	PDM	3/15/05
D	as built	PDM	6/6/05

REVISION HISTORY

TOLERANCE
.x ± .01
.xx ± .001
.xxx ± .0005

THE WASP

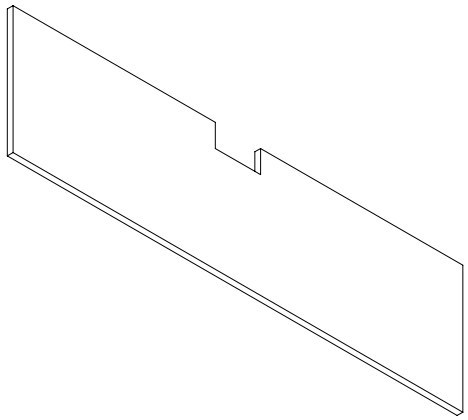
Supplier: MBARI

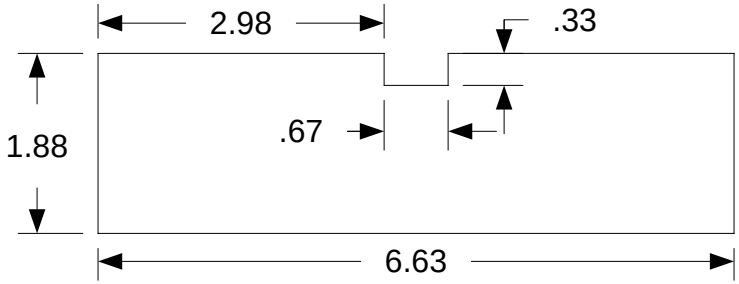
Material: Honeycomb Aluminium

TITLE: ARM

SIZE	UNITS	DWG NO	REV
A	ft	DK-2	D

SCALE	QUANTY	SHEET
NTS	4	1 of 1





REV	DESCRIPTION	BY	DATE
A	Release for notes	PDM	2/2/05
B	Resize	PDM	2/7/05
C	Redline	PDM	3/15/05
D	as built	PDM	6/6/05

REVISION HISTORY

TOLERANCE
.x ± .01
.xx ± .001
.xxx ± .0005

THE WASP

Supplier
Ingomar Packing Co.

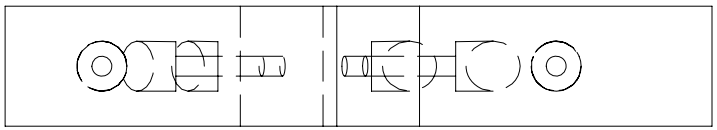
Material
Fiberglass Grating w/ 1" Grid

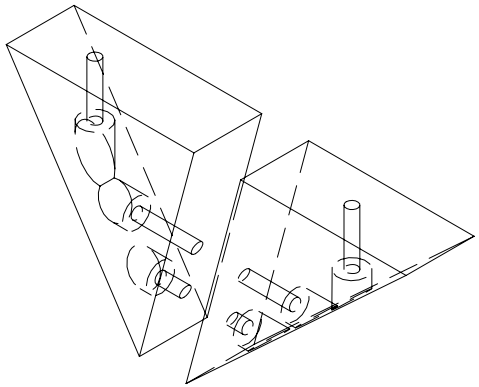
TITLE
Deck Grating

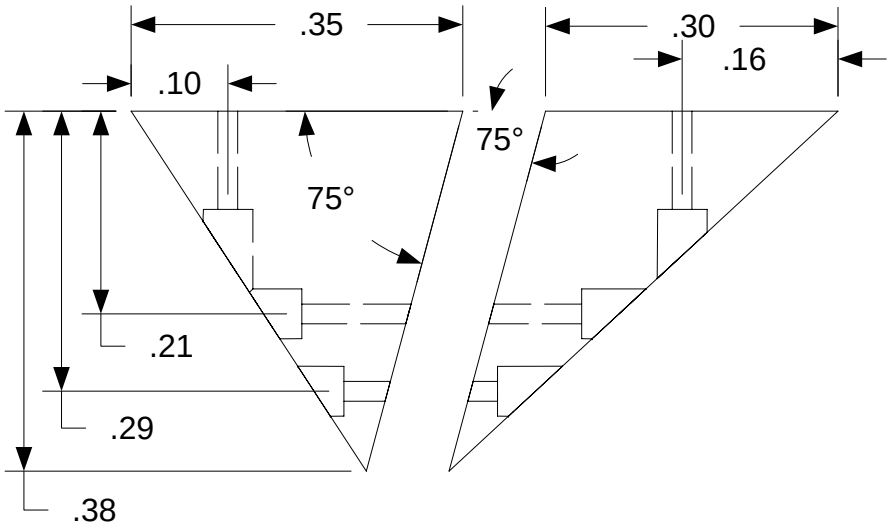
SIZE	UNITS	DWG NO	REV
A	ft	DK-3	D

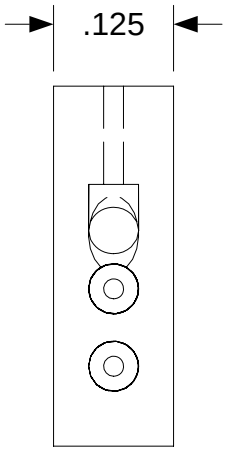
SCALE	QUNTY	SHEET
NTS	2	1 of 1

Note: .25in through-holes &
.625in bore









REV	DESCRIPTION	BY	DATE
A	Release for notes	PDM	3/15/05
B	as built	PDM	6/6/05

TOLERANCE
.x ± .01
.xx ± .001
.xxx ± .0005

REVISION HISTORY

THE WASP

Supplier
MBARI

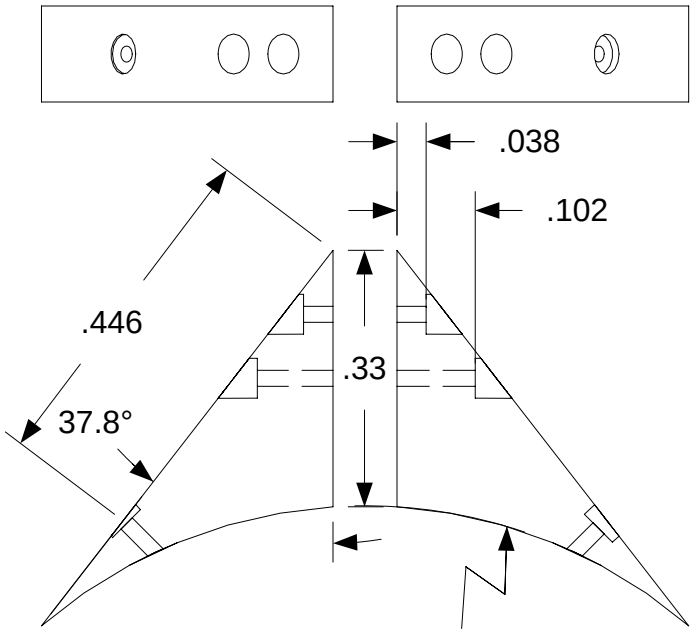
Material
UHMW Plastic

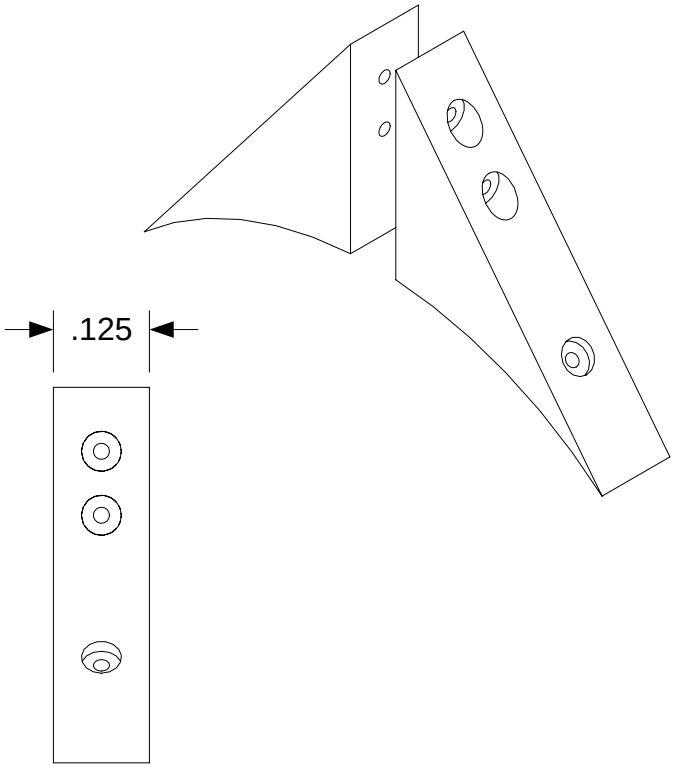
TITLE
Clamp Top

SIZE	UNITS	DWG NO	REV
A	ft	DK-4	B

SCALE	QUNTY	SHEET
NTS	8x	1 of 1

Note: .25in through-holes &
.625in bore
(part is semetrical about middle)





REV	DESCRIPTION	BY	DATE
A	Release for notes	PDM	3/15/05

REVISION HISTORY

TOLERANCE
.x ± .01
.xx ± .001
.xxx ± .0005

THE WASP

Supplier

Material: UHMW Plastic

TITLE: Clamp Bottom

SIZE	UNITS	DWG NO	REV
A	ft	DK-5	A

SCALE	QUNTY	SHEET
NTS	8x	1 of 1

APPENDIX K – MANUFACTURER SPECIFICATIONS

SHARK MARINE
Technologies Inc.

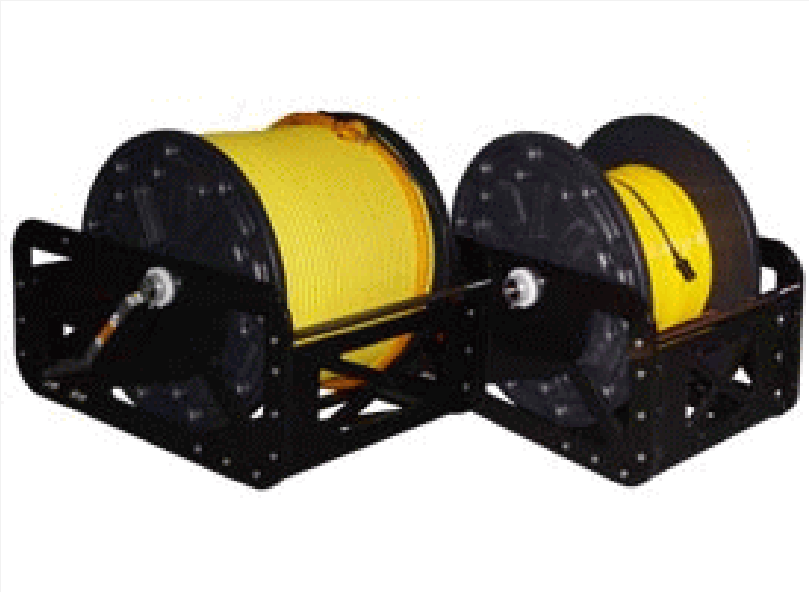
23 Nihan Dr., Unit 4 St. Catharines,
Ontario, Canada. L2N 1L2
Phone: (905) 687-6672
Fax: (905) 687-9742
E-mail: sales@sharkmarine.com

CABLE REELS

Features

- strong
- durable
- light weight
- low cost
- corrosion resistant materials
- base unit 23 kgs.
- options

- powered with remote hand control
  - lifting bar
  - level wind
  - encoder
- easy changing of cables
- same cable can be used with or without the reel system



SHARK MARINE
Technologies Inc.

23 Nihan Dr., Unit 4 St. Catharines,
Ontario, Canada. L2N 1L2
Phone: (905) 687-6672
Fax: (905) 687-9742
E-mail: sales@sharkmarine.com

Reel Specifications:

Reel Size	29.2cm/11.5inch	48.25cm /19inch
Height :		
without handle	64.75cm /25.5 inches	64.75cm /25.5 inches
with handle	82.5cm 32.5 inches)	82.5cm /32.5 inches
Width	58.4cm (23 inches)	75cm /29.5 inches
Length	87.6cm (34.5 inches) with out drive motor 90.17cm (35.5 inches) with drive motor	87.6cm /34.5 inches without drive motor 90.17cm/35.5 inches with drive motor
Weight	22.25kg(49 lbs) empty no drive motor 36kg (79 lbs) empty with drive motor	24.5kg /54 lbs empty no drive motor 38kg/84 lbs empty with drive motor
Drive Speed	Maximum 73m/min(240ft/min) Minimum 3.5m/min(12ft/min)	
Bollard Pull	Max 45.3kg (100lbs) (Adjustable)	

Optional Weather resistance 120vac Drive, ½ hp DC motor equipped with variable speed & dual directional hand control.
Drive Speed Maximum (240ft/min),Minimum (12ft/min) With a Minimum power required 450w 120 vac

Optional deck cable/hand controller pouch.

Shark Marine Technologies Inc. - Reel Capacities

Maximum:

	Sonar .320	Camera .400	Stealth .550	SeaWolf .830
24"x10"x11"core	2965'	1935'	964'	422'
24"x19"x11"core	5643'	3639'	1821'	774'

Figures are based on a perfect wind to the outside edge of reel.

Recommended:

	Sonar .320	Camera .400	Stealth .550	SeaWolf .830
24"x10"x11"core	2500'	1500'	500'	250'
24"x19"x11"core	5000'	3000'	1500'	500'

SHARK MARINE
Technologies Inc.

23 Nihan Dr., Unit 4 St. Catharines,
Ontario, Canada. L2N 1L2
Phone: (905) 687-6672
Fax: (905) 687-9742
E-mail: sales@sharkmarine.com

TETHER AND CABLE REEL
Operations Manual

Reel/ Slipring System

Shark Marine Quick Change Out Slipring System will allow you to utilize your existing cables with little to no modifications.

Shark Marine highly recommends that a strain relief is placed on the topside end of the cable, This strain relief is to be connected to the reel to insure if the cable was ever to be entirely wound off the reel that no pulling force would be exerted in the cable connectors.

Safety

Keep other equipment, hands, loose clothing from all moving parts during operating. Ensure all personal are aware of any tripping hazards that may occur in the operation of this product.

If free wheeling the cable from the reel ensure the handle is removed from the handle nipple and stored in the handle pocket.

Ensure the hand drive handle is not connected and the reel hand lock is disengaged before operation reel with motor drive system.

Motor drive systems must be operating with a Ground Fault Current Interrupter (GFCI)

Setting Strain Relief

The strain relief should be placed on the cable in a location that will allow sufficient cable to go through the reel core and approximately 4 inch out the end so you can connect it to the slipring system.

SHARK MARINE
Technologies Inc.

23 Nihan Dr., Unit 4 St. Catharines,
Ontario, Canada. L2N 1L2
Phone: (905) 687-6672
Fax: (905) 687-9742
E-mail: sales@sharkmarine.com

To install cable

Insuring that the end of your topside cable is compatible with the slipring system, remove the slipring system from the reel. Refer to section **Slipring Connection**
It is recommended to test your system using the slipring system before installing your cable on the reel.
After the cable has been tested Inset the topside end of your cable into the hole marked on the centre core of the reel, you may find that you will needed a flash light to insuring the cable goes in to the cut out of the centre shaft and comes out where you have already removed the slipring system.
At this point you can set you strain relief length location and connect the strain relief to the reel side with hardware provided, and connect the slipring to your cable as instructed in section below.
This system can be used with out sliprings, spool out your need cable connect deck cable to end sticking out from centre of spool.
NOTE: You MUST disconnect the deck cable from reel cable before winding or unwinding cable reel if no slipring is used.
This spool is designed to wind over the top

Slipring Connection

To remove the slipring from the reel, remove the three screws that attach the slipring guard and break out box from the reel frame, followed by the set screw from the end of the centre shaft which holds the slipring body into place.

Carefully slide the slipring body from the centre shaft.
Install cable as stated in **To install cable** section
Connect the cable to the slipring body.
Using rubber tape and in a spiral fashion, wrap this connection starting at the slipring body winding on the cable insuring a tight wrap with plenty of over lap. This will aid in weather proofing this system.

Carefully slide the cable and slipring back into the centre shaft and resecuring the guard and box to reel frame. If you find it difficult to slide the slipring back in to the centre shaft **do not force it** . You may need to remove the manual handle nipple and using a small blunt hook to pull up any slack on the cable inside the centre shaft.
Connect the deck cable to the receptacle on the bottom of the slipring break out box and ensure the strain relief is connected to the eye pad provided.

Hand Drive

To operate, remove the hand crank handle from the handle pocket. Insert it on to the spool handle nipple, making sure the handle lock is secure.
If free wheeling the cable from the reel ensure the handle is removed from the handle nipple and stored in the handle pocket.

Reel hand lock

SHARK MARINE
Technologies Inc.

23 Nihan Dr., Unit 4 St. Catharines,
Ontario, Canada. L2N 1L2
Phone: (905) 687-6672
Fax: (905) 687-9742
E-mail: sales@sharkmarine.com

This Reel System is equipped with a reel locking mechanism, This can be used to prevent the reel from turning during normal operations. The lock has five location holes located on the reel face. Simply pull the lever to engage the lock pin into the reel hole. Ensure that the lock is disengaged before using motor drive if so equipped.
In addition to the Reel Hand Lock, Shark Marine highly recommends that a cable tow clamp is used to maintain cable tension

Motor Drive
Ensure the hand drive handle is removed and the reel hand lock is disengaged before operating reel with motor drive system to reduce damage or injures.
Shark Marine recommends that a Ground Fault Current Interrupter (GFCI) is used to operating the equipment.
Before connecting power ensure that the breaker located on the side of the motor drive box is in the off position. Connect the hand control to the motor drive box ensuring the speed control is in slow.

Maintenance
This is extremely important for salt water use
Wash reel system with mild soap, soft cloth. Rinse gently with fresh water, wiping all excess water and let air dry.

Lubrication
There are two grease nipples found on the centre shaft bearing , these should be lubricated in a regular bases or as needed with a marine type grease and all excess wiped off
Chain & Sprocket should be inspected for corrosion , wiped off and applying thin layer oil on a regular bases

SHARK MARINE
Technologies Inc.

23 Nihan Dr., Unit 4 St. Catharines,
Ontario, Canada. L2N 1L2
Phone: (905) 687-6672
Fax: (905) 687-9742
E-mail: sales@sharkmarine.com

Dimensions (without lifting bar):	64.75cm (25.5") high 58.4cm (23") wide 87.6cm (34.5") deep	64.75cm (25.5") high 75cm (29.5 inches) 87.6cm (34.5") deep
Core Diameter:	11inches (28 cm)	
Flange Diameter:	24 inches (61 cm)	
Spool Width:	29 cm (11.5")	48 cm (19")
Weight:	22.25kg(49 lbs)	24.5kg (54 lbs)
Max. Spool Capacities: .320" Sonar	904 m (2965')	1720 m (5643')
.400" Camera	590 m (1935')	1109 m (3639')
.550" Stealth ROV	294 m (964')	555 m (1821')
Motor Drive:	Bi-Directional 1/2 HP 90VDC PM motor, 120VAC 450watts (240VAC Optional)	
Variable Speed:	Maximum 73m/min(240ft/min) Minimum 3.5m/min(12ft/min)	
Motor Drive Weight:	13.5 kg. (30 lb.)	
Bollard Pull:	45.3 kg (100 lb.) Adjustable	
Drive Hand Control unit:	One Hand operation, Fwd/Rev, Variable Speed	

Options:

Lifting Bar:	Provides a central lifting point for crane lifting
Drive Option:	Bi-Directional variable speed drive (specifications above)
Level wind:	Diamond Screw Level Wind Unit
Encoder:	measures cable pay out, stand alone display or video overlay

Features

- High-performance, Low-power AVR® 8-bit Microcontroller
- Advanced RISC Architecture
  - 131 Powerful Instructions – Most Single-clock Cycle Execution
  - 32 x 8 General Purpose Working Registers
  - Fully Static Operation
  - Up to 16 MIPS Throughput at 16 MHz
  - On-chip 2-cycle Multiplier
- Nonvolatile Program and Data Memories
  - 16K Bytes of In-System Self-Programmable Flash
    Endurance: 10,000 Write/Erase Cycles
  - Optional Boot Code Section with Independent Lock Bits
    In-System Programming by On-chip Boot Program
    True Read-While-Write Operation
  - 512 Bytes EEPROM
    Endurance: 100,000 Write/Erase Cycles
  - 1K Byte Internal SRAM
  - Programming Lock for Software Security
- JTAG (IEEE std. 1149.1 Compliant) Interface
  - Boundary-scan Capabilities According to the JTAG Standard
  - Extensive On-chip Debug Support
  - Programming of Flash, EEPROM, Fuses, and Lock Bits through the JTAG Interface
- Peripheral Features
  - Two 8-bit Timer/Counters with Separate Prescalers and Compare Modes
  - One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
  - Real Time Counter with Separate Oscillator
  - Four PWM Channels
  - 8-channel, 10-bit ADC
    8 Single-ended Channels
    7 Differential Channels in TQFP Package Only
    2 Differential Channels with Programmable Gain at 1x, 10x, or 200x
  - Byte-oriented Two-wire Serial Interface
  - Programmable Serial USART
  - Master/Slave SPI Serial Interface
  - Programmable Watchdog Timer with Separate On-chip Oscillator
  - On-chip Analog Comparator
- Special Microcontroller Features
  - Power-on Reset and Programmable Brown-out Detection
  - Internal Calibrated RC Oscillator
  - External and Internal Interrupt Sources
  - Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby and Extended Standby
- I/O and Packages
  - 32 Programmable I/O Lines
  - 40-pin PDIP, 44-lead TQFP, and 44-pad QFN/MLF
- Operating Voltages
  - 2.7 - 5.5V for ATmega16L
  - 4.5 - 5.5V for ATmega16
- Speed Grades
  - 0 - 8 MHz for ATmega16L
  - 0 - 16 MHz for ATmega16
- Power Consumption @ 1 MHz, 3V, and 25°C for ATmega16L
  - Active: 1.1 mA
  - Idle Mode: 0.35 mA
  - Power-down Mode: < 1 µA

8-bit AVR® Microcontroller with 16K Bytes In-System Programmable Flash

ATmega16
ATmega16L

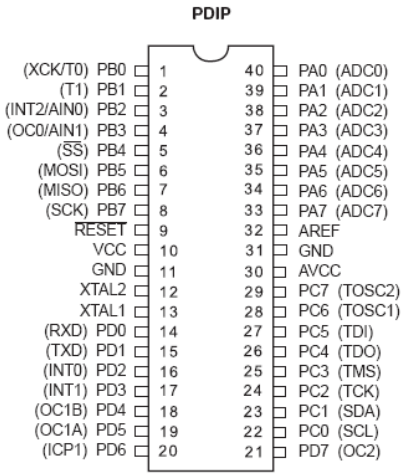
Summary

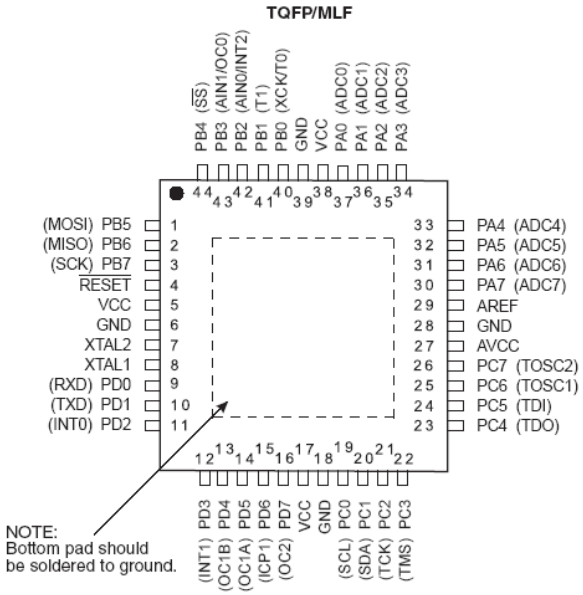
2466KS–AVR–04/05

Note: This is a summary document. A complete document is available on our Web site at www.atmel.com.

Pin Configurations

Figure 1. Pinout ATmega16





Disclaimer

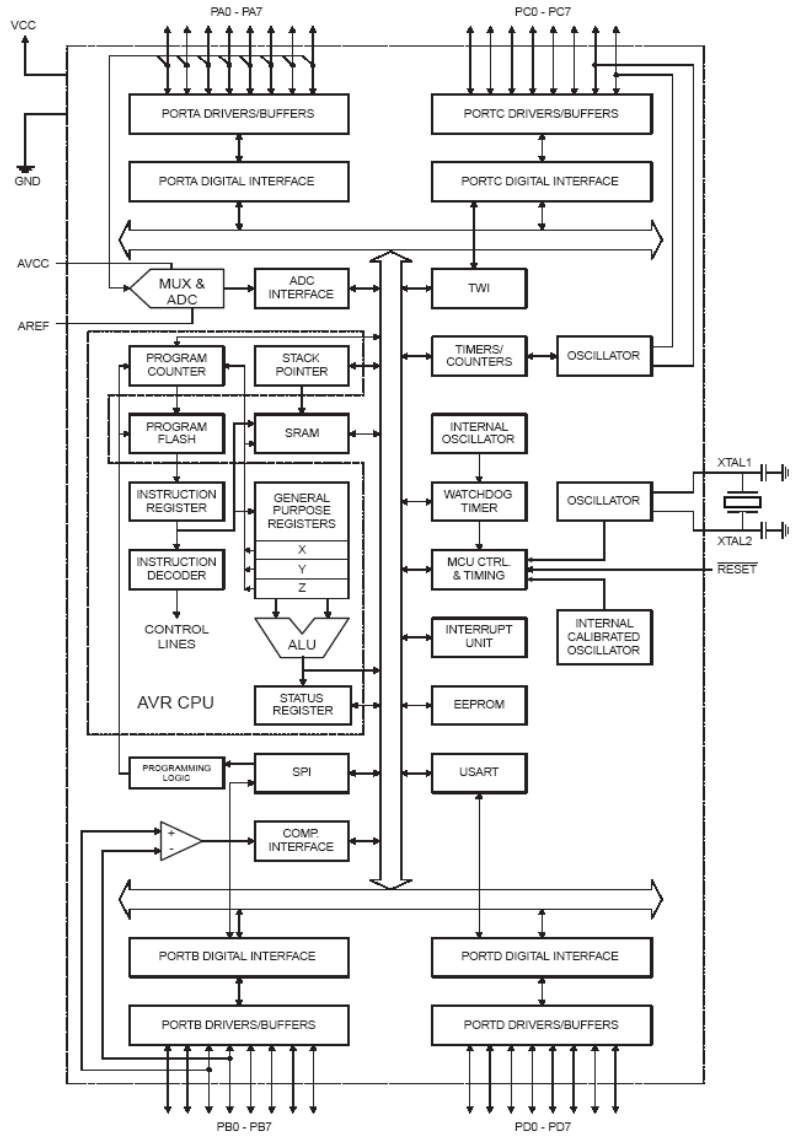
Typical values contained in this datasheet are based on simulations and characterization of other AVR microcontrollers manufactured on the same process technology. Min and Max values will be available after the device is characterized.

Overview

The ATmega16 is a low-power CMOS 8-bit microcontroller based on the AVR enhanced RISC architecture. By executing powerful instructions in a single clock cycle, the ATmega16 achieves throughputs approaching 1 MIPS per MHz allowing the system designer to optimize power consumption versus processing speed.

Block Diagram

Figure 2. Block Diagram



4.2.7 Recommended Minimum Specific GPS/TRANSIT Data (RMC)

$GPRMC,<1>,<2>,<3>,<4>,<5>,<6>,<7>,<8>,<9>,<10>,<11>,<12>*hh<CR><LF>

<1>	UTC time of position fix, hhmmss format
<2>	Status, A = Valid position, V = NAV receiver warning
<3>	Latitude, ddmm.mmmm format (leading zeros will be transmitted)
<4>	Latitude hemisphere, N or S
<5>	Longitude, dddmm.mmmm format (leading zeros will be transmitted)
<6>	Longitude hemisphere, E or W
<7>	Speed over ground, 000.0 to 999.9 knots (leading zeros will be transmitted)
<8>	Course over ground, 000.0 to 359.9 degrees, true (leading zeros will be transmitted)
<9>	UTC date of position fix, ddmmyy format
<10>	Magnetic variation, 000.0 to 180.0 degrees (leading zeros will be transmitted)
<11>	Magnetic variation direction, E or W (westerly variation adds to true course)
<12>	Mode indicator (only output if NMEA 0183 version 2.30 active), A = Autonomous, D = Differential, E = Estimated, N = Data not valid

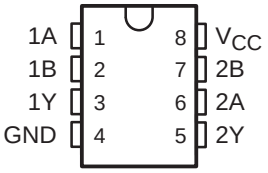
PERIPHERAL DRIVERS FOR HIGH-VOLTAGE, HIGH-CURRENT DRIVER APPLICATIONS

- **Characterized for Use to 300 mA**
- **High-Voltage Outputs**
- **No Output Latch-Up at 30 V (After Conducting 300 mA)**
- **Medium-Speed Switching**
- **Circuit Flexibility for Varied Applications and Choice of Logic Function**
- **TTL-Compatible Diode-Clamped Inputs**
- **Standard Supply Voltages**
- **Plastic DIP (P) With Copper Lead Frame for Cooler Operation and Improved Reliability**
- **Package Options Include Plastic Small Outline Packages, Ceramic Chip Carriers, and Standard Plastic and Ceramic 300-mil DIPs**

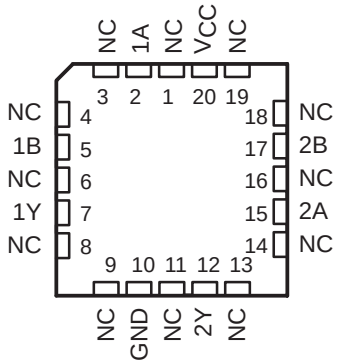
SUMMARY OF SERIES 55461/75461

DEVICE	LOGIC	PACKAGES
SN55461	AND	FK, JG
SN55462	NAND	FK, JG
SN55463	OR	FK, JG
SN75461	AND	D, P
SN75462	NAND	D, P
SN75463	OR	D, P

SN55461, SN55462, SN55463 . . . JG PACKAGE
SN75461, SN75462, SN75463 . . . D OR P PACKAGE
(TOP VIEW)



SN55461, SN55462, SN55463 . . . FK PACKAGE
(TOP VIEW)



NC – No internal connection

description

These dual peripheral drivers are functionally interchangeable with SN55451B through SN55453B and SN75451B through SN75453B peripheral drivers, but are designed for use in systems that require higher breakdown voltages than those devices can provide at the expense of slightly slower switching speeds. Typical applications include logic buffers, power drivers, relay drivers, lamp drivers, MOS drivers, line drivers, and memory drivers.

The SN55461/SN75461, SN55462/SN75462, and SN55463/SN75463 are dual peripheral AND, NAND, and OR drivers respectively (assuming positive logic), with the output of the gates internally connected to the bases of the npn output transistors.

Series SN55461 drivers are characterized for operation over the full military temperature range of –55°C to 125°C. Series SN75461 drivers are characterized for operation from 0°C to 70°C.

absolute maximum ratings over operating free-air temperature range (unless otherwise noted)†

		SN55'	SN75'	UNIT
Supply voltage, V_{CC} (see Note 1)		7	7	V
Input voltage, V_I		5.5	5.5	V
Intermitter voltage (see Note 2)		5.5	5.5	V
Off-state output voltage, V_O		35	35	V
Continuous collector or output current (see Note 3)		400	400	mA
Peak collector or output current ($t_W \leq 10$ ms, duty cycle $\leq 50\%$, see Note 4)		500	500	mA
Continuous total power dissipation		See Dissipation Rating Table		
Operating free-air temperature range, T_A		–55 to 125	0 to 70	°C
Storage temperature range, T_{stg}		–65 to 150	–65 to 150	°C
Case temperature for 60 seconds, T_C	FK package	260		°C
Lead temperature 1,6 mm (1/16 inch) from case for 60 seconds	JG package	300		°C
Lead temperature 1,6 mm (1/16 inch) from case for 10 seconds	D or P package		260	°C

† Stresses beyond those listed under "absolute maximum ratings" may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those indicated under "recommended operating conditions" is not implied. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.

NOTES: 1. Voltage values are with respect to network GND unless otherwise specified.
2. This is the voltage between two emitters A and B.
3. This value applies when the base-emitter resistance (R_{BE}) is equal to or less than 500 Ω.
4. Both halves of these dual circuits may conduct rated current simultaneously; however, power dissipation averaged over a short time interval must fall within the continuous dissipation rating.

DISSIPATION RATING TABLE

PACKAGE	$T_A \leq 25°C$ POWER RATING	DERATING FACTOR ABOVE $T_A = 25°C$	$T_A = 70°C$ POWER RATING	$T_A = 125°C$ POWER RATING
D	725 mW	5.8 mW/°C	464 mW	–
FK	1375 mW	11.0 mW/°C	880 mW	275 mW
JG	1050 mW	8.4 mW/°C	672 mW	210 mW
P	1000 mW	8.0 mW/°C	640 mW	–

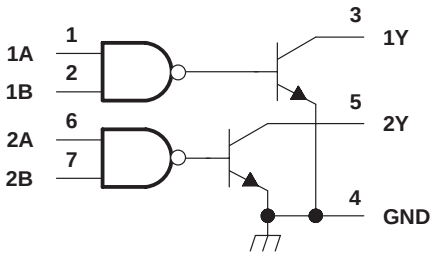
recommended operating conditions

	SN55'			SN75'			UNIT
	MIN	NOM	MAX	MIN	NOM	MAX	
Supply voltage, V_{CC}	4.5	5	5.5	4.75	5	5.25	V
High-level input voltage, V_{IH}	2			2			V
Low-level input voltage, V_{IL}			0.8			0.8	V
Operating free-air temperature, T_A	–55		125	0		70	°C

logic symbol†

† This symbol is in accordance with ANSI/IEEE Std 91-1984 and IEC Publication 617-12.
Pin numbers shown are for D, JG, and P packages.

logic diagram (positive logic)

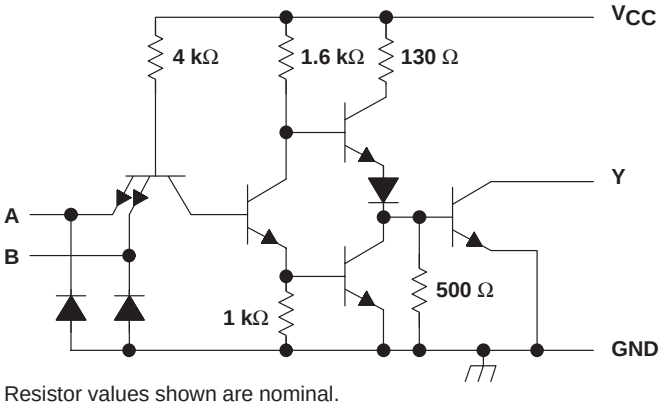


FUNCTION TABLE (each driver)

A	B	Y
L	L	L (on state)
L	H	L (on state)
H	L	L (on state)
H	H	H (off state)

positive logic: $Y = AB$ or $Y = \overline{A} + \overline{B}$

schematic (each driver)



Resistor values shown are nominal.

electrical characteristics over recommended operating free-air temperature range

PARAMETER		TEST CONDITIONS†	SN55461			SN75461			UNIT
			MIN	TYP‡	MAX	MIN	TYP‡	MAX	
V_{IK}	Input clamp voltage	V_{CC} = MIN, I_I = –12 mA		–1.2	–1.5		–1.2	–1.5	V
I_{OH}	High-level output current	V_{CC} = MIN, V_{IH} = MIN, V_{OH} = 35 V			300			100	µA
V_{OL}	Low-level output voltage	V_{CC} = MIN, V_{IL} = 0.8 V, I_{OL} = 100 mA		0.25	0.5		0.25	0.4	V
		V_{CC} = MIN, V_{IL} = 0.8 V, I_{OL} = 300 mA		0.5	0.8		0.5	0.7	
I_I	Input current at maximum input voltage	V_{CC} = MAX, V_I = 5.5 V			1			1	mA
I_{IH}	High-level input current	V_{CC} = MAX, V_I = 2.4 V			40			40	µA
I_{IL}	Low-level input current	V_{CC} = MAX, V_I = 0.4 V		–1	–1.6		–1	–1.6	mA
I_{CCH}	Supply current, outputs high	V_{CC} = MAX, V_I = 5 V		8	11		8	11	mA
I_{CCL}	Supply current, outputs low	V_{CC} = MAX, V_I = 0		56	76		56	76	mA

† For conditions shown as MIN or MAX, use the appropriate value specified under recommended operating conditions.
‡ All typical values are at V_{CC} = 5 V, T_A = 25°C.

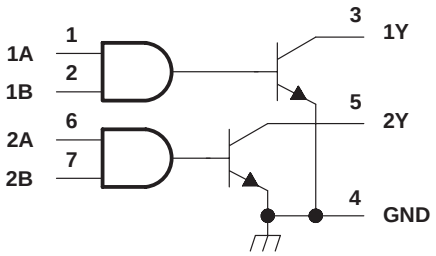
switching characteristics, V_{CC} = 5 V, T_A = 25°C

PARAMETER			TEST CONDITIONS		MIN	TYP	MAX	UNIT
t_{PLH}	Propagation delay time, low-to-high-level output		$I_O \approx$ 200 mA, R_L = 50 Ω,	C_L = 15 pF, See Figure 1		30	55	ns
t_{PHL}	Propagation delay time, high-to-low-level output					25	40	
t_{TLH}	Transition time, low-to-high-level output					8	20	
t_{THL}	Transition time, high-to-low-level output					10	20	
V_{OH}	High-level output voltage after switching	SN55461	V_S = 30 V, See Figure 2	$I_O \approx$ 300 mA,	V_S–10			mV
		SN75461			V_S–10			

logic symbol†

† This symbol is in accordance with ANSI/IEEE Std 91-1984 and IEC Publication 617-12.
Pin numbers shown are for D, JG, and P packages.

logic diagram (positive logic)

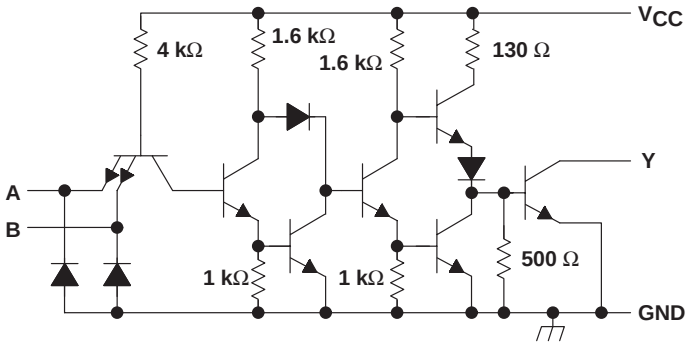


FUNCTION TABLE (each driver)

A	B	Y
L	L	H (off state)
L	H	H (off state)
H	L	H (off state)
H	H	L (on state)

positive logic:
$Y = \overline{AB}$ or $\overline{A} + \overline{B}$

schematic (each driver)



Resistor values shown are nominal.

electrical characteristics over recommended operating free-air temperature range

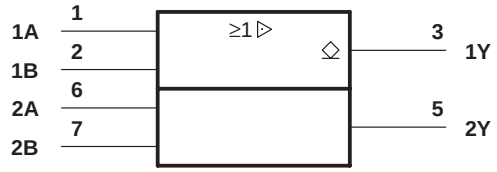
PARAMETER		TEST CONDITIONS†	SN55462			SN75462			UNIT
			MIN	TYP‡	MAX	MIN	TYP‡	MAX	
V_{IK}	Input clamp voltage	V_{CC} = MIN, I_I = –12 mA		–1.2	–1.5		–1.2	–1.5	V
I_{OH}	High-level output current	V_{CC} = MIN, V_{IL} = 0.8 V, V_{OH} = 35 V			300			100	µA
V_{OL}	Low-level output voltage	V_{CC} = MIN, V_{IH} = MIN, I_{OL} = 100 mA		0.25	0.5		0.25	0.4	V
		V_{CC} = MIN, V_{IH} = MIN, I_{OL} = 300 mA		0.5	0.8		0.5	0.7	
I_I	Input current at maximum input voltage	V_{CC} = MAX, V_I = 5.5 V			1			1	mA
I_{IH}	High-level input current	V_{CC} = MAX, V_I = 2.4 V			40			40	µA
I_{IL}	Low-level input current	V_{CC} = MAX, V_I = 0.4 V		–1.1	–1.6		–1.1	–1.6	mA
I_{CCH}	Supply current, outputs high	V_{CC} = MAX, V_I = 0		13	17		13	17	mA
I_{CCL}	Supply current, outputs low	V_{CC} = MAX, V_I = 5 V		61	76		61	76	mA

† For conditions shown as MIN or MAX, use the appropriate value specified under recommended operating conditions.
‡ All typical values are at V_{CC} = 5 V, T_A = 25°C.

switching characteristics, V_{CC} = 5 V, T_A = 25°C

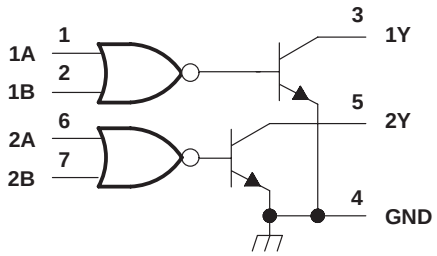
PARAMETER			TEST CONDITIONS		MIN	TYP	MAX	UNIT
t_{PLH}	Propagation delay time, low-to-high-level output		$I_O \approx$ 200 mA, R_L = 50 Ω,	C_L = 15 pF, See Figure 1		45	65	ns
t_{PHL}	Propagation delay time, high-to-low-level output					30	50	
t_{TLH}	Transition time, low-to-high-level output					13	25	
t_{THL}	Transition time, high-to-low-level output					10	20	
V_{OH}	High-level output voltage after switching	SN55462	V_S = 30 V, See Figure 2	$I_O \approx$ 300 mA,	V_S–10			mV
		SN75462			V_S–10			

logic symbol†



† This symbol is in accordance with ANSI/IEEE Std 91-1984 and IEC Publication 617-12.
Pin numbers shown are for D, JG, and P packages.

logic diagram (positive logic)

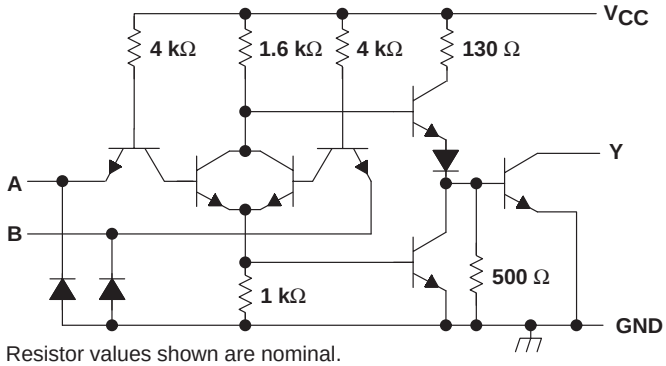


FUNCTION TABLE (each driver)

A	B	Y
L	L	L (on state)
L	H	H (off state)
H	L	H (off state)
H	H	H (off state)

positive logic:
$Y = A + B$ or $\overline{A}\,\overline{B}$

schematic (each driver)



Resistor values shown are nominal.

electrical characteristics over recommended operating free-air temperature range

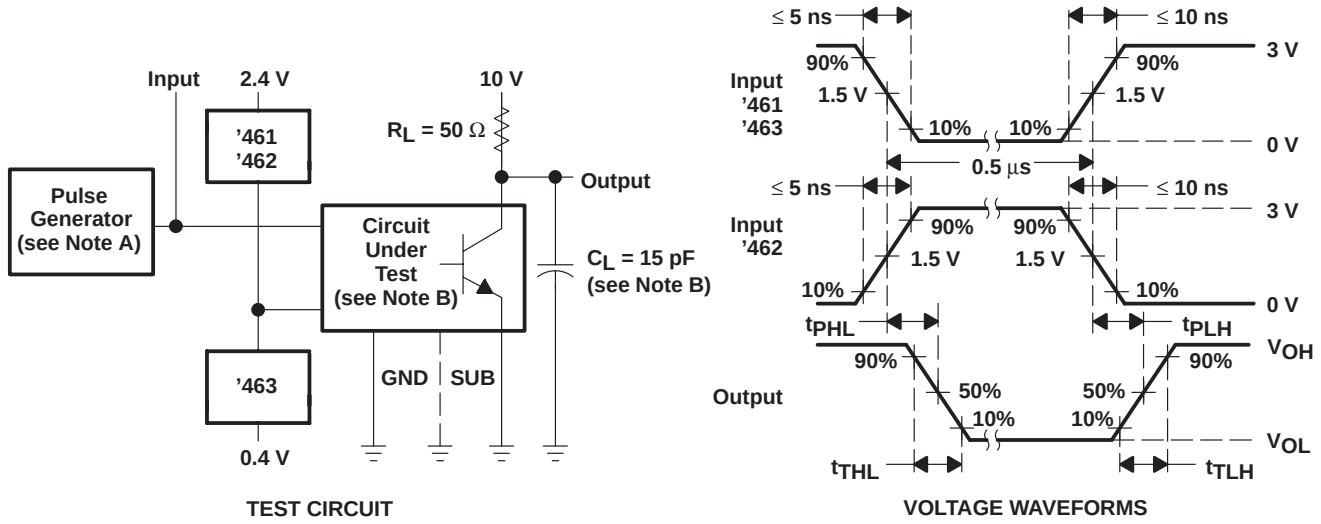
PARAMETER		TEST CONDITIONS†	SN55463			SN75463			UNIT
			MIN	TYP‡	MAX	MIN	TYP‡	MAX	
V_{IK}	Input clamp voltage	V_{CC} = MIN, I_I = –12 mA		–1.2	–1.5		–1.2	–1.5	V
I_{OH}	High-level output current	V_{CC} = MIN, V_{IH} = MIN, V_{OH} = 35 V			300			100	μA
V_{OL}	Low-level output voltage	V_{CC} = MIN, V_{IL} = 0.8 V, I_{OL} = 100 mA		0.25	0.5		0.25	0.4	V
		V_{CC} = MIN, V_{IL} = 0.8 V, I_{OL} = 300 mA		0.5	0.8		0.5	0.7	
I_I	Input current at maximum input voltage	V_{CC} = MAX, V_I = 5.5 V			1			1	mA
I_{IH}	High-level input current	V_{CC} = MAX, V_I = 2.4 V			40			40	μA
I_{IL}	Low-level input current	V_{CC} = MAX, V_I = 0.4 V		–1	–1.6		–1	–1.6	mA
I_{CCH}	Supply current, outputs high	V_{CC} = MAX, V_I = 5 V		8	11		8	11	mA
I_{CCL}	Supply current, outputs low	V_{CC} = MAX, V_I = 0		58	76		58	76	mA

† For conditions shown as MIN or MAX, use the appropriate value specified under recommended operating conditions.
‡ All typical values are at V_{CC} = 5 V, T_A = 25°C.

switching characteristics, V_{CC} = 5 V, T_A = 25°C

PARAMETER			TEST CONDITIONS	MIN	TYP	MAX	UNIT
t_{PLH}	Propagation delay time, low-to-high-level output		$I_O \approx$ 200 mA, C_L = 15 pF, R_L = 50 Ω, See Figure 1		30	55	ns
t_{PHL}	Propagation delay time, high-to-low-level output				25	40	
t_{TLH}	Transition time, low-to-high-level output				8	25	
t_{THL}	Transition time, high-to-low-level output				10	25	
V_{OH}	High-level output voltage after switching	SN55463	V_S = 30 V, $I_O \approx$ 300 mA, See Figure 2	V_S–10			mV
		SN75463		V_S–10			

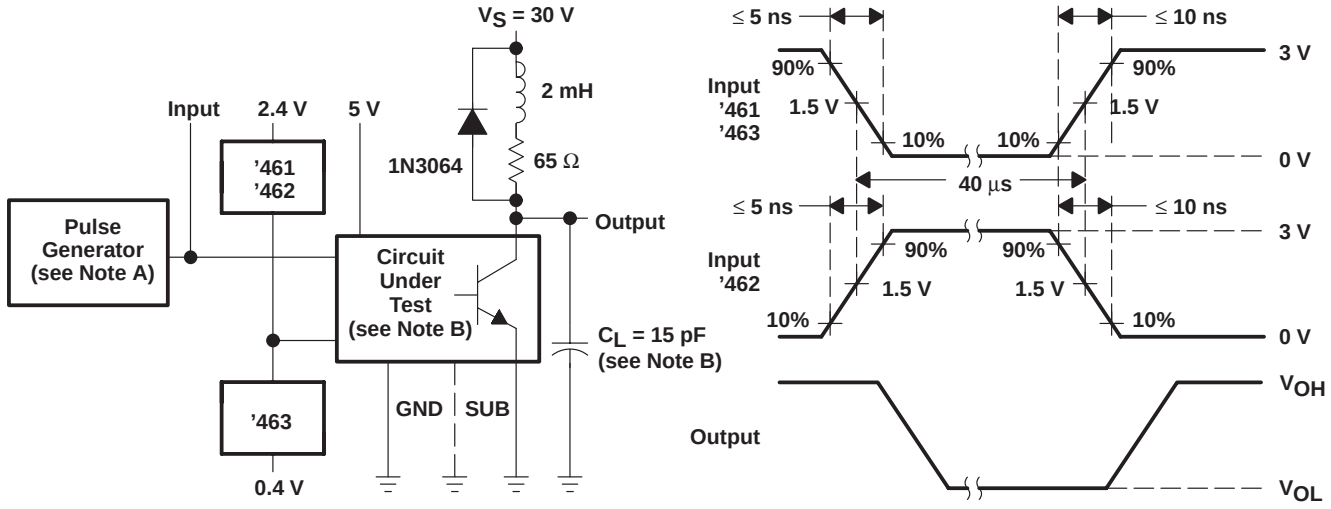
PARAMETER MEASUREMENT INFORMATION



TEST CIRCUIT VOLTAGE WAVEFORMS

NOTES: A. The pulse generator has the following characteristics: PRR ≤ 1 MHz, $Z_O \approx 50\ \Omega$.
B. C_L includes probe and jig capacitance.

Figure 1. Test Circuit and Voltage Waveforms for Switching Times



TEST CIRCUIT VOLTAGE WAVEFORMS

NOTES: A. The pulse generator has the following characteristics: PRR ≤ 12.5 kHz, $Z_O = 50\ \Omega$.
B. C_L includes probe and jig capacitance.

Figure 2. Test Circuit and Voltage Waveforms for Latch-Up Test

PACKAGING INFORMATION

Orderable Device	Status (1)	Package Type	Package Drawing	Pins	Package Qty	Eco Plan (2)	Lead/Ball Finish	MSL Peak Temp (3)
JM38510/12908BPA	ACTIVE	CDIP	JG	8	1	TBD	A42 SNPB	Level-NC-NC-NC
JM38510/12909BPA	OBSOLETE	CDIP	JG	8		TBD	Call TI	Call TI
SN55461JG	OBSOLETE	CDIP	JG	8		TBD	Call TI	Call TI
SN55462JG	OBSOLETE	CDIP	JG	8		TBD	Call TI	Call TI
SN55463JG	OBSOLETE	CDIP	JG	8		TBD	Call TI	Call TI
SN75461D	OBSOLETE	SOIC	D	8		TBD	Call TI	Call TI
SN75461P	OBSOLETE	PDIP	P	8		TBD	Call TI	Call TI
SN75462D	ACTIVE	SOIC	D	8	75	Pb-Free (RoHS)	CU NIPDAU	Level-2-260C-1 YEAR/ Level-1-235C-UNLIM
SN75462DR	ACTIVE	SOIC	D	8	2500	Pb-Free (RoHS)	CU NIPDAU	Level-2-260C-1 YEAR/ Level-1-235C-UNLIM
SN75462P	ACTIVE	PDIP	P	8	50	Pb-Free (RoHS)	CU NIPDAU	Level-NC-NC-NC
SN75463D	OBSOLETE	SOIC	D	8		TBD	Call TI	Call TI
SN75463DR	OBSOLETE	SOIC	D	8		TBD	Call TI	Call TI
SN75463P	ACTIVE	PDIP	P	8	50	Pb-Free (RoHS)	CU NIPDAU	Level-NC-NC-NC
SNJ55461FK	OBSOLETE	LCCC	FK	20		TBD	Call TI	Call TI
SNJ55461JG	OBSOLETE	CDIP	JG	8		TBD	Call TI	Call TI
SNJ55462FK	ACTIVE	LCCC	FK	20	1	TBD	POST-PLATE	Level-NC-NC-NC
SNJ55462JG	ACTIVE	CDIP	JG	8	1	TBD	A42 SNPB	Level-NC-NC-NC
SNJ55463JG	OBSOLETE	CDIP	JG	8		TBD	Call TI	Call TI

(1) The marketing status values are defined as follows:
ACTIVE: Product device recommended for new designs.
LIFEBUY: TI has announced that the device will be discontinued, and a lifetime-buy period is in effect.
NRND: Not recommended for new designs. Device is in production to support existing customers, but TI does not recommend using this part in a new design.
PREVIEW: Device has been announced but is not in production. Samples may or may not be available.
OBSOLETE: TI has discontinued the production of the device.

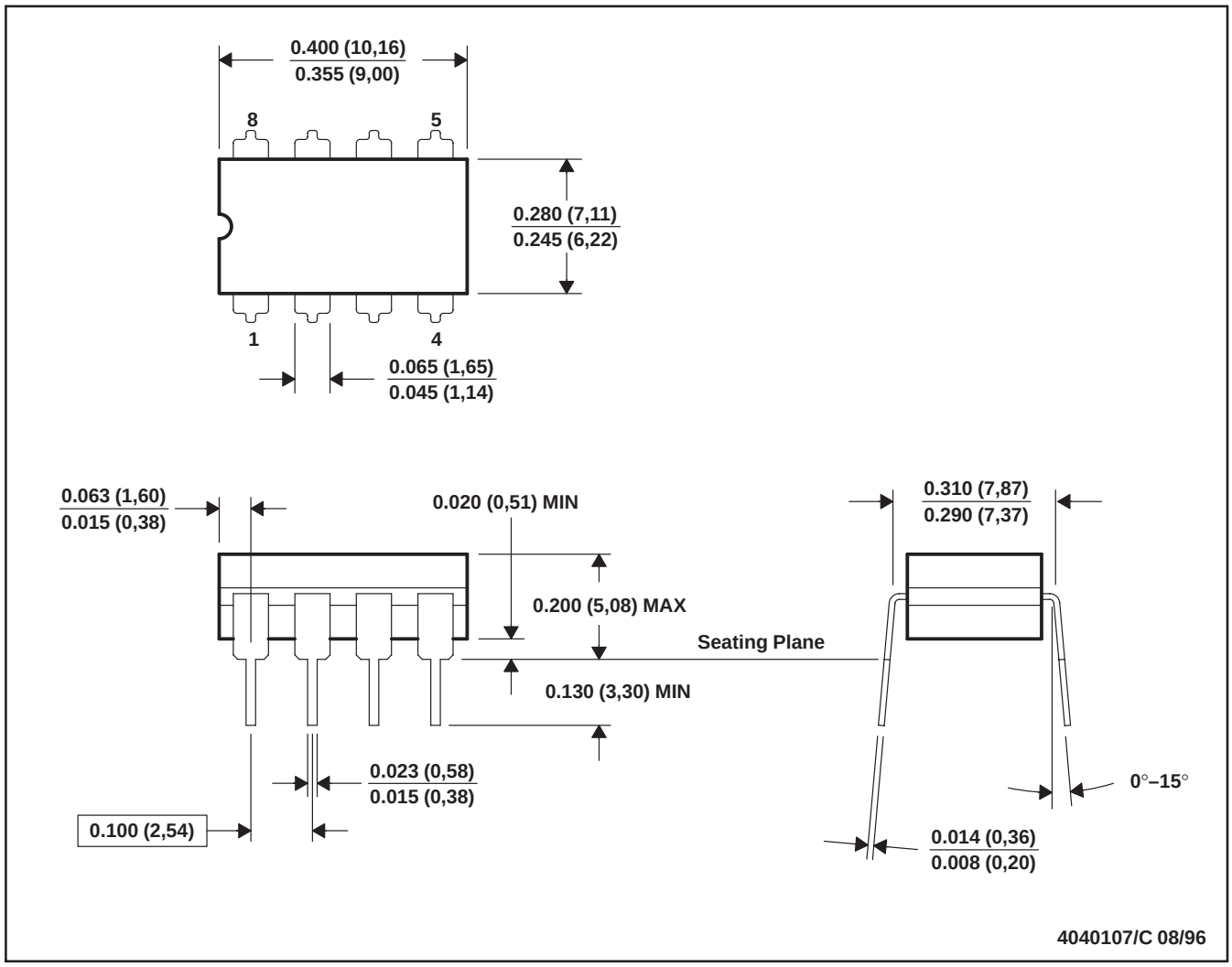
(2) Eco Plan - The planned eco-friendly classification: Pb-Free (RoHS) or Green (RoHS & no Sb/Br) - please check http://www.ti.com/productcontent for the latest availability information and additional product content details.
TBD: The Pb-Free/Green conversion plan has not been defined.
Pb-Free (RoHS): TI's terms "Lead-Free" or "Pb-Free" mean semiconductor products that are compatible with the current RoHS requirements for all 6 substances, including the requirement that lead not exceed 0.1% by weight in homogeneous materials. Where designed to be soldered at high temperatures, TI Pb-Free products are suitable for use in specified lead-free processes.
Green (RoHS & no Sb/Br): TI defines "Green" to mean Pb-Free (RoHS compatible), and free of Bromine (Br) and Antimony (Sb) based flame retardants (Br or Sb do not exceed 0.1% by weight in homogeneous material)

(3) MSL, Peak Temp. -- The Moisture Sensitivity Level rating according to the JEDEC industry standard classifications, and peak solder temperature.

to Customer on an annual basis.

MCER001A – JANUARY 1995 – REVISED JANUARY 1997

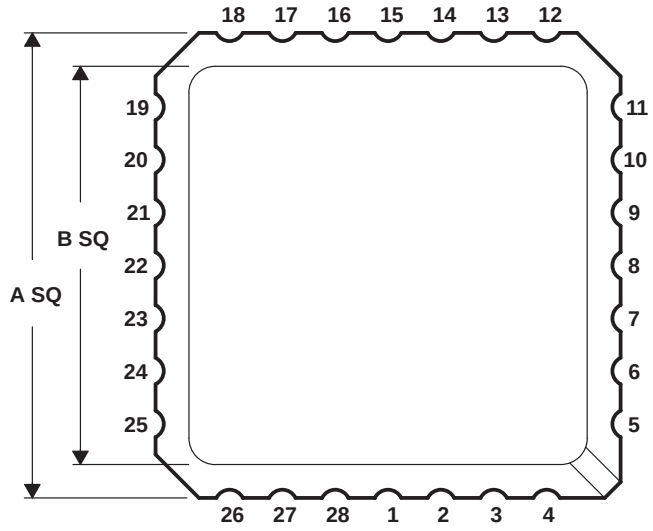
JG (R-GDIP-T8) **CERAMIC DUAL-IN-LINE**



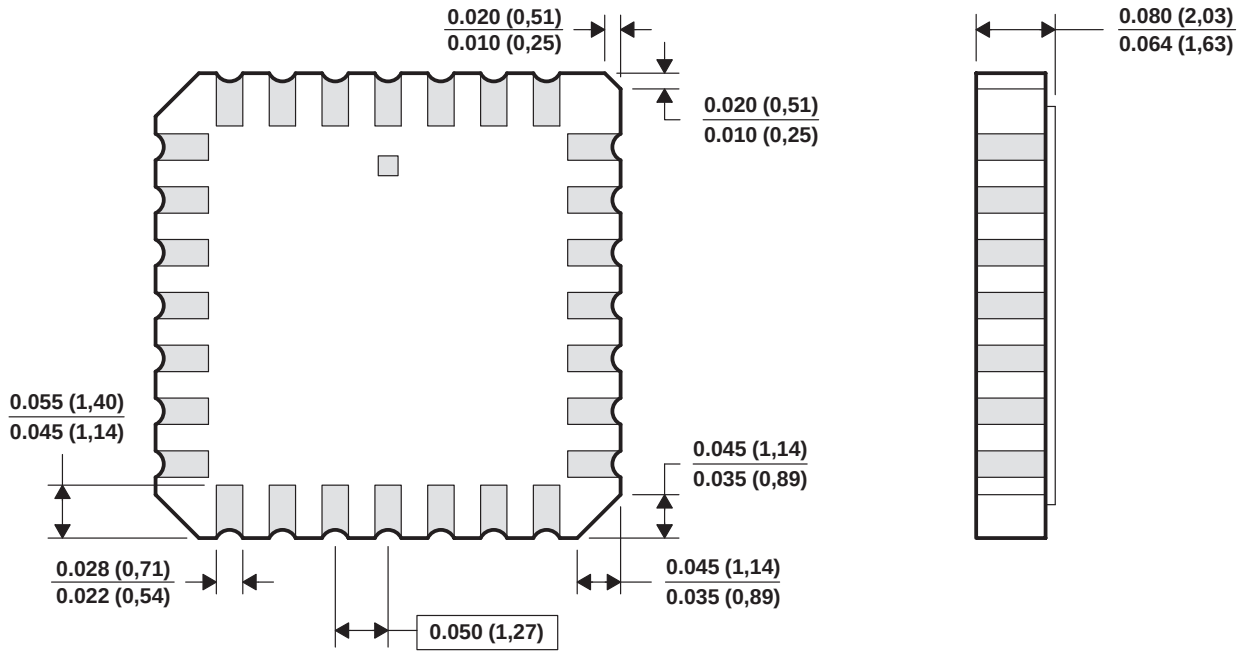
NOTES: A. All linear dimensions are in inches (millimeters).
B. This drawing is subject to change without notice.
C. This package can be hermetically sealed with a ceramic lid using glass frit.
D. Index point is provided on cap for terminal identification.
E. Falls within MIL STD 1835 GDIP1-T8

FK (S-CQCC-N)** **LEADLESS CERAMIC CHIP CARRIER**

28 TERMINAL SHOWN



NO. OF TERMINALS **	A		B	
	MIN	MAX	MIN	MAX
20	0.342 (8,69)	0.358 (9,09)	0.307 (7,80)	0.358 (9,09)
28	0.442 (11,23)	0.458 (11,63)	0.406 (10,31)	0.458 (11,63)
44	0.640 (16,26)	0.660 (16,76)	0.495 (12,58)	0.560 (14,22)
52	0.739 (18,78)	0.761 (19,32)	0.495 (12,58)	0.560 (14,22)
68	0.938 (23,83)	0.962 (24,43)	0.850 (21,6)	0.858 (21,8)
84	1.141 (28,99)	1.165 (29,59)	1.047 (26,6)	1.063 (27,0)

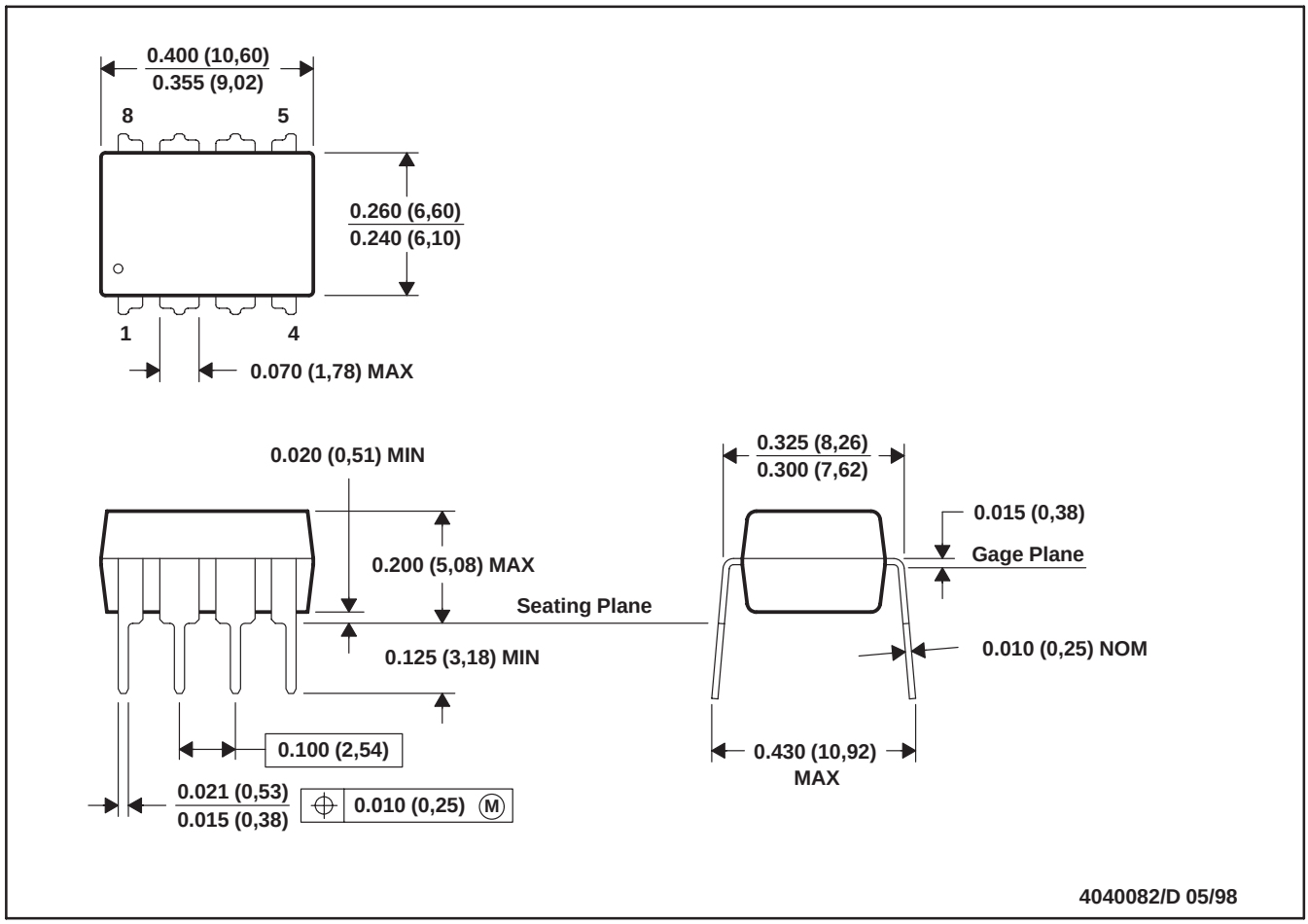


4040140/D 10/96

NOTES: A. All linear dimensions are in inches (millimeters).
B. This drawing is subject to change without notice.
C. This package can be hermetically sealed with a metal lid.
D. The terminals are gold plated.
E. Falls within JEDEC MS-004

MPDI001A – JANUARY 1995 – REVISED JUNE 1999

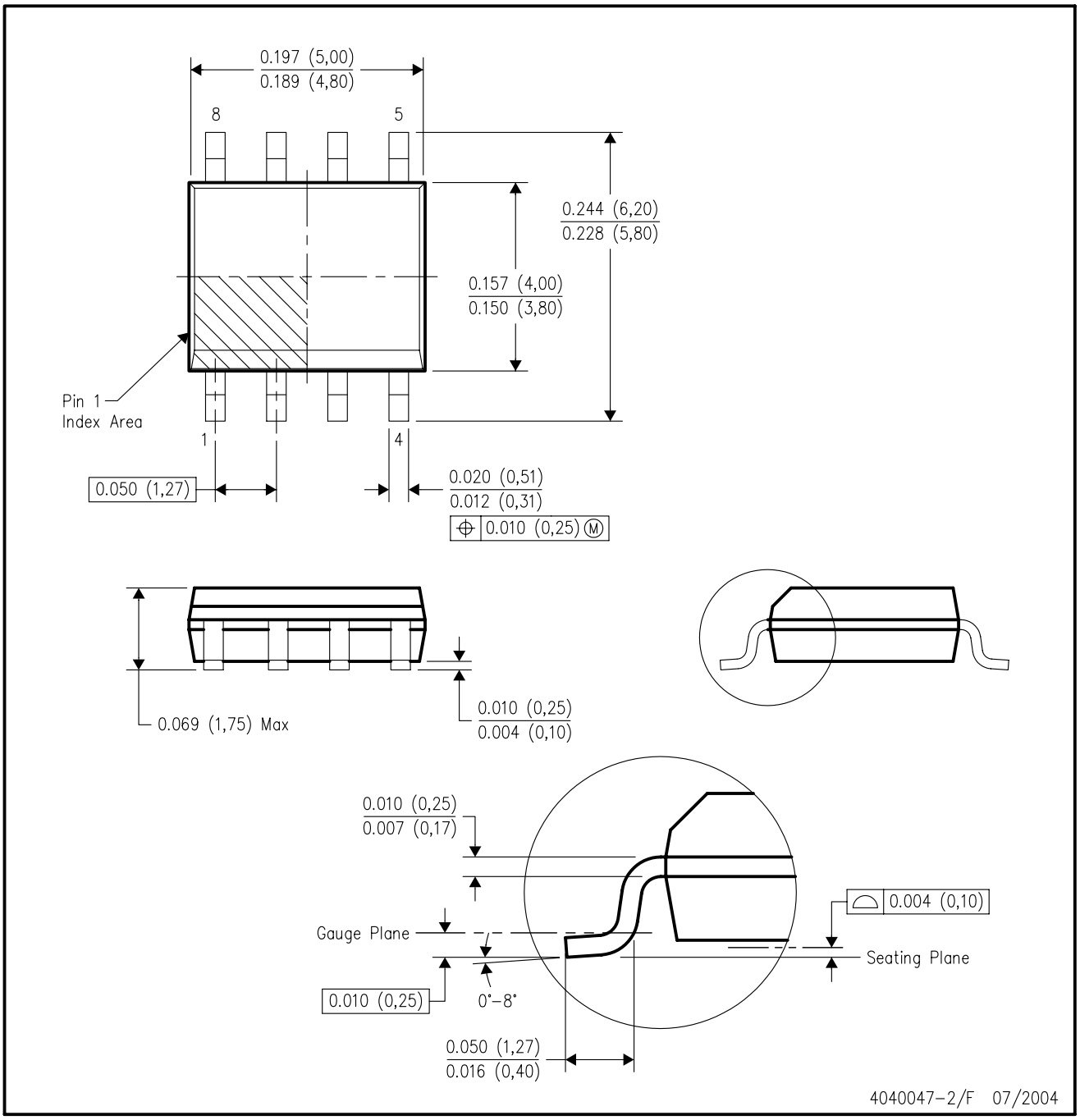
P (R-PDIP-T8) **PLASTIC DUAL-IN-LINE**



NOTES: A. All linear dimensions are in inches (millimeters).
B. This drawing is subject to change without notice.
C. Falls within JEDEC MS-001

For the latest package information, go to http://www.ti.com/sc/docs/package/pkg_info.htm

D (R–PDSO–G8) PLASTIC SMALL–OUTLINE PACKAGE



NOTES:
A. All linear dimensions are in inches (millimeters).
B. This drawing is subject to change without notice.
C. Body dimensions do not include mold flash or protrusion not to exceed 0.006 (0,15).
D. Falls within JEDEC MS–012 variation AA.

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Interface	interface.ti.com	Digital Control	www.ti.com/digitalcontrol
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
		Telephony	www.ti.com/telephony
		Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

Mailing Address: Texas Instruments
Post Office Box 655303 Dallas, Texas 75265

March 2005

PNI – TCM3

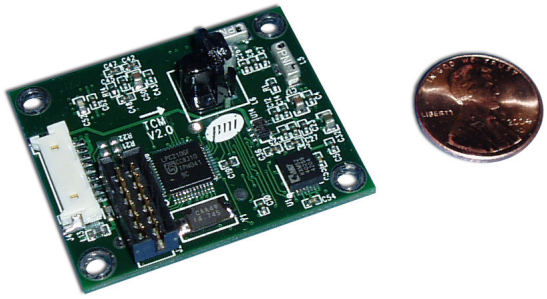
Tilt Compensated 3- axis Compass Module

General Description

The TCM3 uses advanced algorithms, with hard iron and soft iron corrections, to provide highly accurate heading information over its extra wide tilt range at latitudes up to 85°. The small size and low power requirements of the TCM3 allow it to fit into today's size sensitive systems. These advantages make PNI Corporation's TCM3 the choice for applications that require highest accuracy and performance anywhere in the world.

Features include an extra wide tilt range, high accuracy compass heading, low power consumption, large signal noise immunity under all conditions, and a large magnetic field measurement range in a small package.

The TCM3 combines PNI Corporation's patented Magneto-Inductive (MI) sensors and measurement circuit technology with a 3-axis MEMS accelerometer for unparalleled cost effectiveness and performance. The magnetic sensors and accelerometers are calibrated to operate from -40 to 85°C; hence the measurement is very stable over temperature and inherently free from offset drift.



Features

- High accuracy compass heading: 0.5°
- High resolution compass heading: 0.1°
- High repeatability: 0.05°
- Extra wide tilt range: ± 80°
- Multiple measurement modes: Compass heading, magnetic field and 2-axis tilt
- Calibrated field measurement range: ±80 µT (± 0.8 Gauss)
- High resolution field measurement: 0.05µT (0.0005 Gauss)
- Advanced calibration: Hard and Soft Iron
- Small size: 3.5 x 4.3 x 1.3 cm
- Multiple digital interfaces: RS-232, SPI, I2C

Applications

- High performance ROV navigation
- GPS system integration
- Vehicle tracking
- Remote terrestrial antenna direction indicators
- Sonar targeting systems
- 3-axis magnetic field sensing
- Survey equipment

……*Ordering Information*

NAME	PART NUMBER	Package
TCM3	12409	Each
TCM3 w/Cable and CD	90013	Each

For more information, please call PNI Corporation direct at (707) 566-2260, email: sales@pnicorp.com, or visit PNI's website at http://www.pnicorp.com

PNI – TCM3 Tilt Compensated 3-axis Compass Module

Parameter	Typical	Units
Heading Specifications		
Accuracy with <70° of tilt	0.5°	Deg RMS
Accuracy with >70° of tilt	0.8°	
Resolution	0.1°	
Repeatability [1]	0.05°	
Max Dip Angle	85°	Deg
Magnetometer Specifications		
Calibrated Field Measurement Range	± 0.8	Gauss
Magnetic Resolution	± 0.5 mGauss	mGauss
Magnetic Repeatability	± 1.0 mGauss	
Tilt Specifications		
Pitch Accuracy	0.2°	Deg RMS
Roll Accuracy	0.2° for Pitch < 65º 0.5° for Pitch < 80º 1.0° for Pitch < 86º	
Tilt Range	± 80°	Deg
Tilt Resolution	< 0.01°	
Tilt Repeatability [1]	0.05°	Deg RMS
Calibration		
Hard Iron Calibration	Yes	
Soft Iron Calibration	Yes	
Limited Tilt User Calibration	Yes	
Mechanical Specifications		
Dimensions (L x W x H)	3.5 x 4.3 x 1.3	cm
Weight	12	grams
Mounting Options	Screw mounts/standoffs; horizontal	
Connector for RS-232	9-pin	
I/O Specifications		
Latency from Power-On	< 50	mSec
Latency from Sleep Mode	< 1	
Maximum Sample Rate	20	samples/sec
RS-232 Communication Rate	300 to 115200	baud
Output Formats	Binary High Performance Protocol	
Power Specifications		
Supply Voltage	3.6 to 5 V (unregulated)	VDC
Current Draw **Max.**	22	mA
(Continuous Output) **Typ.**	< 20	
Idle Mode [2]	14 - 18	
Power Up (first 400 mSec)	< 55	
Sleep Mode	0.6	
Environmental Specifications		
Operating Temperature	– 40 to 85	℃
Storage Temperature	– 40 to 125	
Shock	50 – 2500 G's, Half Sine Wave Shock with 2 drops at each level	
Vibration	Z-Axis, Skewed Block, at 1, 2 & 4 Grms @ 10 – 1000 KHz for 30 min. per level	
Humidity	70C with 95% R.H. for 168 hrs	

[1]Repeatability is based on statistical data at ±3 sigma limit about the mean.
[2]Based on User settings.

9XTend-PKG™ RF Modems

1 Watt - 900 MHz - Stand-alone Radio Modems by Ma

Long Range Performance

Indoor/urban Range:	up to 3000' (900 m)
Outdoor line-of-sight Range:	up to 40 miles (64 km)
Transmit Power Output:	1mW - 1W (software selectable) up to 4 Watts EIRP w/6 dB antenna
Receiver Sensitivity:	-110 dBm (@9600 bps)
Throughput Data Rate:	9600 or 115200 bps (software selectable)

Advanced Networking & Security

Encryption:	256-bit AES
Spread Spectrum Type:	FHSS (Frequency Hopping Spread Spectrum)

Streaming, Acknowledged & Multi-Transmit modes are supported

Easy-to-Use

No configuration is necessary for out-of-box RF operation. Simply feed data into one RF modem; then the data is sent out the other end of the wireless link.

If more advanced functionality is needed, the modems support an extensive set of AT and binary commands.

Interfaces Available

9XTend-PKG-R™
RS-232/485

9XTend-PKG-U™
USB

9XTend-PKG-E™
Ethernet

9XTend-NEMA™
Weatherproof Enclosure Now Available

Highlights

Price-to-Performance Value.
Due to innovations stamped in its design, the 9XTend-PKGs yield 2-8x the range of competing modems. This allows OEMs & integrators to cover more ground with fewer devices.

Additionally, 9XTend Modems are easy-to-use and therefore help reduce the cost of data system development.

256-bit AES Encryption.
The 9XTend provides security through data encryption that is not available on competing modems. The Advanced Encryption Standard (AES) is used with a 256-bit key.

No time penalty is incurred during encryption or decryption.

Receiver Sensitivity.
MaxStream modems 'hear' what others cannot; therefore supplying greater range and reliability in wireless links.

XTend Modems outperform higher costing modems due in large part to range gained through superior RX sensitivity.

Transmit Power Output.
The 9XTend outputs 1 Watt (30 dBm) of conducted power while consuming only 730 mA current (5V power supply).

FCC (U.S.A.) & IC (Canada) Approved.
Contact MaxStream for complete list of certifications.

Sample Applications

Monitoring of remote systems

Sensor data capture in embedded systems

Home automation & building control

SCADA (Supervisory control & data acquisition)

Fleet management & asset tracking

Call today for:
- Free RF Consultation
- Volume Discounts
- Development Kit Pricing

MaxStream®
(866) 765-9885 toll-free in U.S. & Canada
(801) 765-9885 worldwide
www.maxstream.net

9XTend-PKG™ 900 MHz RF Modems

Specifications		9XTend-PKG-R™ (RS-232/485)	9XTend-PKG-U™ (USB)	9XTend-PKG-E™ (Ethernet)	9XTend-NEMA™ (Weatherproof)
Performance	Transmit Power Output (software selectable)	1 mW - 1 W (0 - 30 dBm)			
	Indoor/Urban Range (w/ 2.1 dB dipole antenna)	up to 3000' (900 m)			
	Outdoor RF line-of-sight Range (w/ 2.1 dB dipole antenna)	up to 14 miles (22 km)			
	Outdoor RF line-of-sight Range (w/ high gain antenna)	up to 40 miles (64 km)			
	Interface Data Rate (software selectable)	10 - 230400 bps (including non-standard baud rates)			
	Throughput Data Rate (software selectable)	9,600 or 115,200 bps			
	RF Data Rate	10,000 bps (@9,600 bps Throughput Data Rate) 125,000 bps (@115,200 bps)			
	Receiver Sensitivity	-110 dBm (@9,600 bps Throughput Data Rate) -100 dBm (@115,200 bps)			
Networking & Security	Frequency	ISM 902 - 928 MHz			
	Spread Spectrum	FHSS (Frequency Hopping Spread Spectrum)			
	Modulation	FSK (Frequency Shift Keying)			
	Supported Network Topologies	Peer-to-peer (no master/slave dependencies), Point-to-point, Point-to-multipoint & Multidrop			
	Channel Capacity	10 hop sequences share 50 frequencies			
	Encryption	256-bit AES Encryption			
Antenna	Connector	RPSMA (reverse polarity SMA)			
	Impedance	50 ohms unbalanced			
Certifications (partial list)	FCC Part 15.247	OUR-9XTEND			
	Industry Canada (IC)	4214A-9XTEND			

		9XTend-PKG-R™	9XTend-PKG-U™	9XTend-PKG-E™	9XTend-NEMA™
Power Requirements	Power Supply Voltage	7 - 28 V	7 - 28 V	7 - 28 V	7 - 28 V
	Receive Current	110 mA	100 mA (Self Power)	110 mA	110 mA
	Pin Sleep Power-down	17 mA	17 mA	17 mA	17 mA
	Serial Port Sleep Power-down	45 mA	45 mA	45 mA	45 mA
	Idle Current (Various Cyclic Sleep Intervals)	19 - 39 mA	21 - 35 mA (Self Power)	19 - 39 mA	19 - 39 mA
	Transmit Current (1mW - 1W TX Power Output)	110 - 900 mA	88 - 480 mA	110 - 900 mA	110 - 900 mA
Physical Properties	Size	2.750" x 5.500" x 1.125" (6.99cm x 13.97cm x 2.86cm)			5.125" x 7.125" x 1.500" (13.02cm x 18.10cm x 3.81cm)
	Weight	7.1 oz (200 g)			12.3 oz (348.7 g)
	Data Connection	DB-9	USB	RJ-45	DB-9 / screw terminal
	Operating Temperature	-40 to 85° C (industrial)			

Specifications are subject to change without notice.

The XTend-NEMA enclosure has been tested to the following standards:
IP 66/67 and IP 66; IK 08; NEMA 1, 4, 4X, 6 (12 and 13); UL 94-5V; UL 508

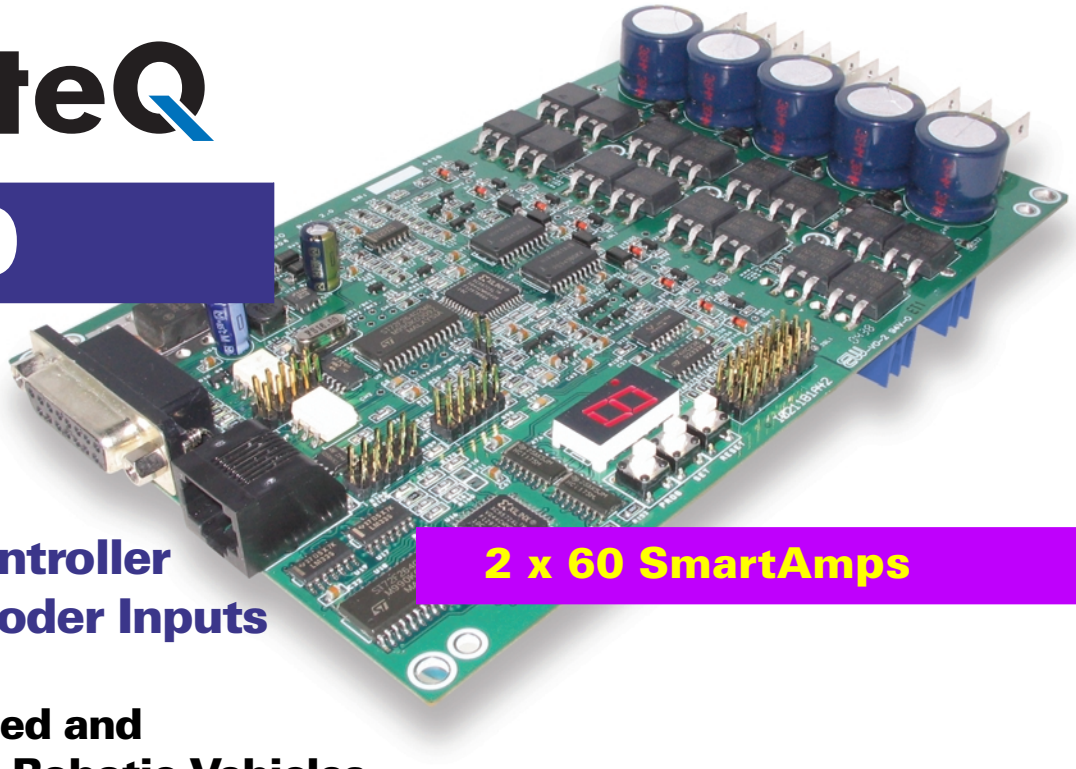
MaxStream®
355 South, 520 West, ste. 180
Lindon, UT 84042

For the best in wireless data solutions and support, contact MaxStream, Inc.

phone: (866) 765-9885 (toll-free in U.S. & Canada)
(801) 765-9885 (worldwide)
fax: (801) 765-9895
web: www.maxstream.net (live chat available)

RoboteQ

AX3500



Dual Channel Forward/Reverse Digital Robot Controller with Optical Encoder Inputs

2 x 60 SmartAmps

for Computer Guided and Remote Controlled Robotic Vehicles

Roboteq's AX3500 controller is a product family designed to convert commands received from a R/C radio, Analog Joystick, wireless modem, or microcomputer into high voltage and high current output for driving one or two DC motors. Designed for maximal ease-of-use in OEM applications, it is delivered with all necessary cables and hardware and is ready to use in minutes.

The controller's two channels can either be operated independently or mixed to set the direction and rotation of a vehicle by coordinating the motion on each side of the vehicle. The motors may be operated in open or closed loop speed mode. Using low-cost position sensors, they may also be set to operate as heavy-duty position servos.

The AX3500 may be ordered in one of many assembly options, depending whether optical encoder input is needed and depending on the application's power, number of channels, and thermal requirements.

The AX2550 can be reprogrammed in the field with the latest features by downloading new operating software from Roboteq's web site. Numerous safety features are incorporated into the controller to ensure reliable and safe operation in the most demanding mobile robotic vehicle applications.

Applications

- Terrestrial and Underwater Robotic Vehicles
- Automatic Guided Vehicles
- Police and Military Robots
- Hazardous Material Handling Robots
- Telepresence Systems
- Animatronics
- Industrial Controls

Key Features	Benefits
Dual MCU digital design	Accurate, reliable, and fully programmable operation. Advanced algorithms
R/C mode support	Connects directly to simple, low cost R/C radios
RS232 Serial mode support	Connects directly to computers for autonomous operation or to wireless modem for two-way remote control
Analog mode support	Connects directly to analog joystick
Optical encoder inputs	Stable speed regardless of load. Accurate measurement of travelled distance
Built-in power drivers for two motors	Supports all common robot drive methods
60A output per channel	Gives robot strongest lifting or pushing power
Programmable current limitation	Protects controller, motors, wiring and battery.
Open loop or closed loop speed control	Low cost or higher accuracy speed control
Closed loop position control	Create low cost, ultra-high torque jumbo servos
Data Logging Output	Capture operating parameters in PC or PDA for analysis
Built-in DC/DC converter	Operates from a single 12V-40V battery
Field upgradable software	Never obsolete. Add features via the internet

Technical Features

Microcomputer-based Digital Design

- Multiple operating modes
- Fully programmable using either built-in switches and 7 segment LED display or through connection to a PC
- Non-volatile storage of user configurable settings. No jumpers needed
- Simple operation
- Software upgradable with new features

Multiple Command Modes

- Serial port (RS-232) input
- Radio-Control Pulse-Width input
- 0-5V Analog Voltage input

Multiple Motor Control modes

- Independent channel operation
- Mixed control (sum and difference) for tank-like steering
- Open Loop or Closed Loop Speed mode
- Position control mode for building high power position servos
- Modes can be set independently for each channel

Optical Encoder Inputs

- Two Quadrature Optical Encoders inputs
- 250kHz max. frequency per channel
- 32-bit up-down counters
- Inputs may be shared with four optional limit switches

Automatic Command Corrections

- Joystick min, max and center calibration
- Selectable deadband width
- Selectable exponentiation factors for each joystick
- 3rd R/C channel input for accessory output activation

Special Function Inputs/Outputs

- 2 Analog inputs. Used as
  - Tachometer inputs for closed loop speed control
  - Potentiometer input for position (servo mode)
  - External temperature sensor inputs
  - User defined purpose (RS232 mode only)
- One Switch input configurable as
  - Emergency stop command
  - Reversing commands when running vehicle inverted
- Up to 2 general purpose outputs for accessories or weapon
  - One 24V, 2A output
  - One low-level digital output
- Up to 2 digital input signals
- 8 RC pulses outputs for connection to additional Roboteq slave controllers or RC servos

Built-in Sensors

- Voltage sensor for monitoring the main 12 to 40V battery
- Voltage monitoring of internal 12V
- Temperature sensors near each Power Transistor bridge

Advanced Data Logging Capabilities

- 12 internal parameters, including battery voltage, captured R/C command, temperature and Amps accessible via RS232 port
- Data may be logged in a PC or microcomputer
- Data Logging Software supplied for PC

Low Power Consumption

- On board DC/DC converter for single 12 to 40V battery system operation
- Optional 12V backup power input for powering safely the controller if the main motor batteries are discharged
- 200mA at 12V or 100mA at 24V idle current consumption
- Power Control wire for turning On or Off the controller from external microcomputer or switch
- No consumption by output stage when motors stopped
- Regulated 5V output for powering R/C radio. Eliminates the need for separate R/C battery.

High Efficiency Motor Power Outputs

- Two independent power output stages
- Dual H bridge for full forward/reverse operation
- Ultra-efficient 2.5 mOhm ON resistance MOSFETs
- Four quadrant operation. Supports regeneration
- 12 to 40 V operation
- User programmable current limit up to 60A depending on heatsink arrangement
- Single channel, 120A optional configuration
- Standard Fast-on connectors for power supply and motors
- 16 kHz Pulse Width Modulation (PWM) output
- Aluminum heat sink. Optional conduction cooling plate

Advanced Safety Features

- Safe power on mode
- Optical isolation on R/C control inputs
- Automatic Power stage off in case of electrically or software induced program failure
- Overvoltage and Undervoltage protection
- Watchdog for automatic motor shutdown in case of command loss (R/C and RS232 modes)
- Large and bright run/failure diagnostics on 7 segment LED display
- Programmable motors acceleration
- Built-in controller overheat sensors
- "Dead-man" switch input
- Emergency Stop input signal and button

Compact Design

- All-in-one, single board design
- Efficient heat sinking. Operates without a fan in most applications.
- 6.75" (171.5mm) L, 4.2" W (107mm), 1.25" (32mm) H
- -20o to +75o C operating environment
- 7.5oz (220g)

Ordering Information

Model	Description
AX3500	Dual Channel 60 SmartAmps DC Motor Controller with Optical encoder input

RoboteQ
www.roboteq.com

8180 E.Del Plomo Dr.
Scottsdale, AZ 85258 - USA
602-617-3931

APPENDIX L – USER MANUAL FOR GRAPHICAL USER INTERFACE

Installation

There are two major software components that are required to be installed: MySQL, Apache/Tomcat Servlet engine, JAVA JDK, etc.

- **Java JDK :** Go to www.java.com to get JDK Download Java 2 Platform, Standard Edition, v 1.4.2 (J2SE v 1.4.2_07 SDK includes the JVM technology)
- After download, run setup and choose all default, let it install

- **Apache Jakarta Project**: go to http://jakarta.apache.org/ and click on:

  - Tomcat
  - Download: Binaries,
  - Downloads: tomcat
  - tomcat5 (http://jakarta.apache.org/site/downloads/downloads_tomcat-5.cgi) 5.0.28 (get the 5.0.28 exe file) and download

- Go to http://java.sun.com/products/javacomm/downloads/index.htm and Download Version 2.0 for Microsoft Windows and Solaris/x86
  - Download: javacomm20-win32.zip
  - Extract files
  - Just need comm.jar file; javax.comm.properties; win32com.dll
  - Copy and paste: comm.jar from javacomm into C:\j2sdk1.4.2_07\jre\lib\ext

  - Copy and paste: javax.comm.properties into C:\j2sdk1.4.2_07\jre\lib

  - Copy and paste: win32com.dll into C:\WINNT\system32

- Install tomcat... default install, port 80, no password, make sure the java VM folder is the right one
- Copy paste the NetrolNew.war file into C:\Program Files\Apache Software Foundation\Tomcat 5.0\webapps
- Restart computer (if needed)
- Turn off firewall on laptop (html computer)
- If Windows IIS is your default service... go to admin tools-->services and stop IIS admin
- Go to http://dev.mysql.com/downloads/mysql/4.1.html to install MYSQL: scroll down and get Windows Essentials (x86) Pick a Mirror and download typical setup... skip creating an account... admin as password
- On laptop make sure MySQL ADMINISTRATOR that the username is root and password is admin.
- Go to http://dev.mysql.com/downloads/administrator/1.0.html and install windows pick a mirror
    - after install on server desktop, go to start, all programs, MYSQL folder, MYSQL ADMINISTRATOR, set ServerHost: localhost, Username: root, Password: admin
    - On next screen go to Restore (on left hand side), open backup file and select the: testbackup 20050221 1011.sql file

Under Properties for MY Computer, Under ADVANCED, Enviromental Variables, System Variables, NEW, Variable name: JAVA_HOME Variable Value: C:\j2sdk1.4.2_07

APPENDIX M – PATENT SEARCH

Appendix M-1 List of Related Patents

U.S. Patent Documents

234794
D270907
D271098
524243
795002
1016808
1167723
1188842
1440644
1904027
1905488
1911427
1911428
1920205
2024822
2229796
2233449
2375286
2889795
2939291
3063397
3160135
3163147
3207110
3224401
3237438
3246476
3279407
3318275
3349740
3391666
3447502
3490406
3537412
3556033
3580205
3623444
3645468
3653354
3658222
3691977
3822559
3830068
3859806
3949693
3982402
3991584
4157023
4174671
4215950
4218906
4243345
4260287
4269540
4286896
4297054
4345855
4436050
4498412
4538937
4570716
4594871
4652177
4657437
4663955
4687376
4716847
4721410
4721411
4723874
4735526
4789108
4802794
4810132
4813869
4820082
4902169
4913591
4917540
4961671
4984934
5038702
5111677
5139366
5188484
5236145
5348423
5377517
5527134
5533834
5555838
5575592
5580187
5655753
5676009
5692859
5836719
5893682

Appendix M-2 *USP 3949693*

United States Patent 3,949,693
Bauer , et al. April 13, 1976

Partially submerged floating platform

Abstract
A floating platform has a polygonal, submerged floating body constructed from concentric pipe sections whereby the space between each two concentric pipes is compartmentized and serves as storage facility as well as ballast tanks. Columns also constructed as upright concentric pipes extend from the submerged float and carry platform defining and establishing frame which carries e.g. a drilling derrick. The interior spaces of all inner pipes serve as transport path, a closed one being provided in the main float and elevator(s) as well as pump-up paths for liquid loads are provided in the columns.

Inventors: Bauer; Peter (Bremen, DT); Klimek; Rudolf (Ganderkesse, DT)
Assignee: ERNO Raumfahrttechnik GmbH (Bremen, DT)
Appl. No.: 517884
Filed: October 25, 1974

Foreign Application Priority Data

May 02, 1974[DT]2421150

Current U.S. Class:114/265; 114/124; 405/207
Intern'l Class: B63B 035/44
Field of Search: 61/46,46.5 114/.5 D,124

Primary Examiner: Shapiro; Jacob
Attorney, Agent or Firm: Siegemund; Ralf H.

Claims

We claim:

1. Partially submerged floating platform having a plurality of vertical columns constructed from concentric pipes with compartmentization of the space between respective inner and outer pipes; a frame interconnecting the columns above the sea level and providing for and establishing a working area of the platform, the improvement comprising: a submerged main floating body of near rotational symmetry and having predominantly horizontal extension, the body being constructed from sections of concentric pipes which extend around a vertical axis in near-annular fashion, the space between respective inner and outer pipe sections being compartmentized, the columns being connected and extending upright from the main body but projecting above sea level; the inner pipe of the body being hollow and constructed as passage way for moving of loads; individual ballast bodies movably disposed in the inner pipe and moving along the passage way to change the effective ballast in relation to the center axis of the platform; the inner pipe of at least one column constructed for transporting loads from the submerged body to the frame.

2. A platform as in claim 1, wherein the main body is constructed as polygon and from straight, concentric pipe sections, joint at obtuse angles.

3. A platform as in claim 1, there being a rail system on a floor in the inner pipes of the body, the ballast load bodies being movable on the rail system.

4. A platform as in claim 3, the ballast loads provided for being driven on the rails.

5. A platform as in claim 1, and including retractible propelling and driving means extending from the body when not retracted and near the zones from which the columns extend upwardly.

Description

BACKGROUND OF THE INVENTION

The present invention relates to a floating platform having a body for displacing water when submerged at a depth which is not or little disturbed by surface waves. More particularly the invention relates to such a platform wherein float columns extend from a submerged body float and are interconnected by a rigid, horizontally extending frame.

Floating platforms with two, parallelly oriented submerged floats are known and these floats carry columns functioning as additional water displacing floats carrying the platform above the sea level. Sometimes the platforms can be vertically displaced on these columns. These platforms with submerged floating body are easily maneuverable and can readily be transported to their destination point, but the two parallelly disposed displacement elements or bodies react to a significant extent to underwater currents and are difficult to maintain in position. This is particularly so if these platforms are used for offshore drilling. The basic reason for this lack in positional stability must be seen in that the submerged bodies exhibit different characteristics as to stability in longitudinal and transverse directions.

Other types of floating platforms are known wherein the submerged floating body is of annular configuration, and the central, open space has an auxiliary platform for resisting vertical movements of the body. This submerged control platform may have openings with means for closing, so that controlled water flow can traverse the platform in either direction. The principle platform above sea level is strutted or otherwise secured to the submerged floating body.

This latter construction has the advantage of isotropic stability and resists vertical displacement to a very significant degree. This annular construction has, however, the disadvantage that the entire pay load including fuel and other provisions to be consumed have to be stored on the main platform so that the center of gravity of this arrangement is rather high and that in turn is disadvantageous as to floating stability.

DESCRIPTION OF THE INVENTION

It is an object of the present invention to provide for a floating platform with submerged support and floating structure and being stabilized with regard to tilting as well as vertical displacement.

In accordance with the preferred embodiment of the present invention it is suggested to provide a main submerged float of quasi-symmetrical configuration, whereby, in particular, concentric pairs of pipe sections are arranged in a polygon, and the spaces between concentric pipe sections are compartmentized and serve generally for floatation control; however, some of the compartments are used for storage of consumables and the other compartments serve as ballast and/or trimming tanks and are flooded or blown to maintain balanced conditions on the platform. A plurality of floatation columns extend upward from that body and project above sea level to support the platform structure constructed as a rigid frame which interconnects these columns. The columns are likewise constructed from concentric pipes with compartmentization of the space between the pipes of each concentric pair, and some of these compartments are filled with usefull load, while others are used also as ballast and/or trimming tanks for buoyancy and stabilizing control.

All of the compartments are, therefor, used as ballast for control of floatation, depths of submerging, tilting if any etc, whereby however, pay load is used as ballast in some compartments and the other compartments may be filled with or emptied from sea water as the conditions require. The usefull load ballast will generally include fuel and other liquids or powdery solids which can be pumped up. The inner tubes provide additional storage space so that only difficult to store objects (drilling rods etc) will be stored in places other than the interior of main float and float columns.

The inner pipes of the main float are preferably interconnected to provide for a continuous passage way in which loads are transported and wherein dummy loads can be rapidly displaced as ballast for stability

control of the entire structure. These ballast loads may run on a track system and may be self-propelled and computer controlled in accordance with any compensating function for purposes of counter acting any tendency for tilting. This way blowing and flooding of tanks for the same purpose can be minimized. The inner pipes of the columns will contain vertical transport means for moving loads up and down. These transport means may include pipes as well as elevators.

The platform in accordance with the invention has the advantage that the principle load (other than surface equipment) is maintained in the principal portions (e.g. 90%) so that the center of gravity is quite low. This is of advantage for avoiding significant vertical displacements. Also, a low center of gravity together with quasi-rotational symmetry of the main float avoids tilting of the platform as much as possible. Since the above-sea level frame interconnecting the columns will carry only that part of the equipment which for some reason or another cannot be stored or placed in either the columns or the submerged main body, the frame will be of relatively light construction, which in turn is again beneficial for a low placement of the center of gravity.

The quasi-rotational symmetry facilitates positioning of the platform because it has no preferred orientation vis a vis any underwater currents. As far as the several compartments is concerned, one will use those on the outside for ballast control, flooding them with sea water or blowing them as needed. The more inwardly arranged compartments will be used as storage space for fuel etc. Some of the outer ones may also serve as storage facility for liquids or solids which, in the case of leakage will not provide for contamination. Storing the contaminants more on the inside permits more ready containment in the case of leakage.

DESCRIPTION OF THE DRAWINGS

While the specification concludes with claims particularly pointing out and distinctly claiming the subject matter which is regarded as the invention, it is believed that the invention, the objects and features of the invention and further objects, features and advantages thereof will be better understood from the following description taken in connection with the accompanying drawings in which:

FIG. 1 is a side elevation of a floating platform constructed in accordance with the preferred embodiment of the invention;

FIG. 2 is a section view through one of the floating carrier columns taken along lines II--II in FIG. 1;

FIG. 3 is a top elevation of the platform shown in FIG. 1; and FIG. 4 and FIG. 5 are section views respectively along lines IV--IV and V--V in FIG. 3.

Proceeding now to the detailed description of the drawings, the Figures show a main submerged body 2 constructed in quasi-rotationally symmetrical configuration, as a multicorner -- multisection polygon approximating an annulus with circular transverse cross-section as can be seen from FIGS. 4 and 5. This body 2 constitutes the principle or main float and is completely submerged.

Float 2 carries four columns 3, also constructed as water displacing floatating elements but extending partially above the sea level L. A rigid frame 4 above the water surface interconnects the four columns. The columns 3 are additionally interconnected by struts or girders or the like. The number of these columns will depend to some extent on the size of this platform, but for reasons of structural stability there should be at least three.

Reference numeral 11 denotes the zones or regions of interconnection as between float 2 and columns 3. The columns 3 have significantly wider cross-section in this zone. Driving and propelling units 10 can be lowered from this zone or retracted so as to maneuver the platform when not retracted.

The columns are vertically traversed e.g. by elevators 13 for moving personnel and load because the lower and submerged portions of the platform are used extensively as storage facilities and otherwise. Chain anchorings 12 are provided for locally positioning the platform and holding it particularly in shallow

waters. These anchors for fastening the chains are secured to float 2 on the outside, but the anchor chains can be stored in the interior thereof.

One of the columns 3 carries the command cabin or control tower 24 with living quarters 25 being provided for in lower stories or decks. The tower 24 may carry a landing platform 26 for helicopters. Various cranes 23 are mounted to the frame 4 and have operating ranges extending at least to the center of the frame as well as to the column 3.

The center of the frame 4 carries the drilling derrick 20 which is equipped as usual. Drilling pipes or rods are stored at 24 right next to tower 20. The frame 4 has a carrier construction 17 for rails 18, and several service modules 19 are moved on these rails for placing them in various operating positions. These modules 19 contain various equipment as needed for drilling, such as equipment for mixing and flushing; equipment for cementing; auxiliary power equipment; equipment for the drilling holes; repair shop and storage of spare parts; additional living quarters for operating personnel if needed.

A storage facility for risen pipes 22 is provided next to the drill pipe store 21. Such storage facility may be needed if not all these pipes can be stored inside of the columns 3.

Each column 3 is constructed as a concentric pipe arrangement having an outer pipe 33 and an inner pipe 34 whereby, the annular space between the concentric pipes is compartmentized by radial walls 35. Additional walls (not shown) compartmentize this space in vertical direction. As stated, an elevator shaft 13 is provided in the center of each column, and pipes 14 for moving powdery, liquidous and/or gaseous media are arranged around that shaft. These pipes are arranged also inside of the inner pipe 34 and move e.g. fuel, exhaust fumes as well as fresh air. Additionally, the inner pipe 34 is used for storage (15), because not everything needed can be stored in the main body 2.

The compartments as between inner and outer pipe can be flooded or emptied as needed for controlling buoyancy and stability of orientation of the platform as a whole. Thus, some of these compartments will serve as ballast and/or trim tanks. However, some of the compartments, namely those more on the inside of the platform construction, such as 16, may be used for storing fuel. This way, all compartments are used as buoyancy and ballast tanks and only some of them are needed for providing supplementary ballast adjustment and control particularly for compensating any change in load conditions on account of consumption of usefull load.

The main submerged floating body 2, particularly the polygonally arranged individual sections thereof, are each comprised of an inner pipe 31 and of an outer pipe 32. Longitudinal walls 30 are provided for compartmentizing the ring space between the two pipes, and transverse walls (not shown) provide for further subdivision of the compartmentization.

The various compartments between inner and outer pipes of body 2 serve also as storage facilities, as well as ballast and/or trim tanks. The compartments more on the outside should be used for storing non-contaminating fluid (e.g. drinking water) or they are flooded with sea water.

The inner pipe 31 constitutes basically a continuous passage way with a transport device 5 arranged on and along the roof of that passage, running all the way around the body 2. This transport structure may be an overhanging rail with carriages running for suspending loads. Rails 29 are provided on the floor of the passage way, particularly for moving carriages 27 which carry ballast weights 28. These ballast weights 28 may be tanks filled with water and are moved on these carriages to the extent needed for trimming the position of the floating platform as a whole. These carriages may be self-propelled vehicles which are computer controlled in accordance with any sensed tilting angles of the floating platform. Any sudden external influence tending to change the stability can readily be compensated by moving these dummy loads into different positions. This can be done quite rapidly if the need arises rather suddenly.

Various power stations 7 and storage containers 6 are provided more or less regularly around the center of the annulus 2, inside of tubes 31 (FIG. 5). Furthermore, the or some of the pipes 31 contain the principal fuel tanks 9 and other ballast tanks 8 (FIG. 4). This way fuel tanks are provided completely in the interior

of the body and leakage towards the outside is quite unlikely, while any leakage from the fuel tanks is readily localized on account of the compartmentization of the space between inner and outer pipes 31, 32.

About 90%, of the pay load, such as provisions and fuel are to be stored in body 2. As a consequence, the center of gravity of the structure as a whole is quite low which is very beneficial for the stability of the platform. The regular and symmetrical distribution of loads generally inside of body 2 enhances stability. On the other hand, the dual transport path inside of pipe 31 permits relocating of loads as well as moving of loads to and from the columns for changing the load distribution as for example, fuel is being used up.

The invention is not limited to the embodiments described above but all changes and modifications thereof not constituting departures from the spirit and scope of the invention are intended to be included.

Appendix M-3 *USP 4174671*

United States Patent 4,174,671
Seidl November 20, 1979

Semisubmerged ship

Abstract
A marine vessel is provided which comprises a platform member, two parallel hulls disposed below the platform and adapted to be below the water surface when the ship is in operation, each of the hulls consists of an after section, a forward section having a smaller cross-sectional area than the after section and a central section having a cross-sectional area substantially smaller than that of the forward section and supported only by the junctions with the forward and after sections. Both forward sections and both after sections are joined to the platform member by strut-like members which are tapered so that their waterplane areas increase rapidly between the waterline and their junctions with the platform member.

Inventors: Seidl; Ludwig H. (Honolulu, HI)
Assignee: Pacific Marine & Supply Co., Ltd. (Honolulu, HI)
Appl. No.: 907122
Filed: May 18, 1978

Current U.S. Class:114/61.14; 114/265
Intern'l Class: B63B 001/10
Field of Search: 114/61,62,56,265,256,283,49,59,312,67 R 244/105
115/20,22,26 D12/62

Primary Examiner: Kunin; Stephen G.
Assistant Examiner: Basinger; Sherman D.
Attorney, Agent or Firm: Limbach, Limbach & Sutton

Claims

What is claimed is:

1. A marine vessel comprising:

(a) a platform member;

(b) two parallel hulls disposed below said platform and adapted to be below the water surface when the vessel is in operation;

(c) each of said hulls comprising a pod shaped after section, a forward section having a generally rectangular cross section of substantially smaller cross-sectional area than the after section and a tubular central section, having a cross-sectional area substantially smaller than that of the forward section, joining the forward end of the after section to the after end of the forward section and supported only by the junctions with the forward and after sections;

(d) each after section being joined to said platform by a strut-like member;

(e) each forward section being joined to said platform by a strut-like member;

(f) all of said strut-like members being tapered so that their waterplane areas increase rapidly between the waterline and the junction of the strut-like member with the platform;

(g) the junctions of the strut-like members with the hull sections being so constructed that a portion of each connected hull section extends forward of the junction.

2. A marine vessel as defined in claim 1 having a third hull of triangular cross section attached to the lower surface of the platform, said triangular hull having a width approximately equal to the transverse distance between the strut-like members, having its bow slightly astern of the bow of the forward section of the first mentioned hulls.

3. A marine vessel as defined in claim 1 wherein an engine is mounted in the after section of each hull.

4. A marine vessel as defined in claim 1 wherein the cross-sectional area of each strut-like member is at a minimum approximately at waterline.

5. The marine vessel defined in claim 2 wherein the depth of the third hull is such that it lies above the water surface during operation of the vessel in calm seas.

Description

BACKGROUND OF THE INVENTION

The catamaran was probably the earliest twinhull vessel to appear. The catamaran consisted of a platform member having two hull members suspended from the platform, one on either side of the platform. The hulls are of uniform cross section and their sides were generally parallel planes.

Subsequently, small waterplane twin hull vessels were developed (sometimes referred to as SWATH ships) by the United States Navy. This type of vessel was a modification of the catamaran in which there were two submerged hulls of uniform cross section which were connected to the platform by elongated struts which have a cross section substantially smaller than the cross section of the submerged hull, hence the small waterplane twin hull characterization.

More recently, the Naval Ocean System Center at San Diego and Honolulu developed a design characterized as a "Two-strut" per side ship. These ships were characterized by submerged twin hulls of uniform cross section and the hulls were suspended from the platform by two narrow struts, one making a connection between the forward end of the submerged hull and the platform and the other making the connection between the after end of the submerged hull and the platform.

The design of the ship of the present invention employs the two-strut per side concept, but by the employment of submerged hulls which have varying cross sectional areas along the hull lengths and by employing strut-like connections between the submerged hulls and the platform which flare outwardly both longitudinally and transversely from the waterline to the junction with the platform, achieves improved stability, improved reduction of wave forces and the more effective use of applied horsepower.

BRIEF DESCRIPTION OF THE INVENTION

The ship of the present invention comprises a platform member, two parallel submerged hulls and two strut-like members connecting each submerged hull to the platform. The submerged hulls consist of an after section, a forward section having a substantially smaller cross sectional area than the cross sectional area of the after section, and a central section of flattened tubular form connecting the forward end of the after section with the after end of the forward section and supported only by the junctions of the central section with the forward and after sections. The forward and after sections of each hull are joined to the platform by strut-like members which flare outwardly both longitudinally and transversely from the waterline to the junction of the strut-like member with the platform. A third hull of triangular cross section is attached to the lower surface of the platform. This hull has a width about equal to the transverse distance to the narrowest transverse distance between the strut-like members which attach the hull to the platform and has a depth such that in normal operation of the ship in relatively calm seas, the third hull is not in contact with the water.

DETAILED DESCRIPTION OF THE INVENTION

In the appended drawings
FIG. 1 is an isometric view of the ship;
FIG. 2 is a side view of the ship;
FIG. 3 is a cross-sectional view of the forward section of the port hull looking aft and taken along line 3--3;
FIG. 4 is a sectional view of the after section of the starboard hull looking aft and taken along line 4--4;
FIG. 5 is a series of horizontal sections of strut-like member 36 showing the cross-sectional areas from the waterline to the junction of the strut-like member with the platform;
FIG. 6 is a series of cross sections of strut-like member 35 showing the cross-sectional areas from the waterline to the junction with the platform;
FIG. 7 is a cross section of FIG. 2 taken along line 7--7.
FIG. 8 is a cross-section of FIG. 2 taken along line 8--8.

Referring now to FIG. 1 platform member 30 is the main deck of the vessel on which any desired superstructure may be erected. Two parallel submerged hulls 31 each consist of an after section 32, a forward section 34, and a central section 33. Strut-like members 35 join the after sections of the hulls to the platform. Strut-like members 36 join the forward sections of the hull to the platform. Narrow hydrofoils 38 are fitted into the after section of after hull member 32 and each supports a rudder 39. Triangular third hull 37 is suspended from the platform between the inner edges of the strut-like members. The bow of this third hull is slightly astern of the bow of the forward sections of the submerged hulls and its depth is preferably such that during operation of the vessel in calm seas the base of the third hull lies above the water surface.

Referring now to FIG. 2 of the appended drawings which is a side view of the ship, platform member 30 constitutes the main deck on which suitable superstructure may be erected. The submerged hull consists of after section 32, central section 33 and forward section 34. The cross-sectional area of after section 32 is substantially greater than that of forward section 34 which in turn is substantially greater than that of central section 33. Strut-like member 36 joins forward section 34 to platform 30. Above the waterline strut-like member 36 is outwardly flared both longitudinally and transversely so that the cross-sectional area of its junction with platform member 30 is substantially greater than that of its junction with forward hull section 34. Strut-like member 35 joins the after section of the submerged hull 32 with the platform and this strut-like member is also flared outwardly from the waterline to its junction with the platform so that the cross-sectional area of its junction with the platform is substantially greater than that of its junction with the after hull section.

Central section 33 of the hull is a flattened tubular form and has a cross-sectional area which is small relative either to the after or forward sections of the hull. Central section 33 of the hull is supported only by its junctions with after section 32 and forward section 34. Above central section 33 and between after section 32 and forward section 34 of the hull is a void space 41 which is filled with water to the waterline when the boat is in operation and through which water may pass freely transversely of the boat. In the upper portion of the void space 41, a portion of the third hull 37 is shown having its bottom above the waterline. Shown in cutaway in the after section of the hull is engine 40. An engine is mounted in each of the after sections of the twin hulls. Small hydrofoil 38 is mounted in the after portion of after hull section 32 and supports rudder 39. If desired hydrofoil 38 may be dispensed with and strut-like member 35 may be extended astern so that it contacts a vertical downward extension of platform 30 and the rudder may be mounted in the strut-like member so extended.

FIG. 3 is a cross-sectional view of forward hull member 34 and strut-like member 36 of the port side hull looking aft. Forward section 34 of the hull is generally rectangular in cross section, but has a bulb form base 43. Strut-like member 36 joins forward hull section 34 to platform 30. One side of the shallow third hull 37 is shown in midships position laterally.

FIG. 4 is a sectional view of the after section of the hull 32 and strut-like member 35 which joins the after section of the hull to platform 30 taken along line 4--4. The after section of the submerged hull 32 is pod-like in form and has a much greater cross-sectional area than does the forward section of the hull, being so sized that engine 40 shown in cutaway may be positioned in the after section of the hull.

FIG. 5 of the drawings shows a series of horizontal sections of strut-like member 36, section a being at the waterline and section f being the section at the junction of strut-like member 36 with platform 30. Sections b, c, d, and e are parallel horizontal sections taken at evenly spaced intervals between the waterline and the platform and show the substantial increase in the waterplane areas of strut-like member 36 between the waterline and the platform.

FIG. 6 shows a series of horizontal cross sections of strut-like member 35 between the waterline and the junction with platform 30. Section g is the section of strut-like member 35 at the waterline and section l is the section of strut-like member 35 at its junction with platform 30. Sections h, i, j, and k are equally spaced sections between the waterline and the platform and show the rapid increase of the waterplane areas of strut-like member 35, between the waterline and the platform.

FIG. 7 is a section of FIG. 2 taken along line 7--7. Triangular third hull 37 lies below platform 30 and has its base above the waterline. Sections of central hull section 33 lie below platform 30, but are not directly connected with it.

FIG. 8 is a section of FIG. 2 taken along line 8--8 and shows platform 30 and triangular third hull 37 lying below the platform and having its base above the waterline. After sections of the submerged hulls 32 are joined to the platform 30 by strut-like members 35.

The design of the submerged hull as shown and described above gives rise to a number of practical operating advantages. The after section of the submerged hull has a substantially greater cross-sectional area than the cross-sectional area of either the forward hull section or the central hull section. In a vessel having an overall length of about 65 feet, the diameter of the pod-shaped after section of the hull is ordinarily greater than 6 feet at its central portion. The design makes it possible for an engine to be disposed in each of the after sections of the two parallel submerged hulls. Disposing the engine in the hull permits the use of a conventional propulsion train and the weight of an engine so disposed lowers the center of gravity of the vessel. Positioning the engine in the after section of the submerged hull saves valuable deck area on the platform and provides maximum possible separation of the engines from passenger accommodations on the platform, thereby reducing engine noise and vibrations to a degree far beyond what might be achieved if the engines were mounted on the platform.

The forward section of the submerged hull is much smaller in cross section and in total volume than the after section. The smaller cross section of the forward section of the submerged hull tends to minimize the disturbance and wake created by it when traveling through the water, and, thus, adverse hydrodynamic interference between the forward section and the after section of the submerged hull is reduced. The weight distribution of the vessel having its engines mounted in the after section of the submerged hull permits the smaller sizing of the forward section since less buoyancy is required in the forward section.

The central section of the submerged hull is not directly joined to the platform and is supported solely by its junctions with the after section and forward section. The design of the central section leaves a void area 41 between the forward section and after section of each hull and this void area reduces overall heave force. The cross-sectional shape of the central section of the hull being of an oblong or horizontally flattened shape as shown in FIG. 7 tends to optimize the heave force reduction effect.

Strut-like members 35 and 36 join the after hull section and the forward hull section, respectively, to the platform. Both have waterplane areas, which increase significantly with vertical distance as shown in FIGS. 5 and 6. The rapid increase in the waterplane areas of the strut-like members above the waterline produces what may be characterized as a "hardening spring" effect in heave, roll and pitch conditions.

Third hull 37, which does not reach the water level under normal operating conditions provides additional longitudinal strength to the vessel and its deep V section reduces wave slap on the underside of the platform during operation in heavy seas. In the event that the lower hull should be partially or completely flooded, the third hull provides reserve buoyancy, which will keep the vessel afloat when the submerged hulls are in damaged condition.

While it is preferred that the third hull be clear of the water during normal operation in calm seas it may be extended downwardly to or below the water surface if desired.

As shown in FIG. 2, the forward transition zone of the after section of the submerged hull to the after strut-like member leads forward and achieves to a degree the effect of a bulbous bow, leading the strut-like member. The forward end of the forward section of the submerged hull is bulbed to produce a similar effect.

Horizontal bilge keels may be mounted on the submerged hulls and stabilizing fins or canards can provide further stabilization against both pitch and roll motions.

The submerged hulls are preferably constructed from mild steel and the platform and third hull from aluminum.

Overall, the design of the present vessel provides improved stability and safety and more efficient use of applied horsepower.

Appendix M-4 *USP 4864958*

United States Patent 4,864,958
Belinsky September 12, 1989

Swap type floating platforms

Abstract
A system for stabilizing floating semi-submersible platforms and compensating ship motion of monohull ships is based on a design protecting waterplane from vertical movement during wave actions. Each vertical strut of the semi-submersible platform utilizing this system consists of a vertical hollow column with its upper end connected to the platform upper structure and lower end open to the surrounding water. Inside the vertical hollow column is inserted a buoyancy vessel which is connected to the upper platform structure by its upper part and which serves as a means of forming a waterplane. During passive mode of this system operation, the roll and pitch of the floating platform (ship), due to wave induced forces, will be introduced because the vertical movement of the water level inside the vertical hollow column will be considerably less than the vertical movement of the water outside the vertical hollow column. During the active mode of SWAP system operation, the roll and pitch of the floating platform (ship), due to wave induced forces, or trim and list, due to the outer moment acting on the ship, will be eliminated or drastically reduced by changing the air pressure in the space between vertical hollow columns, inserted buoyancy vessel and waterline inside the vertical hollow column.

Inventors: Belinsky; Sidney I. (40 Waterside Plaza Apt. 14-A, New York, NY 10010)
Appl. No.: 101285
Filed: September 25, 1987

Current U.S. Class:114/265; 114/125
Intern'l Class: B63B 043/06
Field of Search: 114/265,125,61,264,121

Primary Examiner: Peters, Jr.; Joseph F.
Assistant Examiner: Swinehart; Edwin L.

Claims

I claim:

1. A ship water plane area protected free floating platform comprising: a platform upper deck; at least three struts, each of said struts comprising: a vertical hollow column having a bottom end open to surrounding water and having an upper end connected to said platform upper deck; a buoyancy vessel inserted into said vertical hollow column having a lower end closed and an upper end open to atmosphere, and a ring connecting said upper end of said buoyancy vessel to said upper end of said vertical hollow column; a conduit located below water level connecting said vertical hollow column and said inserted buoyancy vessel of each strut to a corresponding portion of each other said strut; and means for generating vertical forces acting on each of said rings, located between each of said buoyancy vessels and each of said vertical hollow columns to control and trim list of the floating platform.

2. The platform of claim 1 wherein each of said buoyancy vessels is connected to one of said vertical hollow columns in a manner which permits each of said buoyancy vessels to be disconnected and moved in an upward direction.

3. The platform of claim 1 wherein said means for generating forces to control trim and list of said floating platform include: compressed air and vacuum systems which when applied into said space between said vertical hollow column and said inserted buoyancy vessel generate vertical forces on each of said rings

connecting each of said hollow columns to each of said buoyancy vessels; said compressed air and vacuum systems further comprising an air compressor, a compressed air tank, a vacuum tank, pipe lines with remote control valves connecting said space between each of said hollow columns and each of said buoyancy vessels with said compressed air tank, said vacuum tank and atmosphere; means for indicating platform trim and list wherein said remote control valves are responsive thereto; and wherein a preselected space between each of said hollow columns and said buoyancy vessels and above an average water plane level inside each of said hollow columns to accommodate a rise of water level when vacuum is applied above said water plane level and preselected space below said average water plane level to accommodate compressed air without its escape into surrounding water when air pressure is applied above said water plane.

4. A ship water plane area protected, tied, anchored, floating platform comprising: a platform upper deck; and at least three struts, each of said at least three three struts further comprising, a vertical hollow column having a bottom end open to surrounding water and having an upper end connected to said platform upper deck; a downward extension of said vertical hollow column which has a flange on an upper end which is connected to said vertical hollow column; a buoyancy vessel inserted into said vertical hollow column having a lower end closed and an upper end open; and a ring connecting said upper end of said buoyancy vessel to said upper end of said vertical hollow column; a conduit located below water level connecting a space between said vertical hollow columns and the respective buoyancy vessel of each said strut to a similar space in each other said strut; means for generating vertical forces, acting on said ring, located between each of said buoyancy vessels and each of said vertical hollow columns, to control the trim and list of the platform; a foundation on the bottom of the sea including a pontoon with ballasts, an anchoring means and a well template; and catenary mooring lines from the platform to said anchoring means of said foundation.

5. A platform as set forth in claim 4 wherein said means for generating vertical forces to control trim and list of said platform comprises: compressed air and vacuum systems which when applied into said space between each of said vertical hollow columns and each of said inserted buoyancy vessels, generates vertical forces on said rings connecting each of said hollow columns to each of said buoyancy vessels; said compressed air and vacuum system comprising an air compressor, a compressed air tank, a vacuum tank, pipelines with remote control valves connecting said space between each of said hollow columns and each of said buoyancy vessels with said compressed air tank, said vacuum tank and atmosphere; means for indicating platform trim and list said remote control valves being responsive thereto; and a preselected space between each of said hollow columns and each of said buoyancy vessels and above an average water plane level inside each of said hollow columns to accommodate a rise of water level, when vacuum is applied above the water plane, and a preselected space below the average water plane level to accommodate pumpedin compressed air without escape into surrounding water, when air pressure is applied above the water plane.

6. The platform of claim 4 wherein each of said buoyancy vessels is connected to each of said vertical hollow columns in a manner that permits said buoyancy vessels to be disconnected and removed in an upward direction.

7. A ship water plane area protected type catamaran comprising: a propulsion system; at least two submerged hulls, each of said submerged hulls comprising at least two struts interconnecting said upper deck with said submerged hull; each of said struts comprising a vertical hollow column having the form of an elongated box shaped to reduce drag and having its bottom at least partially open to the surrounding water; a buoyancy vessel inserted into said hollow column having a form of an elongated box, a lower end which is closed and an upper end open to the atmosphere and a cover plate connecting said upper end of said buoyancy vessel to an upper end of said hollow column; cross-conduits located below water level and having a form of an elongated box shaped to reduce drag, interconnecting said hollow columns and said inserted buoyancy vessels of oppositely located struts; longitudinal conduits located below water level and passing through said submerged hull interconnecting said hollow columns and said inserted buoyancy vessels of said struts located along one of said submerged hulls; and means for generating vertical forces acting on said cover plate located between said buoyancy vessel and said vertical hollow column to control list and trim of said catamaran.

8. The catamaran of claim 7 wherein each of said buoyancy vessels is connected to said vertical hollow columns so that each of said buoyancy vessels can be disconnected from each of said vertical hollow columns and removed in an upward direction.

9. The catamaran of claim 7 wherein said means for generating vertical forces to control trim and list of the catamaran comprise: compressed air and vacuum systems which when applied into a space between said vertical hollow column and said inserted buoyancy vessel generate vertical forces on said cover plate connecting said hollow column and said buoyancy vessel; wherein said compressed air and vacuum systems include an air compressor, a compressed air tank, a vacuum tank, pipelines with remote control valves, connecting space between each of said hollow columns and each of said buoyancy vessels with said compressed air tank, said vacuum tank and atmosphere; means for indicating catamaran trim and list to which said remote control valves are responsive; and a preselected space between each of said hollow columns and each of said buoyancy vessels and above an average water plane level inside said hollow column to accommodate a rise of water level when vacuum is applied above said water plane and preselected space below said average water plane level to accommodate pumped in compressed air without its escape into surrounding water when air pressure is applied above said water plane.

10. A ship water area protected single body ship comprising: an outer hull, including sidewalls; an inner hull, comprising at least two water-level equalizing conduits crossing said inner hull at said inner hull's lowest part; an upper deck; at least two partitions dividing a space between said outer hull and said inner hull into sections, said sections being connected to each other through openings in lower portions of said partitions; means for generating vertical forces, acting on said upper deck in an area covering said space between said inner hull and said outer hull, thereby controlling trim and list of the single body ship.

11. The ship water area protected single body ship of claim 10 wherein said vertical force generating means comprises: compressed air and vacuum systems, which when applied to said space between said outer hull and said inner hull generate vertical forces on at least a portion of said upper deck connecting said inner hull and said outer hull; wherein said compressed air and vacuum systems include an air compressor, a compressed air tank, vacuum tank and pipelines with remote control valves, said pipelines connecting said space between said inner hull and said outer hull with said compressed air tank, said vacuum tank and atmosphere wherein said remote valves are responsive to a means for indicating trim and list of the ship; and wherein said space between said inner hull and said outer hull is of sufficient size above an average waterplane level to accommodate a rise of water level responsive to said vacuum system and is of sufficient size below the average waterplane level to accommodate a fall in water level responsive to said compressed air system without the escape of air into surrounding waterlevel to accommodate pumped in compressed air without its escape into surrounding water when air pressure is applied above said water plane.

Description

(abbreviation of: Ship Waterplane Area Protected).

BACKGROUND OF THE INVENTION

This invention relates to the Marine Industry and particularly to motion stabilization of floating platforms and ships.

OBJECTIVES OF THE INVENTION

It is an object of the instant invention to improve stability of the floating platforms (ships) during the high seas and to provide them with an ability to automatically keep the platform (ship) on the even keel. Application of this invention is seen for floating drilling platforms, passenger ships, crane ships, floating terminals, SWATH-type ships, etc.

SUMMARY OF THE INVENTION

One of the SWAP type platform innovations is in the structural design that separates floating platform waterplane area from contact with the rapidly alterating waves water level of waves. This feature in a passive way significantly reduces amplitude of platform motion due to the actions of waves. An automated active system designed to keep floating platform always on the even keel (regardless of waves or other external forces) is another innovation of the SWAP type platform.

Separation of the platform waterplane area from wave actions is achieved by locating a vertical hollow column with an open bottom and closed top around the buoyancy vessel, which forms the waterplane area. By interconnecting internal spaces of several oppositely located vertical hollow columns through water level equalizing conduits, the waterplane level inside all the hollow columns will be almost on the same level (or it will alternate equally in each of them) thus significantly reducing trim or list of the platform due to wave actions.

The automation is based on the ability to generate and control vertical forces acting in opposite directions inside the vertical hollow column, introduced to protect waterplane area from waves actions. By connecting space inside vertical hollow column with a vacuum, the downward atmospheric force acting on platform will be generated. By pumping-in compressed air inside this space, upward force acting on the platform will be generated.

By using a pair of symmetrically and oppositely located vertical hollow columns, significant restoring moment can be achieved with low pressure air (up to +0.8 bar) pumped in one and not deep vacuum (up to -0.8 bar) provided to the other.

Combination of the passive system which reduces motion generated by waves only, with active system which activates by actual platform inclination will provide floating platforms with a highly reliable motion compensation system.

Utilization of a vertical hollow column with a downward extension of a significant length will be able to significantly reduce vertical movement (heave) of the floating platform.

BRIEF DESCRIPTION OF THE DRAWINGS
FIG. 1 Plan view of a SWAP type free floating platform
FIG. 2 Elevation, Section A--A from FIG. 1
FIG. 3 Section 8--8 from FIG. 1
FIG. 4 Section C--C from FIG. 2
FIG. 5 Section D--D from FIG. 3
FIG. 6 Platform cross-section
FIG. 7 Platform cross-section through wave slope
FIG. 8 A SWAP type catamaran, Plan E--E, From FIG. 10
FIG. 9 Elevation, Section F--F, from FIG. 8
FIG. 10 Section G--G, from FIG. 8
FIG. 11 A SWAP type anchored platform
FIG. 12 A SWAP type single body ship (plan)
FIG. 13 A SWAP type single body ship (elevation)
FIG. 14 A SWAP type single body ship (section)

Referring to the drawings FIG. 1; FIG. 2; FIG. 3; FIG. 4; FIG. 5 and FIG. 6 the semi-submersible platform 20 includes: upper deck 22, submerged hulls 24, struts 25 containing vertical hollow column 26, and buoyancy vessel 27, horizontal water level equalizing conduits 28 and 30 interconnecting spaces of vertical hollow columns, and system 34 for controlling trim and list of the platform. Upper deck 22 has openings 36, cover 38 and ring 40 for accomodating buoyancy vessel 32. Vertical hollow column 26 has centering guides 42 and its bottom is open to surrounding water.

Buoyancy vessel 32 has connecting flange 48, cylinder 50, bottom 52 and means 54 for controlling water level inside the buoyancy vessel 32. FIG. 6 illustrates System 54 which includes: submerged pump 56,

outflow pipe 58 with valve 60, pump drive 62 and shaft 64. System 34 for controlling trim and list of the platform includes:

air compressor 66, compressed air tank 68, vacuum tank 70, compressed air line 72 with valves 74, vacuum line 76 with valves 78 and atmosphere pipe 80 with valve 82. Air compressor 66 on its exhaust side is interconnected with compressed air tank 68 and atmosphere through valves 84 and 86. On its suction side air compressor 66 is interconnected with vacuum tank 70 and atmosphere through valves 90 and 92. Sensor 94 indicating trim and list of the floating platform 20 is electronically interconnected with all valves of the system and with the compressor drive, thus providing the base for automating of the process of keeping floating platform on the even keel in the event of alterating outer forces acting on the platform.

FIG. 7 illustrates the effect of protecting the waterplane area WA from wave action.

FIG. 8, FIG. 9, and FIG. 10 illustrate a catamaran-type semi-submersible self-propelled ship 21 known as SWATH--Small Waterplane Area Twin Hull--accomodating the instant invention. It includes: Upper deck 22, submerged hulls 24, struts 25 containing vertical hollow columns 26, and buoyancy vessel 27, water level equalizing longitudinal conduit 28 and cross conduits 30. Cross conduit 30 has at its section a hydrofoil form providing additional lift force. A buoyancy vessel 27, inserted into the vertical hollow column 26, has connecting flange 48, body 50 and bottom plate 52. On the bottom plate 52 is mounted system 54 for controlling water level inside the buoyancy vessel 27, which is similar to the system of semisubmersible described earlier. System 34, for controlling trim and list of the ship through regulating pressure (vacuum) inside the vertical columns 26, is similar to the system of semi-submersible described earlier. Means for reducing drag of the vertical columns 26 by utilizing front and back fairings 96. Self-propelled means including of diesel 98, propeller 100 and connecting shaft 102.

FIG. 11 illustrates a SWAP type anchored platform. This platform is of similar design as a SWAP type free floating platform described earlier. The difference is in the additional elements: a downward extension 104 of vertical hollow column 26, tensioned anchor chains 106, a catenary mooring lines 108 and anchors 110, a foundation 112 including pontoon 114, ballast 116, anchoring arrangements 118 and well template 120.

FIG. 12; FIG. 13 and FIG. 14 illustrate a SWAP type single body ship. It consists of inner hull 122 with water level equalizing conduits 124, outer hull 126, part of upper deck 128 covering space above inner and outer hulls, and partitions 130 with opening 132. It incorporates system 34 for controlling trim and list of the ship similar to the same system on the SWAP type free floating platform described earlier.

OPERATION

In an active mode of operation System 34, for controlling trim and list of free and anchored floating platforms, catamaran ship and single body ship, functions in the following manner:

Floating platform deviation from horizontal position (appearance of trim or list or both) triggers the sensor 94, which by remote control opens and closes certain group of valves. For example (FIG. 6) moment M tends to incline the platform (ship) to the right. This leads to signals from indicatior 94 which opens valve 78 on the left column 26L and valve 74 on the right column 26R. As the result of valve 78 opening, vacuum P.sub.v in the space between vertical hollow column 26L and inserted buoyancy vessel 27 will generate atmospheric pressure on the ring 40 and will form force F.sub.v directed downward.

As the result of valve 74 opening, excess pressure P.sub.p in the space between vertical hollow column 26R and inserted buoyancy vessel 32 will generate upward directed force F.sub.p.

Forces F.sub.v and F.sub.p generate countermoment which balances the acting moment and restores the horizontal position of the platform.

APPENDIX N – ELECTRICAL DIAGRAMS

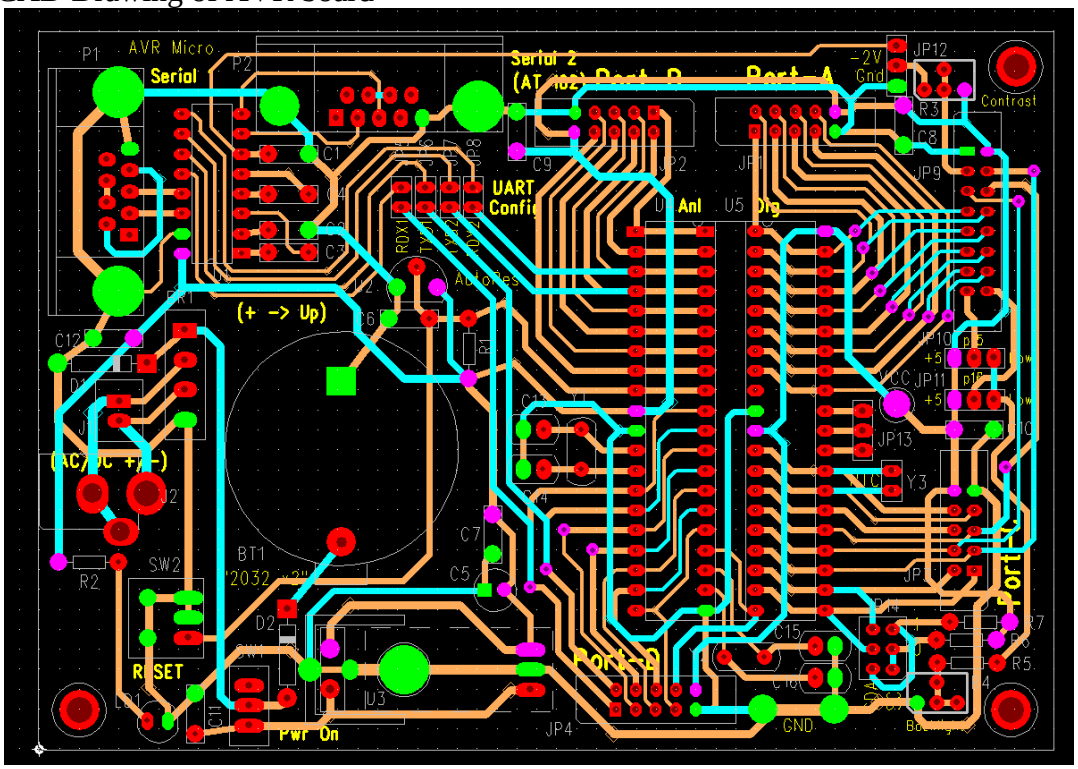
Appendix N-1 Bill of Materials

Bill Of Materials for the AVR board

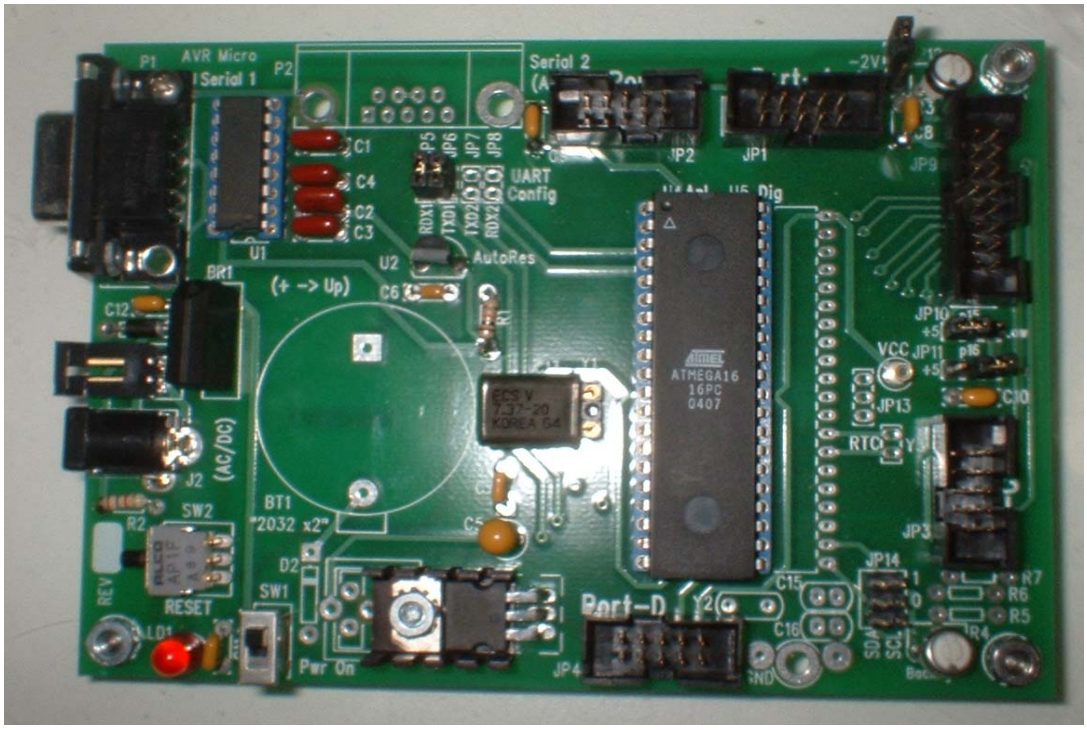
Item	Quantity	Reference	Part
1	1	BR1	DIODE BRIDGE
2	1	BT1	2032
3	11	C1,C2,C3,C4,C6,C7,C8,C9, C10,C11,C12	0.1uF
4	1	C5	10uF
5	4	C13,C14,C15,C16	22pF
6	1	D1	1N4005
7	1	D2	1N5718
8	4	JP1,JP2,JP3,JP4	CONN RECT 10
9	7	JP5,JP6,JP7,JP8,JP10, JP11,JP12	HEADER 3
10	1	JP9	HEADER 8X2
11	1	JP13	CONN RECT 3
12	1	JP14	HEADER 3X2
13	1	J1	HEADER 2
14	1	J2	PHONEJACK
15	1	LD1	LED
16	2	P1,P2	CONN RECT 9
17	3	R1,R6,R7	10K
18	2	R2,R4	1K
19	1	R3	50K
20	1	R5	220
21	1	SW2	SW PUSHBUTTON-SPST/SM
22	1	S1	SW SPDT
23	1	U1	MAX232
24	1	U2	DS1233
25	1	U3	LM2931
26	1	U4	ATMEGA16
27	1	U5	ATMEGA162
28	3	Y1,Y2,Y3	XMhz

Appendix N-2 AVR Boards

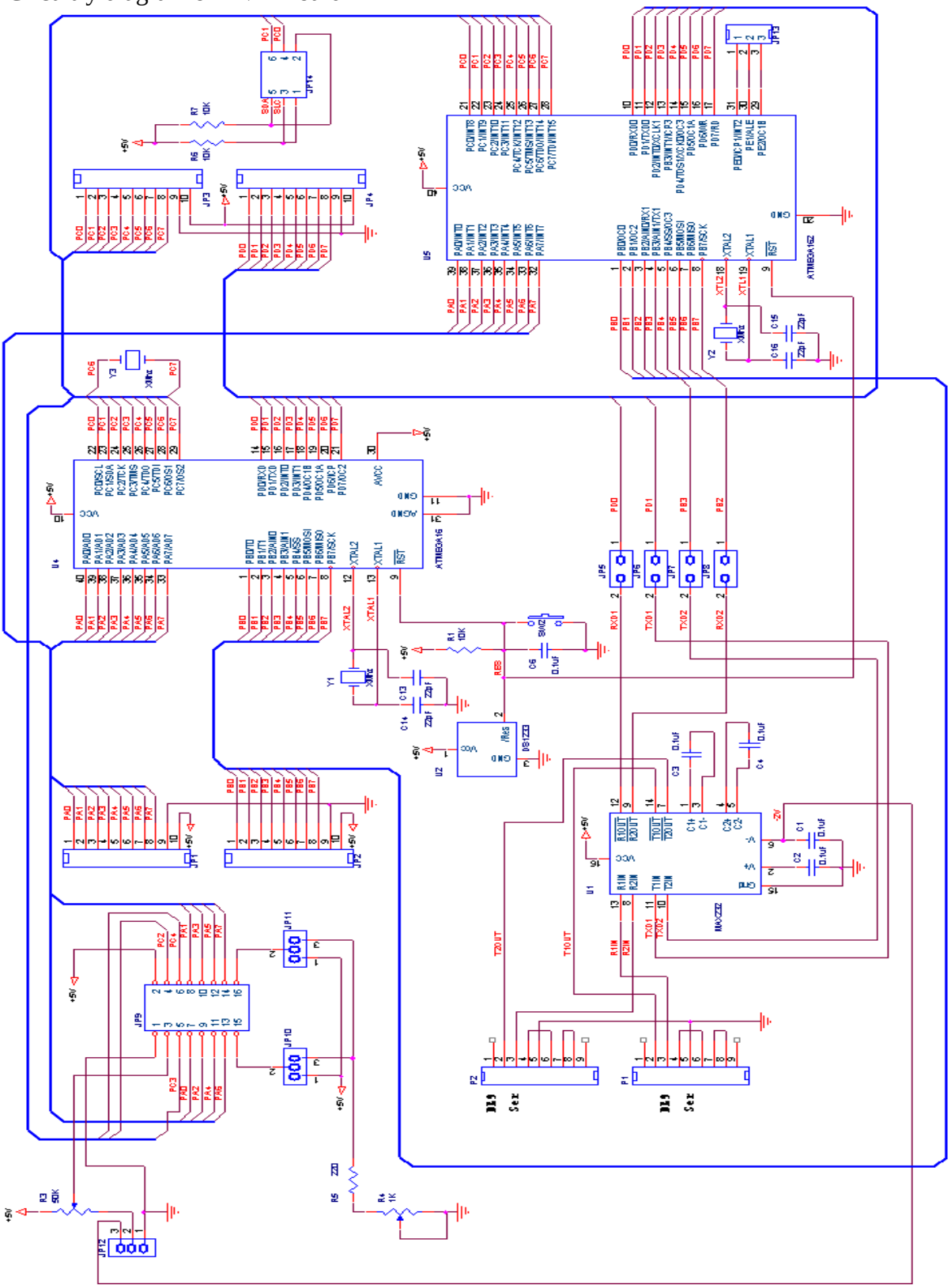
a) CAD Drawing of AVR board



b) Photo of AVR Board

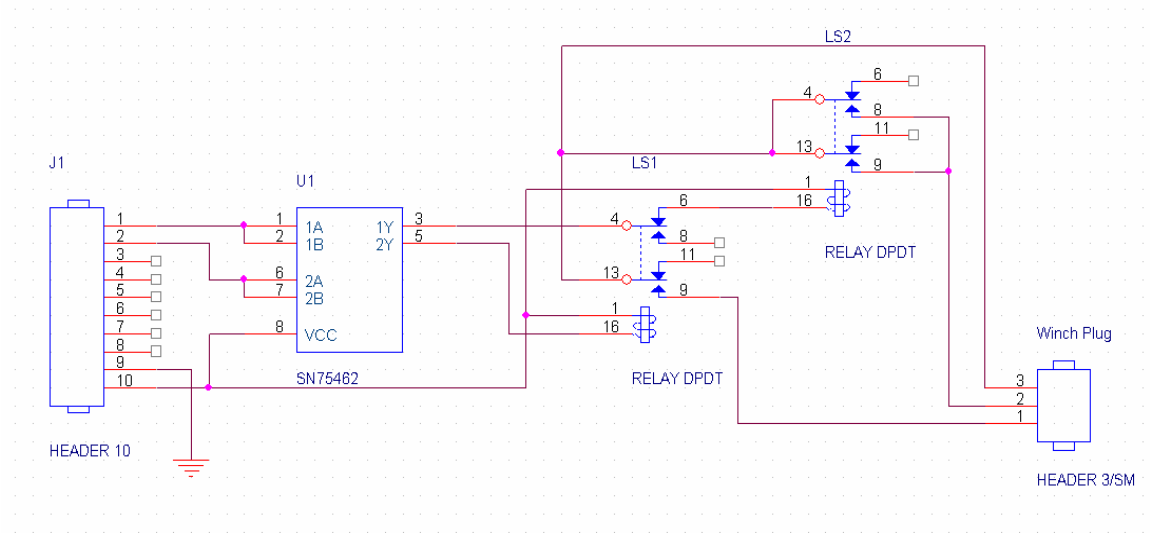


c) Circuitry diagram of AVR Board

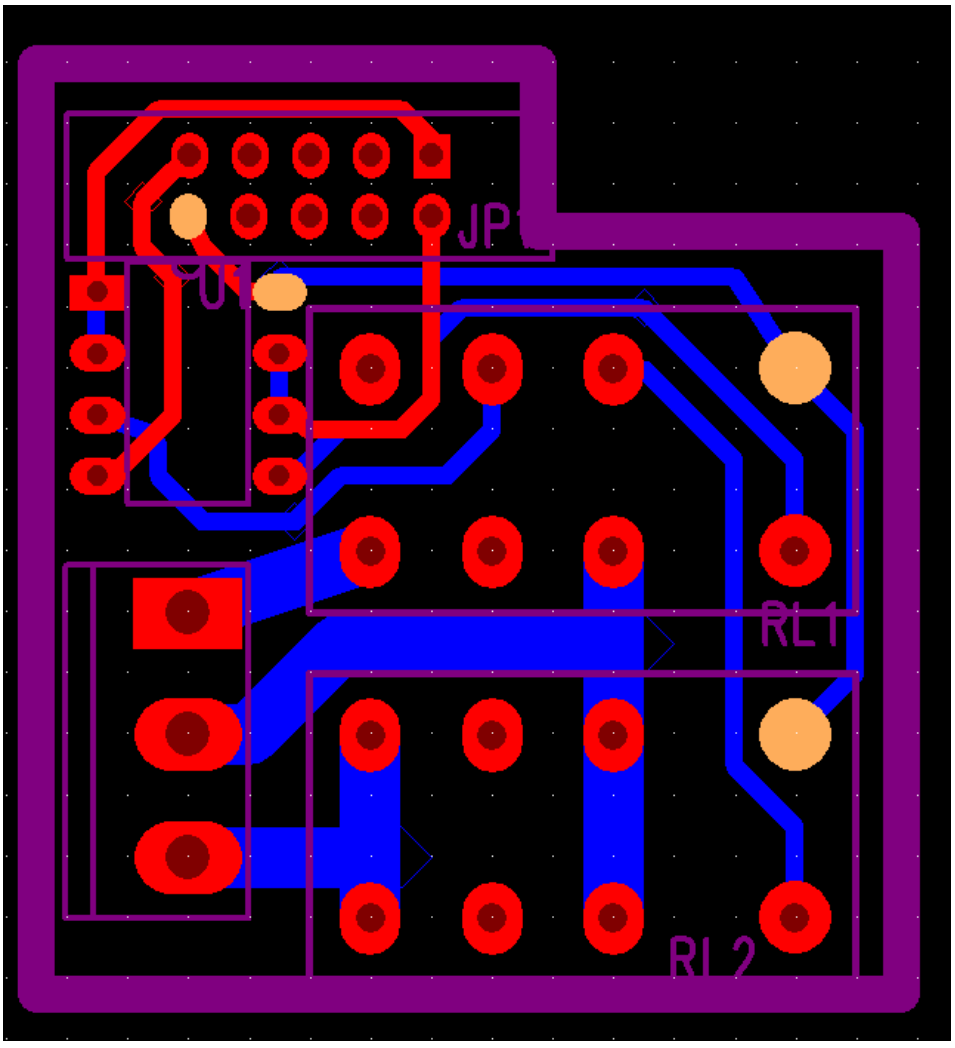


Appendix N – 3 Winch Controller Board

a) Circuitry Diagram of Winch Controller Board

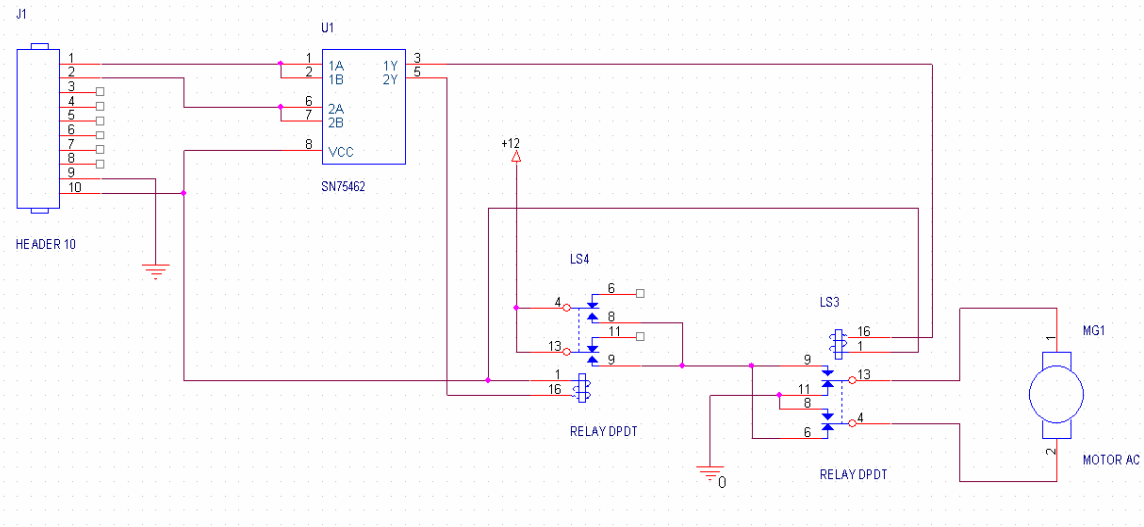


b) CAD drawing of Winch Controller Board

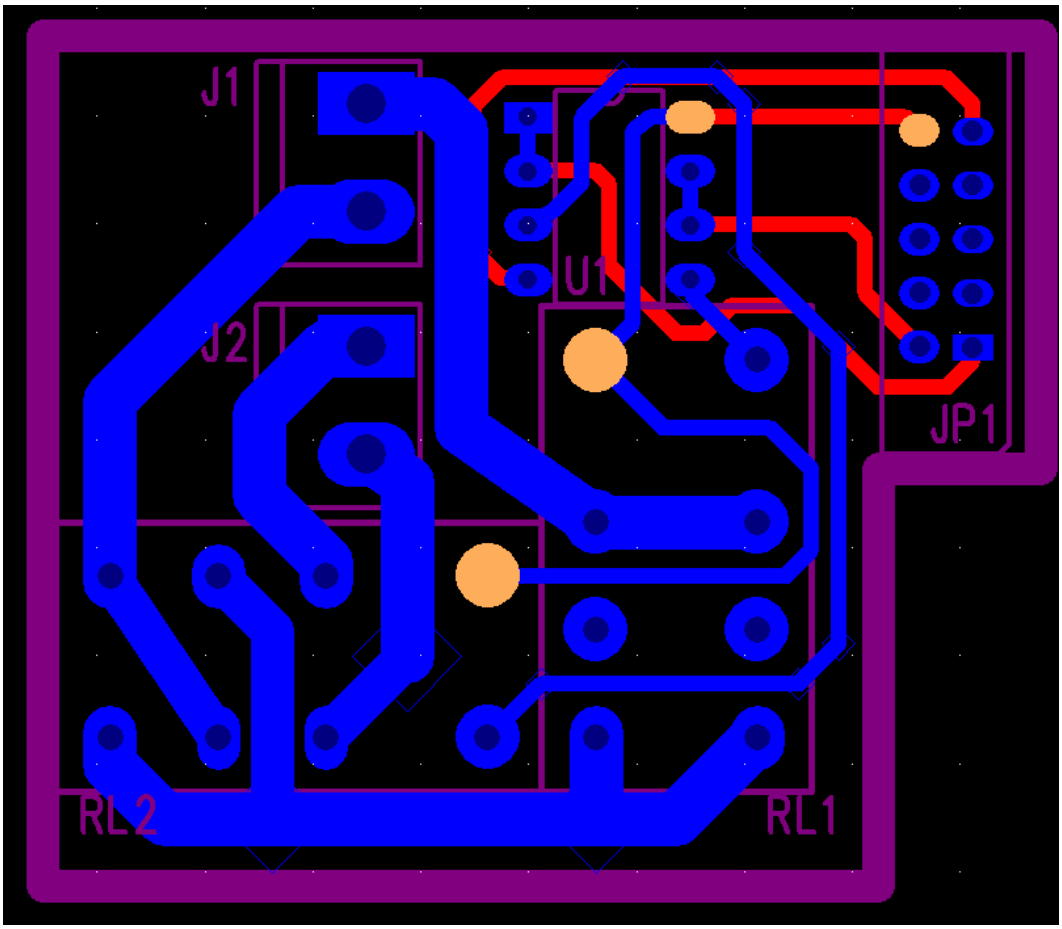


Appendix N – 4 Pan and Tilt Board

a) Circuitry for Pan and Tilt Board



b) CAD drawing for Pan and Tilt Board



Appendix N-5 Command Packet Protocol

ASV Command Packet

Header	Command ID	Separator	Param 1	Separator	Param 2	End byte	Checksum	Delimieter
1 Byte	1 Byte	1 Byte	n Bytes	1 Byte	n Bytes	1 Byte	1 Byte	2 byte

Header: 0x24 [$]
Delimiter: 0x0D0A [\r\n]
End Byte: 0x2A [*]
Separator: 0x2C [,]

Command List

Command Name (Mode)	ID	Paremeter 1	Parameter 2
Autonomous	A	latitude	longitude
Manual	C	X	Y
Science	E	U-up, O-off, D-down	none
Query	?	Char (see list)	none
Set	S	Arg (see list)	Value (see list)
Camera Pan	P	L-left, O-stop, R-right	none

Query Parameters

Parameter	Char
Battery Voltage	E
Power (appl. to motors)	V
Current (appl. to motors)	I
RobotEQ temperature	M
Position	P
Speed	S
Time	T
Course	C
Magnetic Variation	A
Heading	H

Set Parameters

Parameter	Arg	Value
Current Limit	I	0-6 (15A-60A)
Acceleration	A	0-5 (slow-fast)
Proportional Gain	Gp	a positive integer
Integral Gain	Gi	
Derivative Gain	Gd	
Heading	H	1-on, 0-off

APPENDIX O – CODING

Appendix O-1 MyMapPanel.java

```
/*****************************************************************************
 * MyMapPanel.java
 *
 * Created on May 16, 2005, 8:25 AM
 * Author: Nazli Zooleh-Yaghoob-Shahi
 *****************************************************************************/

package screem.netrol.communication.event;


import java.awt.*;
import java.awt.Image;
import java.awt.Color;
import javax.swing.*;
import javax.swing.ImageIcon;
import java.net.*;  // For class URL.
import java.awt.event.MouseListener;
import java.awt.event.MouseEvent;
import java.io.*;
import java.util.*;

public class MyMapPanel extends JPanel implements DataTurbineListener,
MouseListener
{
       final static String newline = "\n";
       JTextArea textArea = new JTextArea();
       public static String url;
       public static ImageIcon ic;
       public static File output = new File("results.txt");
      // public static MyJFrame f = new MyJFrame(700, 400);


      //Constructor for instance creating purposes
      public MyMapPanel() {
      //super (new FlowLayout (FlowLayout.CENTER, 0, 500));

      /*setImageIcon(ic);
      addMouseListener(this); */
    }
     //constructor for creating the image panel

     public MyMapPanel(ImageIcon ic)
     {

         super(new FlowLayout (FlowLayout.CENTER, 0, 50));
         this.ic = ic;
         setImageIcon(ic);
         //addMouseListener(this);
         textArea.setEditable(false);
         JScrollPane scrollPane = new JScrollPane(textArea);
         add(scrollPane);

scrollPane.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
         scrollPane.setPreferredSize(new Dimension(700,100));
     }
     //This method sets the url location to the image
     // and adds the map to MyMapPanel
     public void setImageIcon(ImageIcon ic)
     {
```

```
        this.ic = ic;
        try{
        URL u = new URL(url);
        ic = new ImageIcon(u);
        } catch (MalformedURLException e) {
      System.out.println (e);
    }

        JLabel mapArea = new JLabel(ic, JLabel.CENTER);
        add(mapArea);
        mapArea.addMouseListener(this);
     }
     // sets up the output file to which the realtime GPS data will go
    public void setOutputFile(File output, MouseEvent e)
    {
        this.output = output;
        try {
        FileWriter out = new FileWriter(output);
        double x = e.getX();
        double y = e.getY();
        String s = "Mouse clicked on point" + "("+x+ "," +y+")";
        out.write(s);
        } catch (IOException ex) {
            System.out.println(ex);
        }
    }


    //DataTurbineListener methods
    public void dataTurbineOutputEvent(DataTurbineEvent event){}
    public void dataTurbineInputEvent(DataTurbineEvent event){}
    //Mouselistener methods
    public void mouseReleased(MouseEvent e){}
    public void mousePressed(MouseEvent e) { }
    public void mouseEntered(MouseEvent e) {}
    public void mouseExited(MouseEvent e) { }

    public void mouseClicked(MouseEvent e)
    {
      setOutputFile(output, e);
      saySomething("Position in pixels:(" + e.getX() + ", " + e.getY() + ")",
e);
      String s1 = computeLatLong(e.getX());
      String s2 = computeLatLong(e.getY());
      saySomething("Position in Lat/Long: (" + s1 + ", " + s2 + ")", e);
      Point p = new Point((int)e.getX(), (int)e.getY());
   }
    //This method prints information on the textArea attached to our frame
    //when mouse is clicked on the map image
    void saySomething(String eventDescription, MouseEvent e)
   {
      textArea.append(eventDescription + " detected on " +
e.getComponent().getClass().getName() + "." + newline);
      textArea.setCaretPosition(textArea.getDocument().getLength());
   }
    // calculates latitude and longitude values given the pixel values
    public String computeLatLong(double d)
   {
      double d1;
      double seconds;
      double minutes;
      double m;
      double s;
```

```
      String rval;
      d1 = d - Math.floor(d);
      seconds = d1 / 3600;
      minutes = seconds / 60;
      m = minutes - Math.floor(minutes);
      s = m * 60;                        //minutes converted to seconds
      rval = (int)Math.floor(d) + "deg  " + (int)minutes + "'  " + (int)s +
'"';
      return rval;
   }


    // main method: executes file read and write functions
    //also, creates the JFrame with the textArea and MapArea in it
    public static void main(String[] args) throws IOException
    {

        MyMapPanel p = new MyMapPanel();

        //p.setArchiveFile(output);
        File input = new File("lat_long.txt");
        FileReader in = new FileReader(input);
        /*File output = new File("results.txt");
        FileWriter out = new FileWriter(output); */
        byte[] data = new byte[1024];
        DataTurbineEvent event = new
DataTurbineEvent(input,"lat_long.txt",data);
        byte[] result = event.getSourceData();
        InputStream indata = null;
        OutputStream outdata = null;
        indata = new FileInputStream(input);
        outdata = new FileOutputStream(output);

        int availableLength = indata.available();
        byte[] totalBytes = result;//new byte[availableLength];
        int bytedata = indata.read(totalBytes);

        //writes chunks of bytes into the output file
        outdata.write(totalBytes);

        //pops up the input dialogue window for the user to enter the image URL
       // MyMapPanel mp = new MyMapPanel(ic);
        url = JOptionPane.showInputDialog(null,"Provide image URL:\n","Map
Image", JOptionPane.PLAIN_MESSAGE);
      //  p.setImageIcon(ic);
        ic = new ImageIcon(url);
        //Image img = ic.getImage();
       // p.setImageIcon(ic);

        if(url != null)
        {
            p.setImageIcon(ic);
            MyMapPanel mp = new MyMapPanel(ic);
            //MyMapPanel mp2 = new MyMapPanel();
            //JFrame jf = new JFrame();
            MyJFrame f = new MyJFrame(700, 400);
            f.getContentPane().add(mp);

        }

}
//class JFrame, which creates the outside frame
    //without this class, the GUIwill not exist
```

```
public static class MyJFrame extends JFrame {

  // Constructor.


  public MyJFrame (int width, int height)
  {
    // Set the title and other frame parameters.

    //MyMapPanel mp = new MyMapPanel();
    this.setTitle ("GPS Map");
    this.setResizable (true);
    this.setSize (width, height);


    // Create and add the panel.

    MyMapPanel panel = new MyMapPanel();
    //MyMapPanel p = new MyMapPanel(ic);

    //this.add(panel);

    //this.getContentPane().add (panel);
    this.getContentPane().add(panel);

    // Show the frame.
    this.setVisible (true);
  }


}



}
```

Appendix O-2 LabelImage.java

```
/*
 * LabelImage.java
 *
 * Created on May 30, 2005, 12:04 AM
 */

package screem.netrol.communication.event;

/**
 *
 * @author Nazli
 */
import java.awt.*;
import java.awt.Image;
import javax.swing.*;
import java.awt.event.*;
import java.io.*;
import java.util.*;


/**
 * This class generates a map frame with an image loaded to it.
 */
public class LabelImage extends JFrame implements MouseListener,
DataTurbineListener
{
       ImageIcon icon;
        ImageIcon icon2;
        ImageIcon icon3;
        /**
         * icon used for generating two java input dialoges.
         */
        public static Icon icn;
        /**
         * Label used to create the map image.
         */
        public static JLabel label;
        /**
         * directory path of map image
         */
        public static String path;
        /**
         * Latitude position used for Landmark
         */
        public static double gpsLat;
        /**
         * Longitude position used for Landmark
         */
        public static double gpsLon;
        /**
         * Latitude position used for Landmark
         */
        public static double gpsLat2;
        /**
         * Longitude position used for landmark
         */
        public static double gpsLon2;
        /**
         * Latitude position used for landmark
         */
        public static double gpsLat3;
        /**
```

```
         * Longitude position used for landmark
         */
        public static double gpsLon3;
        /**
         * Width of the loaded map image
         */
        public static int map_width;
        /**
         * Height of the loaded map image
         */
        public static int map_height;
        /**
         * File that contains information about the uploaded map
         */
        public static File mapinfo;
        /**
         * String for toggling points on or off
         */
        public static String toggle;
        /**
         * Image variables used for three different icons
         */
        Image img;
        Image img2;
        Image img3;
        /**
         * String returned from the java input box in MouseClicked
         */
        String pointType;
        /**
         * variables used for drawing lat/long positions that are obtained from
stream.txt
         */
        double latitude;
        double longitude;
        /**
         * when a point is clicked, they are saved to the ArrayList, pointList
         */
        ArrayList pointList = new ArrayList();
       /**
        * Constructor for LabelImage class
        * Specifies three small icons to display different point types.
        * Holds the paint_component function, which is used to plot points
        * on the map.  This function is also recalled in the MouseClicked
        * function.
        */
       public LabelImage()
       {
              /**
                  * Icons that are build for the three images specified earlier
                  */
                 icon = new ImageIcon("C:/dot_small.gif");
                 icon2 = new ImageIcon("C:/icon_dot_small.GIF");
                 icon3 = new ImageIcon("C:/redball.GIF");

                 label = new JLabel( new ImageIcon(path), JLabel.CENTER )
              {
                     public void paintComponent(Graphics g)
                     {
                               super.paintComponent(g);

                               img = icon.getImage();
                               img2 = icon2.getImage();
```

```
                                img3 = icon3.getImage();

                                map_width = label.getWidth();
                                map_height = label.getHeight();


                                //Reading Landmark points from map_info file

                               if(toggle=="Yes")
                               {
                               try{

                               FileReader reader = new FileReader(mapinfo);
                               BufferedReader br = new BufferedReader(reader);
                               String[] lines = new String[7];
                               for(int i=0;i<5;i++)
                               {
                                  lines[i]=br.readLine();
                                  if(lines[i].startsWith("Landmark"))
                                  {
                                      String lat =
lines[i].substring(lines[i].indexOf("(")+1,lines[i].indexOf(","));
                                      String lon =
lines[i].substring(lines[i].indexOf(",")+1, lines[i].indexOf(")"));
                                      Double latvalue = new Double(lat);
                                      Double lonvalue = new Double(lon);
                                      gpsLat = latvalue.doubleValue();
                                      gpsLon = lonvalue.doubleValue();
                                      g.drawImage(img3,(int) gpsLat,(int) gpsLon,
this);
                                 }
                               }
                               } catch (IOException e){}
                               }
                               /**
                                * writing three GPS streams to a file and
plotting them on the map
                                */
                               String stream1 =
"$?,P,A,65.1234,N,123.4321,W*CS\r\n";
                               String stream2 =
"$?,P,A,101.12345,N,200.12345,W*CS\r\n";
                               String stream3 =
"$?,P,V,80.12345,N,156.1654,W*CS\r\n";
                               String[] streams = new String[20];


                                try{
                                    File out = new File("stream.txt");
                                    OutputStream os = new FileOutputStream(out);
                                    PrintStream ps = new PrintStream(os);
                                    ps.println(stream1);
                                    ps.println(stream2);
                                    ps.println(stream3);
                                    FileReader sr = new FileReader(out);
                                    BufferedReader br = new BufferedReader(sr);
                                    for(int i=0;i<5;i++)
                                    {

                                        streams[i]=br.readLine();

                                    }
                                     } catch (IOException e) {}
```

```
                                    int cs = streams[0].indexOf("CS");
                                    String s = streams[0].substring(cs,
cs+2);
                                    if(streams[0].substring(3,4).equals("P"))
                                    {
                                   if(streams[0].substring(5,6).equals("A")
&& s!=null)
                                   {
                                      int a = streams[0].indexOf(',');
                                      int b = streams[0].indexOf('N');
                                      int c = streams[0].lastIndexOf(',');
                                      String lat =
streams[0].substring(a+5,b-1);  //Latitude
                                      String lon =
streams[0].substring(b+2,c);
                                      Double x = new Double(lat);
                                      Double y = new Double(lon);
                                      gpsLat = x.doubleValue();
                                      gpsLon = y.doubleValue();
                                      g.drawImage(img, (int)gpsLat,
(int)gpsLon, this);
                                    }
                                   }
                                    int cs2 = streams[2].indexOf("CS");
                                    String s2 = streams[2].substring(cs2,
cs2+2);
                                    if(streams[2].substring(3,4).equals("P"))
                                    {
                                   if(streams[2].substring(5,6).equals("A")
&& s!=null)
                                   {
                                      int a = streams[2].indexOf(',');
                                      int b = streams[2].indexOf('N');
                                      int c = streams[2].lastIndexOf(',');
                                      String lat =
streams[2].substring(a+5,b-1);  //Latitude
                                      String lon =
streams[2].substring(b+2,c);    //Longitude
                                      Double x = new Double(lat);
                                      Double y = new Double(lon);
                                      gpsLat2 = x.doubleValue();
                                      gpsLon2 = y.doubleValue();
                                      g.drawImage(img, (int)gpsLat2,
(int)gpsLon2, this);

                                    }
                                   }
                                   int cs3 = streams[4].indexOf("CS");
                                    String s3 = streams[4].substring(cs3,
cs3+2);
                                    if(streams[4].substring(3,4).equals("P"))
                                    {
                                   if(streams[4].substring(5,6).equals("A")
&& s!=null)
                                   {
                                      int a = streams[4].indexOf(',');
                                      int b = streams[4].indexOf('N');
                                      int c = streams[4].lastIndexOf(',');
                                      String lat =
streams[4].substring(a+5,b-1);  //Latitude
```

```
                                        String lon =
streams[4].substring(b+2,c);    //Longitude
                                        Double x = new Double(lat);
                                        Double y = new Double(lon);
                                        gpsLat3 = x.doubleValue();
                                        gpsLon3 = y.doubleValue();
                                        g.drawImage(img, (int)gpsLat3,
(int)gpsLon3, this);

                                     }
                                    }

                                if(pointType == "BoatPoint" ||
pointType=="WayPoint")
                                {
                                    for(int i=0;i<pointList.size();i++)
                                    {
                                        Point p = (Point)pointList.get(i);
                                        g.drawImage(img, p.x, p.y, this);
                                    }

                                }


                            }

                       };
                  getContentPane().add( label );

                     label.addMouseListener(this);

            }



             /**
              * MouseListener implemented function
              * @param e specifies a mouse-driven event from the MouseEvent library.
              */
             public void mouseReleased(MouseEvent e){ }
             /**
              * MouseListener implemented function
              * @param e specifies a mouse-driven event from the MouseEvent library.
              */
             public void mousePressed(MouseEvent e) { }
             /**
              * MouseListener implemented function
              * @param e specifies a mouse-driven event from the MouseEvent library.
              */
             public void mouseEntered(MouseEvent e) { }
             /**
              * MouseListener impelemented function.
              * @param e specifies a mouse-driven event from the MouseEvent library.
              */
             public void mouseExited(MouseEvent e) { }

             /**
              * MouseListener Implemented Function.
              * This function is called each time the mouse is clicked on the map.
              * It allows the user to specify point type, and calls repaint() to
              * plot the clicked point on the map.  The repaint function is a call
              * to the paint_component(Graphics g) from outside the function.
              * @param e specifies a mouse-driven event from the MouseEvent library.
```

```
         */
        public void mouseClicked(MouseEvent e)
        {
            Object[] possibilities = {"BoatPoint", "WayPoint"};
            JFrame frame = new JFrame();

            pointType = (String)JOptionPane.showInputDialog(
                    frame,
                    "Select type of Point:\n"
                    +"from the options below:",
                    "Point Type Selection",
                    JOptionPane.PLAIN_MESSAGE,
                    icn,
                    possibilities,
                    "BoatPoints");
                Point p = e.getPoint();
                pointList.add(p);
                System.out.println("current waypoint pixels:"+p.x+", "+p.y);
                repaint();
                computeLatLong(e.getX(),e.getY());
                System.out.println("Point Type:"+pointType);
        }
        /**
         * This function reads values for latitude and longitude scales and
origins
         * from the mission.txt file and uses them to calculate relative values
for
         * latitude and longitude of a clicked point and display the pointin
the
         * MouseClicked function.
         * @param x specifies the latitude position in pixels to be converted
to
         * miles.
         * @param y specifies the longitude position in pixels to be converted
to miles.
         */
        public void computeLatLong(double x, double y)
        {
          String s;
          double lat_origin;
          double lon_origin;
          String lat_scale;
          String lon_scale;

          try{

          File mission = new File("mission.txt");
          FileReader in = new FileReader(mission);
          BufferedReader br = new BufferedReader(in);

          //UPLOADING LAT_SCALE AND LON_SCALE FROM MISSION FILE

             String line1 = br.readLine();
             String line2 = br.readLine();
             lat_scale =
line1.substring(line1.indexOf("=")+1,line1.indexOf("("));
             lon_scale =
line2.substring(line2.indexOf("=")+1,line2.indexOf("("));
             Double lat = new Double(lat_scale);
             double lat_scale_value = lat.doubleValue();
             Double lon = new Double(lon_scale);
             double lon_scale_value = lon.doubleValue();
             String line3 = br.readLine();
```

```
              String line4 = br.readLine();
              String origin1 = line3.substring(line3.indexOf("=")+1);
              String origin2 = line4.substring(line4.indexOf("=")+1);
              Double orgn1 = new Double(origin1);
              Double orgn2 = new Double(origin2);
              lat_origin = orgn1.doubleValue();
              lon_origin = orgn2.doubleValue();
              if(x==0 && y==0)
              {
                latitude = lat_origin;
                longitude = lon_origin;
                System.out.println("Latitude:"+latitude+"
Longitude:"+longitude);
                System.out.println("LatOrigin:"+lat_origin+"
LonOrigin:"+lon_origin);
              }
              else
              {
                 latitude = lat_origin + x * (lat_scale_value);
                 longitude = lon_origin + x* (lon_scale_value);
                 System.out.println("Latitude:"+latitude+"
Longitude:"+longitude);
                 System.out.println("LatOrigin:"+lat_origin+"
LonOrigin:"+lon_origin);
              }
           } catch (IOException e){ }

        }

        /**
         * DataTurbineEvent implemented class.
         * This function is invoked when input is about to be received from an
RBNB
         * channel.
         * @param event event associated with receiving data from the
DataTurbine.
         */
        public void dataTurbineInputEvent(DataTurbineEvent event)
        {
            byte[] dataBytes = event.getSourceData();
            String data = new String(dataBytes);
            StringTokenizer st = new StringTokenizer(data, ",", false);
            while(st.hasMoreTokens())
            {
                String s = st.nextToken();
                System.out.println(s);
            }
        }
        /**
         * DataTurbineEvent implemented class.
         * This function is invoked when output is about to be sent from a
DataTurbine.
         * @param event event associated with sending data from the DataTurbine
         */
        public void dataTurbineOutputEvent(DataTurbineEvent event){ }

        /**
         * Main function definition.
         * Displays map frame
         * Generates map_info.txt file
         * Toggles landmarks on or off based on user selection.
         */
        public static void main(String[] args) throws IOException
```

```
       {

                path = JOptionPane.showInputDialog(null,"Provide path for map
image:\n","Map Image", JOptionPane.PLAIN_MESSAGE);
                System.out.println("Start of Mission Path:");
                String filename = path.substring(path.lastIndexOf("/")+1,
path.indexOf("."));
                System.out.println(filename);
                mapinfo = new File(filename+"_info.txt");
                ImageIcon mapImage = new ImageIcon(path);

                int map_width = mapImage.getIconWidth();
                int map_height= mapImage.getIconHeight();
                /**
                 * writing map information to a file.
                 * The file's name starts with the name of the image
                 */

                try{

                      FileOutputStream fos = new FileOutputStream(mapinfo);
                      PrintStream ps = new PrintStream(fos);
                      ps.println("Map Information for "+filename);
                      ps.println("Map Size:"+map_width+" * "+map_height+"
(Pixel * Pixel)");
                      ps.println("Landmarkpoint1:(54.1234,65.2345)");
                      ps.println("Landmarkpoint2:(10.1234,43.2343)");
                      ps.println("Landmarkpoint3:(67.5434,89.4323)");
                      ps.close();
                     } catch (IOException e) {}

            Object[] possibilities = {"Yes", "No"};
            JFrame frame2 = new JFrame();
            toggle = (String)JOptionPane.showInputDialog(
                    frame2,
                    "Please choose\n"
                    +"from the options below:",
                    "Point Toggle Selection",
                    JOptionPane.PLAIN_MESSAGE,
                    icn,
                    possibilities,
                    "Yes");
                /**
                 * generating frame for the map
                 */
                LabelImage frame = new LabelImage();
                frame.setDefaultCloseOperation( EXIT_ON_CLOSE );
                frame.pack();
                frame.setResizable(false);
                frame.setLocationRelativeTo( null );
                frame.setTitle("Map Interface");
                frame.setVisible(true);


      }

}
```

Appendix O-3 DataTubineListener.java

```
/*
 * DataTurbineListener.java
 *
 * Created on May 16, 2005, 8:29 AM
 */

package screem.netrol.communication.event;

/**
 *
 * @author Nazli
 */
public interface DataTurbineListener {

    public void dataTurbineInputEvent(DataTurbineEvent event);
    public void dataTurbineOutputEvent(DataTurbineEvent event);

}
```

Appendix O-4 DataTurbineEvent.java

```
/*
 * DataTurbineEvent.java
 *
 * Created on May 16, 2005, 8:31 AM
 */

package screem.netrol.communication.event;

/**
 *
 * @author Nazli
 */

import java.util.EventObject;
public class DataTurbineEvent extends EventObject
{
	/**
	* Specifies the name of the channel that fires the event.
	**/
	private String	channelName;

	/**
	* Byte array holding the data received when the event is fired.
	**/
	private byte[]	data;

	/**
	* Constructs a DataTurbineEvent object.
	* @param source the DataTurbine connection (Robot/Operator object) that
originated the event
	* @param channelName the name of the channel that received or sent data
	* @param data the data set or received on the source channel
	**/
	public DataTurbineEvent(Object source, String channelName, byte[] data) {
		super(source);
		this.channelName = channelName;
		this.data = data;
	}

	/**
	* Gets the channel name which fires the event.
	* @return a String that follows the naming scheme of channels:
<i>&ltfunction&gt:&ltname&gt</i>
	**/
	public String getSourceChannel() {
		return channelName;
	}

	/**
	* Gets the data received on the channel
	* @return Byte array received on the data channel
	**/
	public byte[] getSourceData() {
		return data;
	}
}
```

Appendix O-5 MySQLReader.java

```
/*********************************************************************
 *  Main Class  MySQLReader.java
 *
 * Created on May 12, 2005, 8:14 PM
 * @ Author: Nazli Zooleh-Yaghoob-Shahi
 ********************************************************************/
package SDProject;


import java.sql.*;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;


class MySQLReader {

// The connect string : the means to connect to a specified hostname, port
number and database name


public static final String testIP = "jdbc:mysql://localhost:3306/myboat",
loginName = "root", pass = "Tehran2004";
                  public static void main(String[] args)
             {
                 try
                {

                                 Connection conn = null;


             // See if we need to open the connection to the database
             if (conn == null) {
             // Load the JDBC driver
                   Class.forName("com.mysql.jdbc.Driver");


// Connect to the databse

       System.out.println("Connecting to " + testIP + "\n");

conn = DriverManager.getConnection (testIP, loginName, pass);
System.out.println ("Connected\n");
}

// Create a statement
Statement stmt = conn.createStatement ();

// Create a SQL statement
 String qs = "select * from myboatdata;
ResultSet rs = stmt.executeQuery(qs);
System.out.println(rs);

// Step through the result set
boolean more = rs.next();
while (more){

  System.out.print("Latitude: "+rs.getString("latitude"));
  System.out.println("Longitude: "+rs.getString("longitude"));
  more = rs.next();
}
// We're done : Connection is established
```

```
System.out.println  ("done.\n");

}
catch (Exception e)
{
       // Oops
       System.out.println ("Error: " + e.getMessage ());
}
}
}
```

Appendix O-6 Test Input for MyMapPanel.java

```
/**********************************************************************
 * Test Input File for MyMapPanel.java
 * Created on May 12, 2005, 8:14 PM
 * @ Author: Nazli Zooleh-Yaghoob-Shahi
 **********************************************************************/
Latitude         Longitude
34°24'60' N      118°30'23' W
37°21'00' N      121°58'00' W
45°20'50' N      125°40'25' W
50°30'00' N      130°60'30' W
75°19'12' N      135°65'35' W
55°15'10' N      140°70'40' W
60°33'80' N      145°75'50  W
90°15'90' N      150°80'80' W
```

Appendix O-7 GPS.c

```
//*****************************************************************************
// File Name  : gps.c
//
// Title            : gps parsing for SCU ASV
// Revision         : 1.0
// Notes            :
// Target MCU : Atmel AVR series
//
// Revision History:
// When                    Who                 Description of change
// -----------      -----------   ----------------------
// 10-Sep-2002      aschooley           Created the program
//*****************************************************************************


//----- Include Files ---------------------------------------------------------
#include <avr/io.h> // include I/O definitions (port names, pin names, etc)
#include <avr/signal.h>    // include "signal" names (interrupt names)
#include <avr/interrupt.h> // include interrupt support
#include <stdlib.h>

#include "global.h"        // include our global settings
#include "uart_asv.h"// include function library _asv adds a bigger buff
#include "rprintf.h"       // include printf function library
#include "timer.h"         // include timer function library (timing, PWM, etc)
#include "gps_asv.h"              // include gps data support
#include "nmea_asv.h"             // include NMEA gps packet handling
#include "lcd.h"           // include LCD function library
#include "i2c.h"           // include i2c function library

//----- Defines ---------------------------------------------------------------
#define LOCAL_ADDR  0x20


//----- Global Variables ------------------------------------------------------
extern GpsInfoType GpsInfo;

u08 localBuffer[80];
u08 localBufferLength = 80;
u08 slaveServiceStatus = 0;  //written to 1 if slave service is in use
u08 slaveRecieveCounter = 0;
u08 timeOfFixStr[6];
u08 timeOfFixStrLength = 6;
u08 latStr[11];
u08 latStrLength = 11;
u08 lonStr[11];
u08 lonStrLength = 11;
u08 request;

float  autoLat;
float  autoLon;
u08    manualX;
u08    manualY;

//----- Prototypes ------------------------------------------------------------

// uartRxOverflow is a global variable defined in uart.c/uart2.c
// we define it here as <extern> here so that we can use its value
// in code contained in this file
//extern unsigned short uartRxOverflow[2];

void gpsNmea(void);
```

```
u16  compass(void);
u08 i2cSlaveTransmitService(u08 transmitDataLengthMax, u08* transmitData);
void i2cSlaveReceiveService(u08 receiveDataLength, u08* receiveData);

//----- Begin Code ------------------------------------------------------------
int main(void)
{
	// initialize our libraries
	// initialize the UART (serial port)
	uartInit();
	// set the baud rate of UART 0 for our debug/reporting output
	uartSetBaudRate(9600);
	// initialize the timer system
	timerInit();
	// initialize LCD
//	lcdInit();
	// direct printf output to LCD
//	rprintfInit(lcdDataWrite);
	// initialize the I2c bus
	i2cInit();
	// set local device address and allow response to general call
	i2cSetLocalDeviceAddr(LOCAL_ADDR, TRUE);
	// set the Slave Receive Handler function
	// (this function will run whenever a master somewhere else on the bus
	//  writes data to us as a slave)
	i2cSetSlaveReceiveHandler( i2cSlaveReceiveService );
	// set the Slave Transmit Handler function
	// (this function will run whenever a master somewhere else on the bus
	//  attempts to read data from us as a slave)
	i2cSetSlaveTransmitHandler( i2cSlaveTransmitService );
	// set i2c bitrate to 400KHz
	i2cSetBitrate(400);
	// initialize gps library
	gpsInit();
	// initialize gps packet decoder
	nmeaInit();

	// set port b to output for test lights
//	outb(DDRB, 0xFF);
	// set port b pins high
//	outb(PORTB,0xFF);

	// Print hello world message
//	rprintf("GPS System");
//	timerPause(1000);
//	lcdClear();

	// start program loop
	while(1)
	{
		// run gps processing loop
		gpsNmea();
		timerPause(1);

		// run compass
		//compass();

		//TEST CODE
/*		lcdGotoXY(10,2);
		rprintfStr(timeOfFixStr);
		rprintf("*");

		lcdGotoXY(0,3);
```

```
			rprintfStr(localBuffer);
			rprintf("*%d",slaveRecieveCounter);
			timerPause(500);
*/

	}
	return 0;

}



void gpsNmea(void)
{
	// begin gps packet processing loop

	// process received gps packets until receive buffer is exhausted
	while( nmeaProcess(uartGetRxBuffer()) )
	{

	// print/dump current formatted GPS data
//	lcdClear();
//	lcdGotoXY(0,0);
//	rprintfFloat(8, (GpsInfo.PosLLA.lat.f));
//	rprintfFloat(6, GpsInfo.PosLLA.TimeOfFix.f);
//	lcdGotoXY(0,1);
//	rprintfFloat(9, (GpsInfo.PosLLA.lon.f));
//	rprintf("G:");
//	rprintfNum(10,3,0,' ',GpsInfo.PosLLA.updates);


	// do not update status if it is being read by another master
	// on the i2c bus
	if(slaveServiceStatus==0)
		{
			// turn off our slave service acknowledgemet
			cbi(TWCR, TWEA);
			// update time of fix string
			dtostrf(GpsInfo.PosLLA.TimeOfFix.f, 6, 0,timeOfFixStr);

			dtostrf(GpsInfo.PosLLA.lat.f, 11, 5,latStr);

			dtostrf(GpsInfo.PosLLA.lon.f, 11, 5,lonStr);
			// turn on our slave service acknowledgemet
			sbi(TWCR, TWEA);
		}

	}
}
/*
u16 compass(void)
{
	// some variables
	u08 addr;
	u08 data[2];
	u16 heading;

	// set the i2c bitrate to conservative speed (50KHz)
	i2cSetBitrate(50);

	// send the register address you want to access
	// you're sending contents of "addr", one byte
	addr = 2;
```

```
	i2cMasterSend(0xC0,1,&addr);

	// then read from the device to get the data
	// here we read one byte into "data"
	i2cMasterReceive(0xC0,2,data);

	// convert the two u08 to one u16 value
	heading = (256*data[0])+data[1];

	// print current heading to lcd
	lcdGotoXY(0,3);
	rprintf("Compass = %d    " ,heading);

	// set the i2c bitrate to conservative speed (50KHz)
	i2cSetBitrate(400);

	return(heading);
}
*/

u08 i2cSlaveTransmitService(u08 transmitDataLengthMax, u08* transmitData)
{
	// this function will run when a master somewhere else on the bus
	// addresses us and wishes to read data from us

	//indicate that this service is running
	slaveServiceStatus = 1;

	// counter variable
	u08 i;
	u08 lengthTransmitted =0;

	// ADD AN ONLINE/OFFLINE INDICATOR HERE!!

	switch(request)
	{
		case 'T':
			// copy the local buffer to the transmit buffer
			for(i=0; i<timeOfFixStrLength; i++)
			{
				*transmitData++ = timeOfFixStr[i];
				lengthTransmitted = timeOfFixStrLength;
			}
			break;
		case 'L':
			// copy the local buffer to the transmit buffer
			for(i=0; i<latStrLength; i++)
			{
				*transmitData++ = latStr[i];
				lengthTransmitted = latStrLength;
			}
			break;
		case 'l':
				  // copy the local buffer to the transmit buffer
			for(i=0; i<lonStrLength; i++)
			{
				*transmitData++ = lonStr[i];
				lengthTransmitted = lonStrLength;
			}
			break;
	}
```

```
	//incidate that this service is done
	slaveServiceStatus = 0;

	return lengthTransmitted;
}

void i2cSlaveReceiveService(u08 receiveDataLength, u08* receiveData)
{
	u08 i;

	slaveRecieveCounter++;

	// this function will run when a master somewhere else on the bus
	// addresses us and wishes to write data to us



	// copy the received data to a local buffer
	for(i=0; i<receiveDataLength; i++)
	{
		localBuffer[i] = *receiveData++;
	}
	localBufferLength = receiveDataLength;

	localBuffer[i] = 0;

	request = localBuffer[0];

}
```

Appendix O-8 mtr_cntl.c

```
//*****************************************************************************
// File Name  : mtr_cntl.c
//
// Title            : dynamics controller for SCU ASV
// Outputs          : rs232: drive signals for roboteq
//                          i2c:
// Revision         : 1.0
// Notes            :
// Target MCU : Atmel AVR series
//
// Revision History:
// When                    Who                 Description of change
// -----------      -----------   ----------------------
// 10-Sep-2002      aschooley           Created the program
//*****************************************************************************


//----- Include Files ---------------------------------------------------------
#include <avr/io.h>        // include I/O definitions (port names, pin names,
etc)
#include <avr/signal.h>    // include "signal" names (interrupt names)
#include <avr/interrupt.h> // include interrupt support
#include <stdlib.h>

#include "global.h"        // include our global settings
#include "uart_asv.h"             // include function library
#include "rprintf.h"       // include printf function library
#include "timer.h"         // include timer function library (timing, PWM, etc)
#include "lcd.h"           // include LCD function library
#include "i2c.h"           // include i2c function library
#include "packet.h"
#include "RobotEQ.h"
#include "buffer.h"

//----- Defines ---------------------------------------------------------------
#define LOCAL_ADDR  0x40
#define COMM        0x10

//----- Global Variables ------------------------------------------------------

u08 localBuffer[80] = "C,+00,+00*FF";
u08 localBufferLength = 80;
u08 transmitBuffer[8] = "";
u08 transmitBufferLength = 8;
u08 slaveServiceStatus = 0;  //written to 1 if slave service is in use
u08 slaveRecieveCounter = 0;
u08 timeOfFixStr[6];
u08 timeOfFixStrLength = 6;

u08 battVoltage = 0;
u08 pwrApplied = 0;
u08 currentConsumed = 0;
u08 temperature = 0;

extern u08 manualY[4];
extern u08 manualX[4];
extern char xDir;
extern char yDir;
```

```
//----- Prototypes ------------------------------------------------------------

// uartRxOverflow is a global variable defined in uart.c/uart2.c
// we define it here as <extern> here so that we can use its value
// in code contained in this file
//extern unsigned short uartRxOverflow[2];

u08 i2cSlaveTransmitService(u08 transmitDataLengthMax, u08* transmitData);
void i2cSlaveReceiveService(u08 receiveDataLength, u08* receiveData);

//----- Begin Code ------------------------------------------------------------
int main(void)
{
	// initialize our libraries
	// initialize the UART (serial port)
	uartInit();
	/* Set frame format: 7data, 1stop bit, parity even*/
	UCSRC = (1<<URSEL)|(1<<UPM1)|(1<<UCSZ1);
	// set the baud rate of UART 0 for our debug/reporting output
	uartSetBaudRate(9600);
	// initialize the timer system
	timerInit();
	// initialize LCD
//	lcdInit();
	// direct printf output to serial
	rprintfInit(uartSendByte);
	// initialize the I2c bus
	i2cInit();
	// set local device address and allow response to general call
	i2cSetLocalDeviceAddr(LOCAL_ADDR, TRUE);
	// set the Slave Receive Handler function
	// (this function will run whenever a master somewhere else on the bus
	//  writes data to us as a slave)
	i2cSetSlaveReceiveHandler( i2cSlaveReceiveService );
	// set the Slave Transmit Handler function
	// (this function will run whenever a master somewhere else on the bus
	//  attempts to read data from us as a slave)
	i2cSetSlaveTransmitHandler( i2cSlaveTransmitService );
	// set i2c bitrate to 400KHz
	i2cSetBitrate(400);

	rprintf("motor controller\r\n");

	xDir = 0;
	yDir = 0;


	// initialize LCD
//	lcdInit();




	readModify();acceleration();slow();end();
	timerPause(10);
	apply();
	timerPause(10);

	readModify();inputControlMode();RS232Watchdog();end();
	timerPause(10);
	apply();
	timerPause(10);
```

```
	// start program loop
	while(1)
	{

		u08 i = 0;

		switch(localBuffer[0])
			{
				case 'C':
					packetProcessManual(localBuffer);
					if(manualY[0]=='0')
						manualY[1]='0';
					if(manualX[0]=='0')
						manualX[1]='0';


					if(yDir==1)
					{
					set();forward();rprintfStr(manualY);end();
					}
					else
					{
					set();reverse();rprintfStr(manualY);end();
					}

					if(xDir==1)
					{
					set();right();rprintfStr(manualX);end();
					}
					else
					{
					set();left();rprintfStr(manualX);end();
					}

					localBuffer[0]=0;

					break;
				case 'E':

					uartFlushReceiveBuffer();
					timerPause(10);
					query();voltage();end();
					//uartFlushReceiveBuffer();
					timerPause(10);

				//	rprintfInit(lcdDataWrite);
				//	lcdClear();

					while(uartGetByte() != '\r');
					while(uartReceiveBufferIsEmpty() == FALSE &&
i<16)
					{
						transmitBuffer[i]=uartGetByte();
				//		rprintf("%c",transmitBuffer[i]);
						i++;
					}
				//	rprintf("  i: %d",i);
					transmitBuffer[i]=0;

					//lcdGotoXY(0,1);
					//rprintfStr(transmitBuffer);
				//	lcdGotoXY(0,0);
				//	rprintfInit(uartSendByte);
```

```
                        localBuffer[0]=0;
                        break;
                case 'V':

                        uartFlushReceiveBuffer();
                        timerPause(10);
                        query();power();end();
                        //uartFlushReceiveBuffer();
                        timerPause(10);

                //      rprintfInit(lcdDataWrite);
                //      lcdClear();

                        while(uartGetByte() != '\r');
                        while(uartReceiveBufferIsEmpty() == FALSE &&
i<16)
                        {
                                transmitBuffer[i]=uartGetByte();
                //              rprintf("%c",transmitBuffer[i]);
                                i++;
                        }
                //      rprintf("  i: %d",i);
                        transmitBuffer[i]=0;

                        //lcdGotoXY(0,1);
                        //rprintfStr(transmitBuffer);
                //      lcdGotoXY(0,0);
                //      rprintfInit(uartSendByte);
                        localBuffer[0]=0;


                        break;
                case 'I':
                        uartFlushReceiveBuffer();
                        timerPause(10);
                        query();current();end();
                        //uartFlushReceiveBuffer();
                        timerPause(10);

                //      rprintfInit(lcdDataWrite);
                //      lcdClear();

                        while(uartGetByte() != '\r');
                        while(uartReceiveBufferIsEmpty() == FALSE &&
i<16)
                        {
                                transmitBuffer[i]=uartGetByte();
                //              rprintf("%c",transmitBuffer[i]);
                                i++;
                        }
                //      rprintf("  i: %d",i);
                        transmitBuffer[i]=0;

                        //lcdGotoXY(0,1);
                        //rprintfStr(transmitBuffer);
                //      lcdGotoXY(0,0);
                //      rprintfInit(uartSendByte);
                        localBuffer[0]=0;
                        break;
                case 'M':
                        uartFlushReceiveBuffer();
                        timerPause(10);
                        query();temp();end();
```

```
                                    //uartFlushReceiveBuffer();
                                    timerPause(10);

                              //    rprintfInit(lcdDataWrite);
                              //    lcdClear();

                                    while(uartGetByte() != '\r');
                                    while(uartReceiveBufferIsEmpty() == FALSE &&
i<16)
                                    {
                                          transmitBuffer[i]=uartGetByte();
                              //          rprintf("%c",transmitBuffer[i]);
                                          i++;
                                    }
                              //    rprintf("  i: %d",i);
                                    transmitBuffer[i]=0;

                                    //lcdGotoXY(0,1);
                                    //rprintfStr(transmitBuffer);
                              //    lcdGotoXY(0,0);
                              //    rprintfInit(uartSendByte);
                                    localBuffer[0]=0;
                                    break;
                              default:
                                    rprintf("|");
                                    break;
                        }



      }
      return 0;

}



u08 i2cSlaveTransmitService(u08 transmitDataLengthMax, u08* transmitData)
{
      // this function will run when a master somewhere else on the bus
      // addresses us and wishes to read data from us

      //indicate that this service is running
      slaveServiceStatus = 1;

      // counter variable
      u08 i;

      // copy the local buffer to the transmit buffer
      for(i=0; transmitBuffer[i] != 0; i++)
      {
            *transmitData++ = transmitBuffer[i];
      }

      transmitBuffer[0]=0;

      //incidate that this service is done
      slaveServiceStatus = 0;

      return timeOfFixStrLength;
}

void i2cSlaveReceiveService(u08 receiveDataLength, u08* receiveData)
```

```
{
	u08 i;

//	slaveRecieveCounter++;

	// this function will run when a master somewhere else on the bus
	// addresses us and wishes to write data to us



	// copy the received data to a local buffer
	for(i=0; i<receiveDataLength; i++)
	{
		localBuffer[i] = *receiveData++;
	}
	localBufferLength = receiveDataLength;

	localBuffer[i] = 0;



}
```

Appendix O-9 ‘RobotEQ.h’

```
//*****************************************************************************
//
// File Name  : 'RobotEQ.h'
// Title            : RobotEQ defines
// Author           : Aaron Schooley
// Created          : 2005.03.14
// Revised          :
// Version          : 0.1
// Target MCU : Atmel AVR series
// Editor Tabs      : 4
//
// This code is distributed under the GNU Public License
//           which can be found at http://www.gnu.org/licenses/gpl.txt
//
//*****************************************************************************

void set(void)
{
      rprintf("!");
}
void query(void)
{
      rprintf("?");
}
void readModify(void)
{
      rprintf("^");
}
void end(void)
{
      rprintf("\r");
}
void apply(void)
{
      rprintf("^FF\r");
}
void reset(void)
{
      rprintf("%rrrrrr\r");
}


void forward(void)
{
      rprintf("A");
}
void reverse(void)
{
      rprintf("a");
}
void right(void)
{
      rprintf("B");
}
void left(void)
{
      rprintf("b");
}
void accessoryOn(void)
{
      rprintf("C");
}
```

```
void accessoryOff(void)
{
	rprintf("c");
}




void power(void)
{
	rprintf("V");
}
void current(void)
{
	rprintf("A");
}
void temp(void)
{
	rprintf("M");
}
void voltage(void)
{
	rprintf("E");
}



void inputControlMode(void)
{
	rprintf("00");
}
	void RS232(void)
	{
		rprintf(" 01");
	}
	void RS232Watchdog(void)
	{
		rprintf(" 02");
	}

void motorControlMode(void)
{
	rprintf("01");
}
	void separateAB(void)
	{
		rprintf(" 00");
	}
	void mixedAB(void)
	{
		rprintf(" 01");
	}

void currentLimit(void)
{
	rprintf("02");
}
	void _15A(void)
	{
		rprintf(" 00");
	}
	void _22_5A(void)
```

```
	{
		rprintf(" 01");
	}
	void _30A(void)
	{
		rprintf(" 02");
	}
	void _37_5A(void)
	{
		rprintf(" 03");
	}
	void _45A(void)
	{
		rprintf(" 04");
	}
	void _52_5A(void)
	{
		rprintf(" 05");
	}
	void _60A(void)
	{
		rprintf(" 06");
	}


void acceleration(void)
{
	rprintf("03");
}
	void verySlow(void)
	{
		rprintf(" 00");
	}
	void slow(void)
	{
		rprintf(" 01");
	}
	void mediumSlow(void)
	{
		rprintf(" 02");
	}
	void medium(void)
	{
		rprintf(" 03");
	}
	void fast(void)
	{
		rprintf(" 04");
	}
	void fastest(void)
	{
		rprintf(" 05");
	}

/*
	void _00(void)
{
	rprintfInit(uartSendByte);
	rprintf("00");
	rprintfInit(lcdDataWrite);
}
void _10(void)
```

```
{
	rprintfInit(uartSendByte);
	rprintf("0C");
	rprintfInit(lcdDataWrite);
}
void _20(void)
{
	rprintfInit(uartSendByte);
	rprintf("19");
	rprintfInit(lcdDataWrite);
}
void _30(void)
{
	rprintfInit(uartSendByte);
	rprintf("26");
	rprintfInit(lcdDataWrite);
}
void _40(void)
{
	rprintfInit(uartSendByte);
	rprintf("32");
	rprintfInit(lcdDataWrite);
}
void _50(void)
{
	rprintfInit(uartSendByte);
	rprintf("3F");
	rprintfInit(lcdDataWrite);
}
void _60(void)
{
	rprintfInit(uartSendByte);
	rprintf("4C");
	rprintfInit(lcdDataWrite);
}
void _70(void)
{
	rprintfInit(uartSendByte);
	rprintf("58");
	rprintfInit(lcdDataWrite);
}
void _80(void)
{
	rprintfInit(uartSendByte);
	rprintf("65");
	rprintfInit(lcdDataWrite);
}
void _90(void)
{
	rprintfInit(uartSendByte);
	rprintf("72");
	rprintfInit(lcdDataWrite);
}
void _100(void)
{
	rprintfInit(uartSendByte);
	rprintf("7F");
	rprintfInit(lcdDataWrite);
}
*/
```

Appendix O-10 cmps.c

```
//*************************************************************************
// File Name   : cmps.c
//
// Title              : -compass interface protocol with basic smoothing algorithm
//                    : -currently the soothing algorithm is there but not being used
//                    : -designed to be used with the devantech cmps03
// Revision           : 1.0
// Notes              :
// Target MCU  : Atmel AVR series
//
// Revision History:
// When               Who                  Description of change
// ----------- -----------    ----------------------
// 10-Nov-2004 aschooley             Created the program
//*************************************************************************


//----- Include Files ---------------------------------------------------
#include <avr/io.h>           // include I/O definitions (port names, pin names, etc)
#include <avr/signal.h>       // include "signal" names (interrupt names)
#include <avr/interrupt.h>    // include interrupt support

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

#include "global.h"           // include our global settings
#include "uart_asv.h"  // include function library _asv adds a bigger buff
#include "rprintf.h"   // include printf function library
#include "timer.h"            // include timer function library (timing, PWM, etc)
#include "i2c.h"              // include i2c function library

//----- Defines -------------------------------------------------------------
#define LOCAL_ADDR     0x30   // local i2c address

//----- Global Variables ----------------------------------------------------
u08 localBuffer[80];                 // used as a slave recieve buffer
u08 localBufferLength = 80;
char updateFlag = FALSE;             // flag to signal data being updated
u08 headingStr[8];                          // string for our current heading
u08 headingStrBackup[8];             // backup of heading string
u08 length = 0;                                    // length of heading string

//----- Prototypes ----------------------------------------------------------
u16  compass(void);
u08 i2cSlaveTransmitService(u08 transmitDataLengthMax, u08* transmitData);
void i2cSlaveReceiveService(u08 receiveDataLength, u08* receiveData);

//----- Begin Code ------------------------------------------------------------
int main(void)
{
        // initialize our libraries
        // initialize the UART (serial port)
        uartInit();
        // set the baud rate of UART 0 for our debug/reporting output
        uartSetBaudRate(9600);
        // initialize the timer system
        timerInit();
        // direct printf output to UART
        rprintfInit(uartSendByte);
        // initialize the I2c bus
        i2cInit();
        // set local device address and allow response to general call
        i2cSetLocalDeviceAddr(LOCAL_ADDR, TRUE);
        // set the Slave Receive Handler function
        // (this function will run whenever a master somewhere else on the bus
        //  writes data to us as a slave)
        i2cSetSlaveReceiveHandler( i2cSlaveReceiveService );
        // set the Slave Transmit Handler function
```

```
	// (this function will run whenever a master somewhere else on the bus
	//  attempts to read data from us as a slave)
	i2cSetSlaveTransmitHandler( i2cSlaveTransmitService );
	// set i2c bitrate to 400KHz
	i2cSetBitrate(400);

	// Print hello world message
	rprintf("Compass System\r");
	timerPause(1000);
	// Print a header for output data
	//  - this header and the data following it is not really necessary
	//    the only reason i included it was for debugging purposes since
	//    the UART isn't being used for anything else, also if you copy the
	//    output into excel it has been formatted so that you can plot the
	//    data easily and actually see the response time and amount of
	//    smoothing the algorithm really does
	rprintf("raw smooth\r");

	// initalize filter variables
	u16 current = compass();
	u16 smoothed = current * 16;
	u16 smoothOutput = 0;
	int delta = 0;

	// start program loop
	while(1)
	{

		// demo smoothing algorythm.  Its not great but something.  Currently
		// is its not actually used for anything.  To transmit the smoothed
		// value replace 'current' with 'smoothed' in the sprintf statement.
		// This algorithm is largely based on one created by emesystems.com
		// for a basic stamp but the idea is same.

		// get the current raw comapss value
		current = compass();
		// get the difference between the current raw and smoothed values
		delta = (current * 16) - smoothed;
		// divide delta by 4 (essentially your gain)
		delta = (delta + 3)/4;
		// adjust the smoothed value so that it begins to converge toward
		// the actual heading at 1/4 the difference each iteration
		smoothed = (smoothed + delta);
		// divide the output by 16 format it correctly
		smoothOutput = smoothed/16;


		// write updateFlag to TRUE, this will divert our slave service to
		// read from the backup buffer while we are updating our main buffer
		// this is done to avoid partially written corrupt data, instead
		// we trade off for slightly old data
		updateFlag = TRUE;
		// update time of fix string, sprintf works similar to rprintf instead
		// it's output will go into headingStr and returns length.
		length = sprintf(headingStr,"%d",current);
		// turn our update flag off
		updateFlag = FALSE;
		// copy over our backup string with our new one
		strcpy(headingStrBackup , headingStr);


		// insert a pause to prevent loading down the i2c bus
		// if it takes 16 iterations to converge it'll take approx .8 of a sec.
		timerPause(50);

		// print the raw and smoothed values, formatted for excel
		rprintf("%d\t%d\r",current,smoothOutput);
	}
	return 0;
}
```

```
//gets compass info from the cmps03 and returns it
u16 compass(void)
{
	// some variables
	u08 addr;
	u08 data[2];
	u16 heading;

	// send the register address you want to access
	// you're sending contents of "addr", one byte
	addr = 2;
	i2cMasterSend(0xC0,1,&addr);

	// then read from the device to get the data
	// here we read two bytes into "data"
	i2cMasterReceive(0xC0,2,data);

	// convert the two u08 to one u16 value
	heading = (256*data[0])+data[1];

	// return our current heading
	return(heading);
}


// transmits headingStr upon request
u08 i2cSlaveTransmitService(u08 transmitDataLengthMax, u08* transmitData)
{
	// this function will run when a master somewhere else on the bus
	// addresses us and wishes to read data from us

	// counter variable
	u08 i;

	// this condidion statement determines which string will be returned;
	// headingStr or headingStrBackup.  The newest value will be returned
	// unless it is being written to, if that is the case a backup copy of
	// this value is kept and that will be transmittted instead
	if(updateFlag == FALSE)
	{
		// copy headingStr to the transmit buffer if it is not being written to
		for(i=0; headingStr[i] != 0; i++)
		{
			*transmitData++ = headingStr[i];
		}
	}
	else
	{
		// copy headingStrBackup to the transmit buffer if headingStr is
		// currently being updated when this interrupt runs
		for(i=0; headingStrBackup[i] != 0; i++)
		{
			*transmitData++ = headingStrBackup[i];
		}
	}

	// return the number of bytes written
	return i;
}

void i2cSlaveReceiveService(u08 receiveDataLength, u08* receiveData)
{
	u08 i;

	// this function will run when a master somewhere else on the bus
	// addresses us and wishes to write data to us

	// copy the received data to a local buffer
	for(i=0; i<receiveDataLength; i++)
	{
		localBuffer[i] = *receiveData++;
```

```
        }
        localBufferLength = receiveDataLength;

        localBuffer[i] = 0;
}
```

Appendix O-11 ‘gps_asv.c’

```
//****************************************************************************
//
// File Name  : 'gps_asv.c'
// Title            : GPS position storage and processing function library
// Author           : Pascal Stang - Copyright (C) 2002-2005
// Version          : 0.1
// Target MCU : Atmel AVR Series
// Editor Tabs      : 4
//
// NOTE: This code is currently below version 1.0, and therefore is considered
// to be lacking in some functionality or documentation, or may not be fully
// tested.  Nonetheless, you can expect most functions to work.
//
// This code is distributed under the GNU Public License
//           which can be found at http://www.gnu.org/licenses/gpl.txt
//
// revised by Aaron Schooley
//****************************************************************************

#ifndef WIN32
      #include <avr/io.h>
      #include <avr/signal.h>
      #include <avr/interrupt.h>
      #include <avr/pgmspace.h>
      #include <math.h>
      #include <stdlib.h>
#endif

#include "global.h"
#include "rprintf.h"
#include "gps_asv.h"
#include "lcd.h"

// Global variables
GpsInfoType GpsInfo;
float desiredLat = 3721.1700;
float desiredLon = -12156.410;

// Functions
void gpsInit(void)
{

}

GpsInfoType* gpsGetInfo(void)
{
      return &GpsInfo;
}

void gpsInfoPrint(void)
{


/*    rprintfProgStrM("TOW:      ");   rprintfFloat(8, GpsInfo.TimeOfWeek.f);
            rprintfCRLF();
      rprintfProgStrM("WkNum:    ");   rprintfNum(10,4,0,' ',GpsInfo.WeekNum);
            rprintfCRLF();
      rprintfProgStrM("UTCoffset:");   rprintfFloat(8, GpsInfo.UtcOffset.f);
            rprintfCRLF();
      rprintfProgStrM("Num SVs:  ");   rprintfNum(10,4,0,' ',GpsInfo.numSVs);
            rprintfCRLF();
```

```
	rprintfProgStrM("X_ECEF:   ");   rprintfFloat(8, GpsInfo.PosECEF.x.f);
		rprintfCRLF();
	rprintfProgStrM("Y_ECEF:   ");   rprintfFloat(8, GpsInfo.PosECEF.y.f);
		rprintfCRLF();
	rprintfProgStrM("Z_ECEF:   ");   rprintfFloat(8, GpsInfo.PosECEF.z.f);
		rprintfCRLF();
	rprintfProgStrM("TOF:      ");   rprintfFloat(8,
GpsInfo.PosECEF.TimeOfFix.f);    rprintfCRLF();
	rprintfProgStrM("Updates:  ");   rprintfNum(10,6,0,'
',GpsInfo.PosECEF.updates);      rprintfCRLF();

	//u08 str[20];
	//rprintfProgStrM(" PosLat: ");
	rprintfStr(dtostrf(GpsInfo.PosLat.f, 10, 5, str));
	rprintfProgStrM("PosLat:   ");   rprintfFloat(8,
180*(GpsInfo.PosLLA.lat.f/PI));  rprintfCRLF();
	rprintfProgStrM("PosLon:   ");   rprintfFloat(8,
180*(GpsInfo.PosLLA.lon.f/PI));  rprintfCRLF();
	rprintfProgStrM("PosLat:   ");   rprintfFloat(8,
(GpsInfo.PosLLA.lat.f));   rprintfCRLF();
	rprintfProgStrM("PosLon:   ");   rprintfFloat(8,
(GpsInfo.PosLLA.lon.f));   rprintfCRLF();
	rprintfProgStrM("PosAlt:   ");   rprintfFloat(8, GpsInfo.PosLLA.alt.f);
		rprintfCRLF();
	rprintfProgStrM("TOF:      ");   rprintfFloat(8,
GpsInfo.PosLLA.TimeOfFix.f);     rprintfCRLF();
	rprintfProgStrM("Updates:  ");   rprintfNum(10,6,0,'
',GpsInfo.PosLLA.updates); rprintfCRLF();

	rprintfProgStrM("Vel East: ");   rprintfFloat(8, GpsInfo.VelENU.east.f);
		rprintfCRLF();
	rprintfProgStrM("Vel North:");   rprintfFloat(8,
GpsInfo.VelENU.north.f);         rprintfCRLF();
	rprintfProgStrM("Vel Up:   ");   rprintfFloat(8, GpsInfo.VelENU.up.f);
		rprintfCRLF();
//	rprintfProgStrM("TOF:      ");   rprintfFloat(8,
GpsInfo.VelENU.TimeOfFix.f);     rprintfCRLF();
	rprintfProgStrM("Updates:  ");   rprintfNum(10,6,0,'
',GpsInfo.VelENU.updates); rprintfCRLF();

	rprintfProgStrM("Vel Head: ");   rprintfFloat(8,
GpsInfo.VelHS.heading.f);        rprintfCRLF();
	rprintfProgStrM("Vel Speed:");   rprintfFloat(8, GpsInfo.VelHS.speed.f);
		rprintfCRLF();
//	rprintfProgStrM("TOF:      ");   rprintfFloat(8,
GpsInfo.VelHS.TimeOfFix.f);      rprintfCRLF();
	rprintfProgStrM("Updates:  ");   rprintfNum(10,6,0,'
',GpsInfo.VelHS.updates);  rprintfCRLF();
*/

	lcdGotoXY(0,0);
	rprintfFloat(8, (GpsInfo.PosLLA.lat.f));
	rprintfFloat(6, GpsInfo.PosLLA.TimeOfFix.f);
	lcdGotoXY(0,1);
	rprintfFloat(9, (GpsInfo.PosLLA.lon.f));
	rprintf("G:");
	rprintfNum(10,3,0,' ',GpsInfo.PosLLA.updates);
	//rprintf("V:");
	//rprintfNum(10,3,0,' ',GpsInfo.VelHS.updates);


/*
		if(desiredLat>GpsInfo.PosLLA.lat.f)
```

```
                {      // go south
                       cbi(PORTB,1);
                       sbi(PORTB,0);
                }
                else if(desiredLat<GpsInfo.PosLLA.lat.f)
                {      // go north
                       sbi(PORTB,1);
                       cbi(PORTB,0);
                }
                else   cbi(PORTB,2);

                if(desiredLon>GpsInfo.PosLLA.lon.f)
                {      // go east
                       cbi(PORTB,3);
                       sbi(PORTB,4);
                }
                else if(desiredLon<GpsInfo.PosLLA.lon.f)
                {      // go west
                       sbi(PORTB,3);
                       cbi(PORTB,4);
                }
                else   cbi(PORTB,5);

        */
        }
```

Appendix O-12 ‘gps_asv.h’

```
//****************************************************************************
//
// File Name  : 'gps_asv.h'
// Title            : GPS position storage and processing function library
// Author           : Pascal Stang - Copyright (C) 2002
// Created          : 2002.08.29
// Revised          : 2002.08.29
// Version          : 0.1
// Target MCU : Atmel AVR Series
// Editor Tabs      : 4
//
// NOTE: This code is currently below version 1.0, and therefore is considered
// to be lacking in some functionality or documentation, or may not be fully
// tested.  Nonetheless, you can expect most functions to work.
//
// This code is distributed under the GNU Public License
//           which can be found at http://www.gnu.org/licenses/gpl.txt
//
// revised by Aaron Schooley
//****************************************************************************

#ifndef GPS_H
#define GPS_H

#include "global.h"

// constants/macros/typdefs
typedef union union_float_u32
{
      float f;
      unsigned long i;
      unsigned char b[4];
} float_u32;

typedef union union_double_u64
{
      double f;
      unsigned long long i;
      unsigned char b[8];
} double_u64;

struct PositionLLA
{
      float_u32 lat;
      float_u32 lon;
      float_u32 alt;
      float_u32 TimeOfFix;
      u16 updates;
};

struct VelocityENU
{
      float_u32 east;
      float_u32 north;
      float_u32 up;
      float_u32 TimeOfFix;
      u16 updates;
};

struct VelocityHS
{
      float_u32 heading;
```

```
	float_u32 speed;
	float_u32 TimeOfFix;
	u16 updates;
};

struct PositionECEF
{
	float_u32 x;
	float_u32 y;
	float_u32 z;
	float_u32 TimeOfFix;
	u16 updates;
};

struct VelocityECEF
{
	float_u32 x;
	float_u32 y;
	float_u32 z;
	float_u32 TimeOfFix;
	u16 updates;
};

typedef struct struct_GpsInfo
{
	float_u32 TimeOfWeek;
	u16 WeekNum;
	float_u32 UtcOffset;
	u08 numSVs;

	struct PositionLLA PosLLA;
	struct PositionECEF PosECEF;
	struct VelocityECEF VelECEF;
	struct VelocityENU VelENU;
	struct VelocityHS VelHS;

} GpsInfoType;

// functions
void gpsInit(void);
GpsInfoType* gpsGetInfo(void);
void gpsInfoPrint(void);

#endif
```

Appendix O-13 ‘nmea_asv.c’

```
//***************************************************************************
//
// File Name  : 'nmea_asv.c'
// Title            : NMEA protocol function library
// Author           : Pascal Stang - Copyright (C) 2002
// Created          : 2002.08.27
// Revised          : 2002.08.27
// Version          : 0.1
// Target MCU : Atmel AVR Series
// Editor Tabs      : 4
//
// NOTE: This code is currently below version 1.0, and therefore is considered
// to be lacking in some functionality or documentation, or may not be fully
// tested.  Nonetheless, you can expect most functions to work.
//
// This code is distributed under the GNU Public License
//           which can be found at http://www.gnu.org/licenses/gpl.txt
//
// revised by Aaron Schooley
//***************************************************************************

#ifndef WIN32
       #include <avr/io.h>
       #include <avr/signal.h>
       #include <avr/interrupt.h>
       #include <avr/pgmspace.h>
#endif
#include <string.h>
#include <stdlib.h>
#include <math.h>

#include "global.h"
#include "buffer.h"
#include "rprintf.h"
#include "gps_asv.h"
#include "lcd.h"
#include "timer.h"

#include "nmea_asv.h"

// Program ROM constants

// Global variables
extern GpsInfoType GpsInfo;
u08 NmeaPacket[NMEA_BUFFERSIZE];

u08 GPGGACount = 0;
u08 GPVTGCount = 0;

void nmeaInit(void)
{
}

u08* nmeaGetPacketBuffer(void)
{
       return NmeaPacket;
}

u08 nmeaProcess(cBuffer* rxBuffer)
{
       u08 foundpacket = NMEA_NODATA;
       u08 startFlag = FALSE;
```

```
	//u08 data;
	u16 i,j;

	// process the receive buffer
	// go through buffer looking for packets
	while(rxBuffer->datalength)
	{
		// look for a start of NMEA packet
		if(bufferGetAtIndex(rxBuffer,0) == '$')
		{
			// found start
			startFlag = TRUE;
			// when start is found, we leave it intact in the receive
buffer
			// in case the full NMEA string is not completely received.
The
			// start will be detected in the next nmeaProcess
iteration.

			// done looking for start
			break;
		}
		else
			bufferGetFromFront(rxBuffer);
	}

	// if we detected a start, look for end of packet
	if(startFlag)
	{
		for(i=1; i<(rxBuffer->datalength)-1; i++)
		{
			// check for end of NMEA packet <CR><LF>
			if((bufferGetAtIndex(rxBuffer,i) == '\r') &&
(bufferGetAtIndex(rxBuffer,i+1) == '\n'))
			{
				// have a packet end
				// dump initial '$'
				bufferGetFromFront(rxBuffer);
				// copy packet to NmeaPacket
				for(j=0; j<(i-1); j++)
				{
					// although NMEA strings should be 80
characters or less,
					// receive buffer errors can generate
erroneous packets.
					// Protect against packet buffer overflow
					if(j<(NMEA_BUFFERSIZE-1))
						NmeaPacket[j] =
bufferGetFromFront(rxBuffer);
					else
						bufferGetFromFront(rxBuffer);
				}
				// null terminate it
				NmeaPacket[j] = 0;
				// dump <CR><LF> from rxBuffer
				bufferGetFromFront(rxBuffer);
				bufferGetFromFront(rxBuffer);

				#ifdef NMEA_DEBUG_PKT
				rprintf("Rx NMEA packet type: ");
				rprintfStrLen(NmeaPacket, 0, 5);
				rprintfStrLen(NmeaPacket, 5, (i-1)-5);
				rprintfCRLF();
```

```
                        #endif
                        // found a packet
                        // done with this processing session
                        foundpacket = NMEA_UNKNOWN;
                        break;
                }
        }
    }

    if(foundpacket)
    {
        // check message type and process appropriately
        if(!strncmp(NmeaPacket, "GPGGA", 5))
        {
            // process packet of this type
            nmeaProcessGPGGA(NmeaPacket);
            // report packet type
            foundpacket = NMEA_GPGGA;
        }
        else if(!strncmp(NmeaPacket, "GPVTG", 5))
        {
            // process packet of this type
            nmeaProcessGPVTG(NmeaPacket);
            // report packet type
            foundpacket = NMEA_GPVTG;
        }
        else if(!strncmp(NmeaPacket, "GPRMC", 5))
        {
            // process packet of this type
            nmeaProcessGPRMC(NmeaPacket);
            // report packet type
            foundpacket = NMEA_GPRMC;
        }
    }
    else if(rxBuffer->datalength >= rxBuffer->size)
    {
        // if we found no packet, and the buffer is full
        // we're logjammed, flush entire buffer
        bufferFlush(rxBuffer);
    }
    return foundpacket;
}

void nmeaProcessGPGGA(u08* packet)
{
    u08 i;
    char* endptr;
//  double degrees, minutesfrac;

    #ifdef NMEA_DEBUG_GGA
    rprintf("NMEA: ");
    rprintfStr(packet);
    rprintfCRLF();
    #endif

    // start parsing just after "GPGGA,"
    i = 6;
    // attempt to reject empty packets right away
    if(packet[i]==',' && packet[i+1]==',')
        return;

    // get UTC time [hhmmss.sss]
    GpsInfo.PosLLA.TimeOfFix.f = strtod(&packet[i], &endptr);
```

```
	while(packet[i++] != ',');                    // next field: latitude

	// get latitude [ddmm.mmmmm]
	GpsInfo.PosLLA.lat.f = strtod(&packet[i], &endptr);
	// convert to pure degrees [dd.dddd] format
//	minutesfrac = modf(GpsInfo.PosLLA.lat.f/100, &degrees);
//	GpsInfo.PosLLA.lat.f = degrees + (minutesfrac*100)/60;
	// convert to radians
//	GpsInfo.PosLLA.lat.f *= (M_PI/180);
	while(packet[i++] != ',');                    // next field: N/S
indicator

	// correct latitute for N/S
	if(packet[i] == 'S') GpsInfo.PosLLA.lat.f = -GpsInfo.PosLLA.lat.f;
	while(packet[i++] != ',');                    // next field: longitude

	// get longitude [ddmm.mmmmm]
	GpsInfo.PosLLA.lon.f = strtod(&packet[i], &endptr);
	// convert to pure degrees [dd.dddd] format
//	minutesfrac = modf(GpsInfo.PosLLA.lon.f/100, &degrees);
//	GpsInfo.PosLLA.lon.f = degrees + (minutesfrac*100)/60;
	// convert to radians
//	GpsInfo.PosLLA.lon.f *= (M_PI/180);
	while(packet[i++] != ',');                    // next field: E/W
indicator

	// correct latitute for E/W
	if(packet[i] == 'W') GpsInfo.PosLLA.lon.f = -GpsInfo.PosLLA.lon.f;
	while(packet[i++] != ',');                    // next field: position
fix status

	// position fix status
	// 0 = Invalid, 1 = Valid SPS, 2 = Valid DGPS, 3 = Valid PPS
	// check for good position fix
	if( (packet[i] != '0') && (packet[i] != ',') )
	{
		GpsInfo.PosLLA.updates++;
		GPGGACount++;
		lcdGotoXY(0,2);
		rprintf("GPGGA");
		rprintfNum(10,3,0,' ',(GPGGACount));
	}
	else
	{
		lcdGotoXY(0,2);
		rprintf("OFFLINE");
	}
	while(packet[i++] != ',');                    // next field: satellites
used

	// get number of satellites used in GPS solution
	GpsInfo.numSVs = atoi(&packet[i]);
	while(packet[i++] != ',');                    // next field: HDOP
(horizontal dilution of precision)
	while(packet[i++] != ',');                    // next field: altitude

	// get altitude (in meters)
	GpsInfo.PosLLA.alt.f = strtod(&packet[i], &endptr);

	while(packet[i++] != ',');                    // next field: altitude
units, always 'M'
	while(packet[i++] != ',');                    // next field: geoid
seperation
```

```
	while(packet[i++] != ',');			// next field: seperation
units
	while(packet[i++] != ',');			// next field: DGPS age
	while(packet[i++] != ',');			// next field: DGPS
station ID
	while(packet[i++] != '*');			// next field: checksum


}

void nmeaProcessGPVTG(u08* packet)
{
	u08 i;
	char* endptr;

	#ifdef NMEA_DEBUG_VTG
	rprintf("NMEA: ");
	rprintfStr(packet);
	rprintfCRLF();
	#endif

	// start parsing just after "GPVTG,"
	i = 6;
	// attempt to reject empty packets right away
	if(packet[i]==',' && packet[i+1]==',')
		return;

	// get course (true north ref) in degrees [ddd.dd]
	GpsInfo.VelHS.heading.f = strtod(&packet[i], &endptr);
	while(packet[i++] != ',');			// next field: 'T'
	while(packet[i++] != ',');			// next field: course
(magnetic north)

	// get course (magnetic north ref) in degrees [ddd.dd]
	//GpsInfo.VelHS.heading.f = strtod(&packet[i], &endptr);
	while(packet[i++] != ',');			// next field: 'M'
	while(packet[i++] != ',');			// next field: speed
(knots)

	// get speed in knots
	//GpsInfo.VelHS.speed.f = strtod(&packet[i], &endptr);
	while(packet[i++] != ',');			// next field: 'N'
	while(packet[i++] != ',');			// next field: speed
(km/h)

	// get speed in km/h
	//GpsInfo.VelHS.speed.f = strtod(&packet[i], &endptr);
	while(packet[i++] != ',');			// next field: 'K'
	while(packet[i++] != ',');			// next field: mode
indicator

	if( (packet[i] != 'N') && (packet[i] != ',') )
	{
		GpsInfo.VelHS.updates++;
		GPVTGCount++;
		lcdGotoXY(0,3);
		rprintf("GPVTG");
		rprintfNum(10,3,0,' ',(GPVTGCount));
		cbi(PORTB,5);
		timerPause(100);
		sbi(PORTB,5);
	}
	else
```

```
	{
		lcdGotoXY(0,2);
		rprintf("OFFLINE");
	}                                                                    //
next field: checksum

	while(packet[i++] != '*');




}

void nmeaProcessGPRMC(u08* packet)
{

	u08 i;
	char* endptr;

	#ifdef NMEA_DEBUG_GGA
	rprintf("NMEA: ");
	rprintfStr(packet);
	rprintfCRLF();
	#endif

	// start parsing just after "GPRMC,"
	i = 6;
	// attempt to reject empty packets right away
	if(packet[i]==',' && packet[i+1]==',')
		return;

	// get UTC time [hhmmss.sss]
	GpsInfo.PosLLA.TimeOfFix.f = strtod(&packet[i], &endptr);
	while(packet[i++] != ',');                    // next field: position
fix status

	// position fix status
	// A = Valid, V = NAV Reciever Warning
	// check for good position fix
	if( (packet[i] != 'V') && (packet[i] != ',') )
	{
		GpsInfo.PosLLA.updates++;
		GPGGACount++;
		lcdGotoXY(0,2);
		rprintf("GPRMC");
		rprintfNum(10,3,0,' ',(GPGGACount));
	}
	else
	{
		lcdGotoXY(0,2);
		rprintf("OFFLINE");
	}
	while(packet[i++] != ',');                    // next field: latitude

	// get latitude [ddmm.mmmmm]
	GpsInfo.PosLLA.lat.f = strtod(&packet[i], &endptr);
	while(packet[i++] != ',');                    // next field: N/S
indicator

	// correct latitute for N/S
	if(packet[i] == 'S') GpsInfo.PosLLA.lat.f = -GpsInfo.PosLLA.lat.f;
	while(packet[i++] != ',');                    // next field: longitude
```

```
	// get longitude [ddmm.mmmmm]
	GpsInfo.PosLLA.lon.f = strtod(&packet[i], &endptr);
	while(packet[i++] != ',');                    // next field: E/W
indicator

	// correct latitute for E/W
	if(packet[i] == 'W') GpsInfo.PosLLA.lon.f = -GpsInfo.PosLLA.lon.f;
	while(packet[i++] != ',');                    // next field: speed
(knots)

	// get speed in knots
	//GpsInfo.VelHS.speed.f = strtod(&packet[i], &endptr);
	while(packet[i++] != ',');                    // next field: course
(true)

	// get course (true north ref) in degrees [ddd.dd]
	GpsInfo.VelHS.heading.f = strtod(&packet[i], &endptr);
	while(packet[i++] != ',');                    // next field: date of fix

	while(packet[i++] != ',');                    // next field: magnetic
variation
	while(packet[i++] != ',');                    // next field: magnetic
variation direction
	while(packet[i++] != ',');                    // next field: mode
inticator
	while(packet[i++] != '*');                    // next field: checksum

}
```

Appendix O-14 ‘nmea_asv.h’

```
//*****************************************************************************
//
// File Name  : 'nmea_asv.h'
// Title            : NMEA protocol function library
// Author           : Pascal Stang - Copyright (C) 2002
// Created          : 2002.08.27
// Revised          : 2002.08.27
// Version          : 0.1
// Target MCU : Atmel AVR Series
// Editor Tabs      : 4
//
// NOTE: This code is currently below version 1.0, and therefore is considered
// to be lacking in some functionality or documentation, or may not be fully
// tested.  Nonetheless, you can expect most functions to work.
//
// This code is distributed under the GNU Public License
//           which can be found at http://www.gnu.org/licenses/gpl.txt
//
// revised by Aaron Schooley
//*****************************************************************************

#ifndef NMEA_H
#define NMEA_H

#include "global.h"
#include "buffer.h"

// constants/macros/typdefs
#define NMEA_BUFFERSIZE          80

// Message Codes
#define NMEA_NODATA       0     // No data. Packet not available, bad, or not
decoded
#define NMEA_GPGGA        1     // Global Positioning System Fix Data
#define NMEA_GPVTG        2     // Course over ground and ground speed
#define NMEA_GPGLL        3     // Geographic position - latitude/longitude
#define NMEA_GPGSV        4     // GPS satellites in view
#define NMEA_GPGSA        5     // GPS DOP and active satellites
#define NMEA_GPRMC        6     // Recommended minimum specific GPS data
#define NMEA_UNKNOWN      0xFF// Packet received but not known

// Debugging
//#define NMEA_DEBUG_PKT   ///< define to enable debug of all NMEA messages
//#define NMEA_DEBUG_GGA   ///< define to enable debug of GGA messages
//#define NMEA_DEBUG_VTG   ///< define to enable debug of VTG messages

// functions
void nmeaInit(void);
u08* nmeaGetPacketBuffer(void);
u08 nmeaProcess(cBuffer* rxBuffer);
void nmeaProcessGPGGA(u08* packet);
void nmeaProcessGPVTG(u08* packet);
void nmeaProcessGPRMC(u08* packet);

#endif
```

Appendix O-15 ‘uart_asv.c’

```
//*****************************************************************************
//
// File Name  : 'uart_asv.c'
// Title            : UART driver with buffer support
// Author           : Pascal Stang - Copyright (C) 2000-2002
// Created          : 11/22/2000
// Revised          : 06/09/2003
// Version          : 1.3
// Target MCU : ATMEL AVR Series
// Editor Tabs      : 4
//
// This code is distributed under the GNU Public License
//           which can be found at http://www.gnu.org/licenses/gpl.txt
//
// revised by Aaron Schooley
//*****************************************************************************

#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/signal.h>

#include "buffer.h"
#include "uart_asv.h"

// UART global variables
// flag variables
volatile u08   uartReadyTx;                    ///< uartReadyTx flag
volatile u08   uartBufferedTx;          ///< uartBufferedTx flag
// receive and transmit buffers
cBuffer uartRxBuffer;                          ///< uart receive buffer
cBuffer uartTxBuffer;                          ///< uart transmit buffer
unsigned short uartRxOverflow;          ///< receive overflow counter

#ifndef UART_BUFFERS_EXTERNAL_RAM
       // using internal ram,
       // automatically allocate space in ram for each buffer
       static char uartRxData[UART_RX_BUFFER_SIZE];
       static char uartTxData[UART_TX_BUFFER_SIZE];
#endif

typedef void (*voidFuncPtru08)(unsigned char);
volatile static voidFuncPtru08 UartRxFunc;

//! enable and initialize the uart
void uartInit(void)
{
       // initialize the buffers
       uartInitBuffers();
       // initialize user receive handler
       UartRxFunc = 0;

       // enable RxD/TxD and interrupts
       outb(UCR, BV(RXCIE)|BV(TXCIE)|BV(RXEN)|BV(TXEN));

       // set default baud rate
       uartSetBaudRate(UART_DEFAULT_BAUD_RATE);
       // initialize states
       uartReadyTx = TRUE;
       uartBufferedTx = FALSE;
       // clear overflow count
       uartRxOverflow = 0;
       // enable interrupts
```

```
	sei();
}

//! create and initialize the uart transmit and receive buffers
void uartInitBuffers(void)
{
	#ifndef UART_BUFFERS_EXTERNAL_RAM
		// initialize the UART receive buffer
		bufferInit(&uartRxBuffer, uartRxData, UART_RX_BUFFER_SIZE);
		// initialize the UART transmit buffer
		bufferInit(&uartTxBuffer, uartTxData, UART_TX_BUFFER_SIZE);
	#else
		// initialize the UART receive buffer
		bufferInit(&uartRxBuffer, (u08*) UART_RX_BUFFER_ADDR,
UART_RX_BUFFER_SIZE);
		// initialize the UART transmit buffer
		bufferInit(&uartTxBuffer, (u08*) UART_TX_BUFFER_ADDR,
UART_TX_BUFFER_SIZE);
	#endif
}

//! redirects received data to a user function
void uartSetRxHandler(void (*rx_func)(unsigned char c))
{
	// set the receive interrupt to run the supplied user function
	UartRxFunc = rx_func;
}

//! set the uart baud rate
void uartSetBaudRate(u32 baudrate)
{
	// calculate division factor for requested baud rate, and set it
	u16 bauddiv = ((F_CPU+(baudrate*8L))/(baudrate*16L)-1);
	outb(UBRRL, bauddiv);
	#ifdef UBRRH
	outb(UBRRH, bauddiv>>8);
	#endif
}

//! returns the receive buffer structure
cBuffer* uartGetRxBuffer(void)
{
	// return rx buffer pointer
	return &uartRxBuffer;
}

//! returns the transmit buffer structure
cBuffer* uartGetTxBuffer(void)
{
	// return tx buffer pointer
	return &uartTxBuffer;
}

//! transmits a byte over the uart
void uartSendByte(u08 txData)
{
	// wait for the transmitter to be ready
	while(!uartReadyTx);
	// send byte
	outp( txData, UDR );
	// set ready state to FALSE
	uartReadyTx = FALSE;
}
```

```
//! gets a single byte from the uart receive buffer (getchar-style)
int uartGetByte(void)
{
	u08 c;
	if(uartReceiveByte(&c))
		return c;
	else
		return -1;
}

//! gets a byte (if available) from the uart receive buffer
u08 uartReceiveByte(u08* rxData)
{
	// make sure we have a receive buffer
	if(uartRxBuffer.size)
	{
		// make sure we have data
		if(uartRxBuffer.datalength)
		{
			// get byte from beginning of buffer
			*rxData = bufferGetFromFront(&uartRxBuffer);
			return TRUE;
		}
		else
		{
			// no data
			return FALSE;
		}
	}
	else
	{
		// no buffer
		return FALSE;
	}
}

//! flush all data out of the receive buffer
void uartFlushReceiveBuffer(void)
{
	// flush all data from receive buffer
	//bufferFlush(&uartRxBuffer);
	// same effect as above
	uartRxBuffer.datalength = 0;
}

//! return true if uart receive buffer is empty
u08 uartReceiveBufferIsEmpty(void)
{
	if(uartRxBuffer.datalength == 0)
	{
		return TRUE;
	}
	else
	{
		return FALSE;
	}
}

//! add byte to end of uart Tx buffer
void uartAddToTxBuffer(u08 data)
{
	// add data byte to the end of the tx buffer
```

```
	bufferAddToEnd(&uartTxBuffer, data);
}

//! start transmission of the current uart Tx buffer contents
void uartSendTxBuffer(void)
{
	// turn on buffered transmit
	uartBufferedTx = TRUE;
	// send the first byte to get things going by interrupts
	uartSendByte(bufferGetFromFront(&uartTxBuffer));
}
/*
//! transmit nBytes from buffer out the uart
u08 uartSendBuffer(char *buffer, u16 nBytes)
{
	register u08 first;
	register u16 i;

	// check if there's space (and that we have any bytes to send at all)
	if((uartTxBuffer.datalength + nBytes < uartTxBuffer.size) && nBytes)
	{
		// grab first character
		first = *buffer++;
		// copy user buffer to uart transmit buffer
		for(i = 0; i < nBytes-1; i++)
		{
			// put data bytes at end of buffer
			bufferAddToEnd(&uartTxBuffer, *buffer++);
		}

		// send the first byte to get things going by interrupts
		uartBufferedTx = TRUE;
		uartSendByte(first);
		// return success
		return TRUE;
	}
	else
	{
		// return failure
		return FALSE;
	}
}
*/
//! UART Transmit Complete Interrupt Handler
UART_INTERRUPT_HANDLER(SIG_UART_TRANS)
{
	// check if buffered tx is enabled
	if(uartBufferedTx)
	{
		// check if there's data left in the buffer
		if(uartTxBuffer.datalength)
		{
			// send byte from top of buffer
			outp( bufferGetFromFront(&uartTxBuffer), UDR );
		}
		else
		{
			// no data left
			uartBufferedTx = FALSE;
			// return to ready state
			uartReadyTx = TRUE;
		}
	}
```

```
	else
	{
		// we're using single-byte tx mode
		// indicate transmit complete, back to ready
		uartReadyTx = TRUE;
	}
}

//! UART Receive Complete Interrupt Handler
UART_INTERRUPT_HANDLER(SIG_UART_RECV)
{
	u08 c;

	// get received char
	c = inp(UDR);

	// if there's a user function to handle this receive event
	if(UartRxFunc)
	{
		// call it and pass the received data
		UartRxFunc(c);
	}
	else
	{
		// otherwise do default processing
		// put received char in buffer
		// check if there's space
		if( !bufferAddToEnd(&uartRxBuffer, c) )
		{
			// no space in buffer
			// count overflow
			uartRxOverflow++;
		}
	}
}
```

AppendixO-16 ‘uart_asv.h’

```
//***************************************************************************
//
// File Name  : 'uart_asv.h'
// Title            : UART driver with buffer support
// Author           : Pascal Stang - Copyright (C) 2000-2002
// Created          : 11/22/2000
// Revised          : 02/01/2004
// Version          : 1.3
// Target MCU : ATMEL AVR Series
// Editor Tabs      : 4
//
// This code is distributed under the GNU Public License
//           which can be found at http://www.gnu.org/licenses/gpl.txt
//
// revised by Aaron Schooley
//***************************************************************************

#ifndef UART_H
#define UART_H

#include "global.h"
#include "buffer.h"

//! default baud rate
//! can be changed by using uartSetBaudRate()
#define UART_DEFAULT_BAUD_RATE   9600

// buffer memory allocation defines
// buffer sizes
#ifndef UART_TX_BUFFER_SIZE
#define UART_TX_BUFFER_SIZE             0x0040 ///< number of bytes for uart
transmit buffer
#endif
#ifndef UART_RX_BUFFER_SIZE
#define UART_RX_BUFFER_SIZE             0x0050 ///< number of bytes for uart
receive buffer
#endif

// define this key if you wish to use
// external RAM for the    UART buffers
//#define UART_BUFFER_EXTERNAL_RAM
#ifdef UART_BUFFER_EXTERNAL_RAM
       // absolute address of uart buffers
       #define UART_TX_BUFFER_ADDR      0x1000
       #define UART_RX_BUFFER_ADDR      0x1100
#endif

// type of interrupt handler to use
// *do not change unless you know what you're doing
// Value may be SIGNAL or INTERRUPT
#ifndef UART_INTERRUPT_HANDLER
#define UART_INTERRUPT_HANDLER   SIGNAL
#endif

// compatibility with most newer processors
#ifdef UCSRB
       #define UCR                              UCSRB
#endif
// compatibility with old Mega processors
#if defined(UBRR) && !defined(UBRRL)
       #define      UBRRL                       UBRR
```

```
#endif
// compatibility with dual-uart processors
// (if you need to use both uarts, please use the uart2 library)
#if defined(__AVR_ATmega128__)
      #define UDR                           UDR0
      #define UCR                           UCSR0B
      #define UBRRL                   UBRR0L
      #define UBRRH                   UBRR0H
      #define SIG_UART_TRANS          SIG_UART0_TRANS
      #define SIG_UART_RECV           SIG_UART0_RECV
      #define SIG_UART_DATA           SIG_UART0_DATA
#endif
#if defined(__AVR_ATmega161__)
      #define UDR                           UDR0
      #define UCR                           UCSR0B
      #define UBRRL                   UBRR0
      #define SIG_UART_TRANS          SIG_UART0_TRANS
      #define SIG_UART_RECV           SIG_UART0_RECV
      #define SIG_UART_DATA           SIG_UART0_DATA
#endif

// functions

//! initializes transmit and receive buffers
// called from uartInit()
void uartInitBuffers(void);

//! initializes uart
void uartInit(void);

//! redirects received data to a user function
void uartSetRxHandler(void (*rx_func)(unsigned char c));

//! sets the uart baud rate
void uartSetBaudRate(u32 baudrate);

//! returns pointer to the receive buffer structure
cBuffer* uartGetRxBuffer(void);

//! returns pointer to the transmit buffer structure
cBuffer* uartGetTxBuffer(void);

//! sends a single byte over the uart
void uartSendByte(u08 data);

//! gets a single byte from the uart receive buffer (getchar-style)
// returns the byte, or -1 if no byte is available
int uartGetByte(void);

//! gets a single byte from the uart receive buffer
// Function returns TRUE if data was available, FALSE if not.
// Actual data is returned in variable pointed to by "data".
// example usage:
// char myReceivedByte;
// uartReceiveByte( &myReceivedByte );
u08 uartReceiveByte(u08* data);

//! returns TRUE/FALSE if receive buffer is empty/not-empty
u08 uartReceiveBufferIsEmpty(void);

//! flushes (deletes) all data from receive buffer
void uartFlushReceiveBuffer(void);
```

```
//! add byte to end of uart Tx buffer
void uartAddToTxBuffer(u08 data);

//! begins transmission of the transmit buffer under interrupt control
void uartSendTxBuffer(void);

//! sends a buffer of length nBytes via the uart using interrupt control
u08  uartSendBuffer(char *buffer, u16 nBytes);

#endif
```

Appendix O-17 ‘packet.c’

```
//***************************************************************************
//
// File Name  : 'packet.c'
// Title            : packet retrival and parsing protocol function library
//
// Updated          : 2005.3.14
//
//***************************************************************************

#ifndef WIN32
       #include <avr/io.h>
       #include <avr/signal.h>
       #include <avr/interrupt.h>
       #include <avr/pgmspace.h>
#endif
#include <stdlib.h>
#include <string.h>
#include <math.h>

#include "global.h"
#include "buffer.h"
#include "rprintf.h"
//#include "gps_asv.h"
#include "timer.h"
#include "lcd.h"

#include "i2c.h"

#include "packet.h"

// Program ROM constants

// Global variables
//extern GpsInfoType PacketInfo;
u08 Packet[PACKET_BUFFERSIZE];
long manualY;
long manualX;
u16 manualUpdates;

void packetInit(void)
{
}

u08* packetGetPacketBuffer(void)
{
       return Packet;
}

u08 getPacketProcess(cBuffer* rxBuffer)
{

//     rprintf("getPacketProcess\r\n");

       u08 foundpacket = PACKET_NODATA;
       u08 startFlag = FALSE;
       //u08 data;
       u16 i,j;
       j=0;


       // process the receive buffer
       // go through buffer looking for packets
```

```
	while(rxBuffer->datalength)
	{
		// look for a start of NMEA packet
		if(bufferGetAtIndex(rxBuffer,0) == '$')
		{
//			rprintf("   startFlag\r\n");
			// found start
			startFlag = TRUE;
			// when start is found, we leave it intact in the receive
buffer
			// in case the full NMEA string is not completely received.
The
			// start will be detected in the next nmeaProcess
iteration.

			// done looking for start
			break;
		}
		else
			bufferGetFromFront(rxBuffer);
	}

	// if we detected a start, look for end of packet
	if(startFlag)
	{
		for(i=1; i<(rxBuffer->datalength)-1; i++)
		{
			// check for end of NMEA packet <CR><LF>
			if((bufferGetAtIndex(rxBuffer,i) == '\r') &&
(bufferGetAtIndex(rxBuffer,i+1) == '\n'))
			{
//				rprintf("      endFlag\r\n");
				// have a packet end
				// dump initial '$'
				bufferGetFromFront(rxBuffer);
				// copy packet to NmeaPacket
				for(j=0; j<(i-1); j++)
				{
					// although NMEA strings should be 80
characters or less,
					// receive buffer errors can generate
erroneous packets.
					// Protect against packet buffer overflow
					if(j<(PACKET_BUFFERSIZE-1))
						Packet[j] =
bufferGetFromFront(rxBuffer);
					else
						bufferGetFromFront(rxBuffer);
				}
				// null terminate it
				Packet[j] = 0;
				// dump <CR><LF> from rxBuffer
				bufferGetFromFront(rxBuffer);
				bufferGetFromFront(rxBuffer);

				#ifdef PACKET_DEBUG_PKT
				rprintf("Rx packet type: ");
				rprintfStrLen(Packet, 0, 5);
				rprintfStrLen(Packet, 5, (i-1)-5);
				rprintfCRLF();
				#endif
				// found a packet
				// done with this processing session
```

```
                        foundpacket = PACKET_UNKNOWN;
                        break;
                }
        }
    }

//  rprintfStr(Packet);


    if(foundpacket)
    {
//      rprintf(" FP");
/*      // check message type and process appropriately
        if(!strncmp(Packet, "A", 1))
        {
            // process packet of this type
            packetProcessAuto(Packet);
            // report packet type
            foundpacket = PACKET_AUTO;
        }
        else if(!strncmp(Packet, "C", 1))
        {
//          rprintf(" C");
//          rprintf(" %d",j);

//          i2cMasterSend(0x20,9,Packet);

            // process packet of this type
            packetProcessManual(Packet);
            // report packet type
            foundpacket = PACKET_MANUAL;
        }*/
    }
    else if(rxBuffer->datalength >= rxBuffer->size)
    {
        // if we found no packet, and the buffer is full
        // we're logjammed, flush entire buffer
        bufferFlush(rxBuffer);
    }

//  timerPause(100);
//  lcdClear();

    return foundpacket;
}


void packetProcessAuto(u08* packet)
{/*
    u08 i;
    char* endptr;
    double degrees, minutesfrac;

    #ifdef NMEA_DEBUG_GGA
    rprintf("NMEA: ");
    rprintfStr(packet);
    rprintfCRLF();
    #endif

    // start parsing just after "GPGGA,"
    i = 6;
    // attempt to reject empty packets right away
    if(packet[i]==',' && packet[i+1]==',')
```

```
            return;

      // get UTC time [hhmmss.sss]
      GpsInfo.PosLLA.TimeOfFix.f = strtod(&packet[i], &endptr);
      while(packet[i++] != ',');                    // next field: latitude

      // get latitude [ddmm.mmmmm]
      GpsInfo.PosLLA.lat.f = strtod(&packet[i], &endptr);
      // convert to pure degrees [dd.dddd] format
      minutesfrac = modf(GpsInfo.PosLLA.lat.f/100, &degrees);
      GpsInfo.PosLLA.lat.f = degrees + (minutesfrac*100)/60;
      // convert to radians
      GpsInfo.PosLLA.lat.f *= (M_PI/180);
      while(packet[i++] != ',');                    // next field: N/S
indicator

      // correct latitute for N/S
      if(packet[i] == 'S') GpsInfo.PosLLA.lat.f = -GpsInfo.PosLLA.lat.f;
      while(packet[i++] != ',');                    // next field: longitude

      // get longitude [ddmm.mmmmm]
      GpsInfo.PosLLA.lon.f = strtod(&packet[i], &endptr);
      // convert to pure degrees [dd.dddd] format
      minutesfrac = modf(GpsInfo.PosLLA.lon.f/100, &degrees);
      GpsInfo.PosLLA.lon.f = degrees + (minutesfrac*100)/60;
      // convert to radians
      GpsInfo.PosLLA.lon.f *= (M_PI/180);
      while(packet[i++] != ',');                    // next field: E/W
indicator

      // correct latitute for E/W
      if(packet[i] == 'W') GpsInfo.PosLLA.lon.f = -GpsInfo.PosLLA.lon.f;
      while(packet[i++] != ',');                    // next field: position
fix status

      // position fix status
      // 0 = Invalid, 1 = Valid SPS, 2 = Valid DGPS, 3 = Valid PPS
      // check for good position fix
      if( (packet[i] != '0') && (packet[i] != ',') )
            GpsInfo.PosLLA.updates++;
      while(packet[i++] != ',');                    // next field: satellites
used

      // get number of satellites used in GPS solution
      GpsInfo.numSVs = atoi(&packet[i]);
      while(packet[i++] != ',');                    // next field: HDOP
(horizontal dilution of precision)
      while(packet[i++] != ',');                    // next field: altitude

      // get altitude (in meters)
      GpsInfo.PosLLA.alt.f = strtod(&packet[i], &endptr);

      while(packet[i++] != ',');                    // next field: altitude
units, always 'M'
      while(packet[i++] != ',');                    // next field: geoid
seperation
      while(packet[i++] != ',');                    // next field: seperation
units
      while(packet[i++] != ',');                    // next field: DGPS age
      while(packet[i++] != ',');                    // next field: DGPS
station ID
      while(packet[i++] != '*');                    // next field: checksum
*/}
```

```
void packetProcessManual(u08* packet)
{

//     lcdGotoXY(0,1);
//     rprintf("\r\n");
//     rprintfStr(packet);
//     timerPause(1000);
//     rprintf("\r\n");
//     lcdClear();


       u08 i;

       char* endptr;

       #ifdef PACKET_DEBUG_VTG
       rprintf("PACKET: ");
       rprintfStr(packet);
       rprintfCRLF();
       #endif

       // start parsing just after "C,"
       i = 2;
       // attempt to reject empty packets right away
       if(packet[i]==',' && packet[i+1]==',')
              return;

       // get x movement
       manualX = strtol(&packet[i], &endptr, 10);
       while(packet[i++] != ',');

       // get y movement
       manualY = strtol(&packet[i], &endptr, 10);
       while(packet[i++] != '*');

       // increment updates
       manualUpdates++;
}
```

Appendix O-18 ‘packet.h’

```
//*****************************************************************************
//
// File Name  : 'packet.h'
// Title            : packet retrivial and parsing protocol function library
//
//*****************************************************************************

#ifndef PACKET_H
#define PACKET_H


#include "global.h"
#include "buffer.h"

// constants/macros/typdefs
#define PACKET_BUFFERSIZE        80

// Message Codes
#define PACKET_NODATA            0      // No data. Packet not available, bad,
or not decoded
#define PACKET_AUTO              1      // Global Positioning System Fix Data
#define PACKET_MANUAL            2      // Course over ground and ground speed
#define PACKET_UNKNOWN     0xFF// Packet received but not known

// Debugging
//#define PACKET_DEBUG_PKT ///< define to enable debug of all PACKET messages
//#define PACKET_DEBUG_GGA ///< define to enable debug of GGA messages
//#define PACKET_DEBUG_VTG ///< define to enable debug of VTG messages

// functions
void packetInit(void);
u08* packetGetPacketBuffer(void);
u08 getPacketProcess(cBuffer* rxBuffer);
void packetProcessAuto(u08* packet);
void packetProcessManual(u08* packet);

#endif
```

Appendix O-19 comm.c

```
//*****************************************************************************
// File Name  : comm.c
//
// Title            : Communtications controller for SCU ASV
// Description        responsible for communitcating between system controllers
//              via i2c and the shore computer via rs232 and a wireless link
//
// Revision         : 1.0
// Notes            :
// Target MCU : Atmel AVR series
// Editor Tabs      : 4
//
// Revision History:
// When                    Who                 Description of change
// -----------      -----------  -----------------------
// 03-Feb-2003      aschooley    Created the program
//*****************************************************************************


//----- Include Files ---------------------------------------------------------
#include <avr/io.h>        // include I/O definitions (port names, pin names,
etc)
#include <avr/signal.h>    // include "signal" names (interrupt names)
#include <avr/interrupt.h> // include interrupt support

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

#include "global.h"        // include our global settings
#include "uart_asv.h"             // include uart function library
#include "rprintf.h"       // include printf function library
#include "timer.h"         // include timer function library (timing, PWM, etc)
//#include "vt100.h"              // include VT100 terminal support
#include "i2c.h"           // include i2c support
//#include "debug.h"              // include debug functions
#include "packet.h"
#include "gps_asv.h"
#include "lcd.h"



//----- Defines ---------------------------------------------------------------
-
#define LOCAL_ADDR  0x10
#define GPS                0x20
#define CMPS        0x30
#define MOTOR       0x40
#define SCI                0x50
#define PANTLT             0x60

//----- Globals ---------------------------------------------------------------
-
// local data buffer

u08 localBufferLength = 10;
u08 localBuffer[10];
u08 data[10];
//extern GpsInfoType PacketInfo;
extern u08 Packet[PACKET_BUFFERSIZE];
extern long manualX;
extern long manualY;
```

```
extern u16 manualUpdates;
u08    destination = 0;


//----- Prototypes ------------------------------------------------------------
void getPacketAndTransmitt(void);
void i2cSlaveReceiveService(u08 receiveDataLength, u08* receiveData);
//u08 i2cSlaveTransmitService(u08 transmitDataLengthMax, u08* transmitData);

//----- Begin Code ------------------------------------------------------------
int main(void)
{
       // initialize our libraries
       // initialize the UART (serial port)
       uartInit();
       // initialize the timer system
       timerInit();
       // make all rprintf statements use uart for output
       rprintfInit(uartSendByte);
       // initialize LCD
//     lcdInit();
       // direct printf output to LCD
//     rprintfInit(lcdDataWrite);


       // initialize i2c function library
       i2cInit();
       // set local device address and allow response to general call
       i2cSetLocalDeviceAddr(LOCAL_ADDR, TRUE);
       // set the Slave Receive Handler function
       // (this function will run whenever a master somewhere else on the bus
       //  writes data to us as a slave)
       i2cSetSlaveReceiveHandler( i2cSlaveReceiveService );
       // set the Slave Transmit Handler function
       // (this function will run whenever a master somewhere else on the bus
       //  attempts to read data from us as a slave)
//     i2cSetSlaveTransmitHandler( i2cSlaveTransmitService );
       i2cSetBitrate(400);


       // print a little intro message so we know things are working
//     lcdClear();
       rprintf("ASV COMM COMPUTER ONLINE\r\n");
       timerPause(1000);
//     lcdClear();

       // set port b to output leds and turn lights off
//     outb(DDRB, 0xFF);
//     outb(PORTB,0xFF);

       // program loop
       while(1)
       {

              getPacketAndTransmitt();

       }

       return 0;
}


void getPacketAndTransmitt(void)
```

```
{

	int i;
	int genCS = 0;
	char genCSstr[80];
	for(i=0;i<localBufferLength;i++)
	{
		localBuffer[i]=0;
	}


	while( getPacketProcess(uartGetRxBuffer()) )
	{
		//print/dump current formatted GPS data
//		rprintfCRLF();


		for(i=0;Packet[i]!='*';i++)
		{
			genCS = genCS^Packet[i];
			rprintf("%d",i);
		}
		i++;
//		rprintf("  %x",genCS);

		u08 size;
		size=sprintf(genCSstr,"%x",genCS);

		u08 transCSstr[5];
		transCSstr[0]=Packet[i];i++;
		transCSstr[1]=Packet[i];i++;
		transCSstr[2]=0;

		int valid = strcmp(genCSstr, transCSstr);

		if(valid==0)
			rprintf("  +\r\n");
		else
			rprintf("  -\r\n");
/*
		rprintf("Gen:");
		rprintfStr(genCSstr);
		rprintf("  tans:");
		rprintfStr(transCSstr);
		rprintf("  Valid:%d",valid);

		rprintfCRLF();
		rprintf("packet:");
		rprintfStr(Packet);
		rprintfCRLF();
*/


		if(Packet[0]=='C')
		{
			destination = MOTOR;
			i2cMasterSend(destination, i, Packet);
		}
		if(Packet[0]=='A')
		{
			destination = MOTOR;
			i2cMasterSend(destination, i, Packet);
		}
```

```
		if(Packet[0]=='?')
		{

	if(Packet[2]=='E'||Packet[2]=='V'||Packet[2]=='I'||Packet[2]=='M')
			{
				destination = MOTOR;

					i2cMasterSend(destination,1,&Packet[2]);
					timerPause(50);
					i2cMasterReceive(destination,6,localBuffer);
					rprintfStr(localBuffer);
			}
			else if(Packet[2]=='L'||Packet[2]=='l'||Packet[2]=='T')
			{
				destination = GPS;

					i2cMasterSend(destination,1,&Packet[2]);
					timerPause(1);
					i2cMasterReceive(destination,6,localBuffer);
					rprintfStr(localBuffer);


			}
			else if(Packet[2]=='H')
			{
				destination = CMPS;
				//rprintf("requesting info from Commpass\r\n");
				i2cMasterReceive(destination,4,localBuffer);
				rprintfStr(localBuffer);
			}
		}
		if(Packet[0]=='S')
		{
			if(Packet[2]=='I'||Packet[2]=='A'||Packet[2]=='G')
				destination = MOTOR;
			i2cMasterSend(destination, i, Packet);
		}
		if(Packet[0]=='P')
		{
			destination = PANTLT;
			i2cMasterSend(destination, i, Packet);
		}
		if(Packet[0]=='E')
		{
			destination = SCI;
			i2cMasterSend(destination, i, Packet);
		}

	}
}



// slave operations
void i2cSlaveReceiveService(u08 receiveDataLength, u08* receiveData)
{
	u08 i;

	// this function will run when a master somewhere else on the bus
	// addresses us and wishes to write data to us

	//showByte(*receiveData);
//	cbi(PORTB, PB6);
```

```
	// copy the received data to a local buffer
	for(i=0; i<receiveDataLength; i++)
	{
		localBuffer[i] = *receiveData++;
	}
	localBufferLength = receiveDataLength;

	rprintfStr(localBuffer);
}

/*
u08 i2cSlaveTransmitService(u08 transmitDataLengthMax, u08* transmitData)
{
	u08 i;

	// this function will run when a master somewhere else on the bus
	// addresses us and wishes to read data from us

	//showByte(*transmitData);
	cbi(PORTB, PB7);

	// copy the local buffer to the transmit buffer
	for(i=0; i<localBufferLength; i++)
	{
		*transmitData++ = localBuffer[i];
	}

	localBuffer[0]++;

	return localBufferLength;
}*/
```

Appendix O-20 sci.c

```
//***************************************************************************
// File Name   : sci.c
//
// Title               : science controller for ASV, currently only has winch control
// Revision            : 1.0
// Notes               :
// Target MCU  : Atmel AVR series
// Editor Tabs : 4
//
// Revision History:
// When               Who                   Description of change
// ----------- -----------    ----------------------
// 10-Sep-2002 aschooley             Created the program
//***************************************************************************


//----- Include Files ---------------------------------------------------------
#include <avr/io.h>           // include I/O definitions (port names, pin names, etc)
#include <avr/signal.h>       // include "signal" names (interrupt names)
#include <avr/interrupt.h>    // include interrupt support

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

#include "global.h"           // include our global settings
#include "uart_asv.h"         // include function library _asv adds a bigger buff
#include "rprintf.h"   // include printf function library
#include "timer.h"            // include timer function library (timing, PWM, etc)
#include "lcd.h"              // include LCD function library
#include "i2c.h"              // include i2c function library

//----- Defines ---------------------------------------------------------------
#define LOCAL_ADDR    0x50


//----- Global Variables ------------------------------------------------------

u08 localBuffer[80];
u08 localBufferLength = 80;
u08 transmitStr[10];
u16 keepAliveStage1 = 0;
u16 keepAliveStage2 = 0;


//----- Prototypes ------------------------------------------------------------
//u08 i2cSlaveTransmitService(u08 transmitDataLengthMax, u08* transmitData);
void i2cSlaveReceiveService(u08 receiveDataLength, u08* receiveData);

//----- Begin Code ------------------------------------------------------------
int main(void)
{
       // initialize our libraries
       // initialize the UART (serial port)
       uartInit();
       // set the baud rate of UART 0 for our debug/reporting output
       uartSetBaudRate(9600);
       // initialize the timer system
       timerInit();
       // send printf to the UART
       rprintfInit(uartSendByte);

       // initialize the I2c bus
       i2cInit();
       // set local device address and allow response to general call
       i2cSetLocalDeviceAddr(LOCAL_ADDR, TRUE);
       // set the Slave Receive Handler function
       // (this function will run whenever a master somewhere else on the bus
       //  writes data to us as a slave)
```

```
        i2cSetSlaveReceiveHandler( i2cSlaveReceiveService );
        // set the Slave Transmit Handler function
        // (this function will run whenever a master somewhere else on the bus
        //  attempts to read data from us as a slave)
//      i2cSetSlaveTransmitHandler( i2cSlaveTransmitService );
        // set i2c bitrate to 400KHz
        i2cSetBitrate(400);


        //set port b to output
        outb(DDRB, 0xFF);
        // set port b pins low
        sbi(PORTB,0);
        sbi(PORTB,1);

        // Print hello world message
        rprintf("Science Controller");
        timerPause(1000);


        // start program loop
        while(1)
        {

                // look for what command has been sent
                switch(localBuffer[2])
                {
                case 'U':                               // winch up
                        // printf for debug
                        rprintf("up\r");
                        // turn on port bit
                        cbi(PORTB,0);
                        // reset our watchdog
                        keepAliveStage1 = 0;
                        keepAliveStage2 = 0;
                        // clear the buffer value
                        localBuffer[2] = 0;
                        break;
                case 'D':                               // winch down
                        // printf for debug
                        rprintf("down\r");
                        // turn on port bit
                        cbi(PORTB,1);
                        // reset our watchdog
                        keepAliveStage1 = 0;
                        keepAliveStage2 = 0;
                        // clear the buffer value
                        localBuffer[2] = 0;
                        break;
                case 'O':                               // winch off
                        // printf for debug
                        rprintf("off\r");
                        // turn off port bits
                        sbi(PORTB,0);
                        sbi(PORTB,1);
                        // clear the buffer value
                        localBuffer[2] = 0;
                        break;
                }

                // turn off the winch if no update is recieved when this counts up
                if(keepAliveStage1 == 65535)
                {
                        keepAliveStage2++;
                        if(keepAliveStage2 == 3)
                        {
                                // turn off port bits
                                sbi(PORTB,0);
                                sbi(PORTB,1);
                        }
                }
```

```
                keepAliveStage1++;


        }
        return 0;

}

/*u08 i2cSlaveTransmitService(u08 transmitDataLengthMax, u08* transmitData)
{
        // this function will run when a master somewhere else on the bus
        // addresses us and wishes to read data from us

        // counter variable
        u08 i;

        // copy the local buffer to the transmit buffer
        for(i=0; i<length; i++)
        {
                *transmitData++ = transmitStr[i];
        }

        return length;
}
*/

void i2cSlaveReceiveService(u08 receiveDataLength, u08* receiveData)
{
        u08 i;

        // this function will run when a master somewhere else on the bus
        // addresses us and wishes to write data to us

        // copy the received data to a local buffer
        for(i=0; i<receiveDataLength; i++)
        {
                localBuffer[i] = *receiveData++;
        }
        localBufferLength = receiveDataLength;

        localBuffer[i] = 0;

}
```

Appendix O-21 pantlt.c

```
//*************************************************************************
// File Name  : pantlt.c
//
// Title            : dc motor controller for camera pan
// Revision         : 1.0
// Notes            :
// Target MCU: Atmel AVR series
// Editor Tabs      : 4
//
// Revision History:
// When                    Who                 Description of change
// -----------      -----------  ----------------------
// 10-Sep-2002      aschooley           Created the program
//*************************************************************************


//----- Include Files ---------------------------------------------------
#include <avr/io.h>        // include I/O definitions (port names, pin names,
etc)
#include <avr/signal.h>    // include "signal" names (interrupt names)
#include <avr/interrupt.h> // include interrupt support

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

#include "global.h"        // include our global settings
#include "uart_asv.h"             // include function library _asv adds a bigger
buff
#include "rprintf.h"       // include printf function library
#include "timer.h"         // include timer function library (timing, PWM, etc)
#include "gps_asv.h"              // include gps data support
#include "nmea_asv.h"             // include NMEA gps packet handling
#include "lcd.h"           // include LCD function library
#include "i2c.h"           // include i2c function library

//----- Defines -------------------------------------------------------------
#define LOCAL_ADDR  0x60


//----- Global Variables ----------------------------------------------------
u08 localBuffer[80];
u08 localBufferLength = 80;
u08 slaveServiceStatus = 0;  //written to 1 if slave service is in use
u08 slaveRecieveCounter = 0;

//----- Prototypes ----------------------------------------------------------

// uartRxOverflow is a global variable defined in uart.c/uart2.c
// we define it here as <extern> here so that we can use its value
// in code contained in this file
//extern unsigned short uartRxOverflow[2];

u08 i2cSlaveTransmitService(u08 transmitDataLengthMax, u08* transmitData);
void i2cSlaveReceiveService(u08 receiveDataLength, u08* receiveData);

//----- Begin Code ------------------------------------------------------------
int main(void)
{
      // initialize our libraries
      // initialize the UART (serial port)
      uartInit();
```

```
	// set the baud rate of UART 0 for our debug/reporting output
	uartSetBaudRate(9600);
	// initialize the timer system
	timerInit();
	// initialize LCD
//	lcdInit();
	// direct printf output to LCD
//	rprintfInit(lcdDataWrite);
	rprintfInit(uartSendByte);

	// initialize the I2c bus
	i2cInit();
	// set local device address and allow response to general call
	i2cSetLocalDeviceAddr(LOCAL_ADDR, TRUE);
	// set the Slave Receive Handler function
	// (this function will run whenever a master somewhere else on the bus
	//  writes data to us as a slave)
	i2cSetSlaveReceiveHandler( i2cSlaveReceiveService );
	// set the Slave Transmit Handler function
	// (this function will run whenever a master somewhere else on the bus
	//  attempts to read data from us as a slave)
	i2cSetSlaveTransmitHandler( i2cSlaveTransmitService );
	// set i2c bitrate to 400KHz
	i2cSetBitrate(400);


	// set port b to output for test lights
//	outb(DDRB, 0xFF);
	// set port b pins high
//	outb(PORTB,0xFF);

	// Print hello world message
	rprintf("Compass System");
	timerPause(1000);
//	lcdClear();


	// start program loop
	while(1)
	{


	}
	return 0;

}



u08 i2cSlaveTransmitService(u08 transmitDataLengthMax, u08* transmitData)
{
	// this function will run when a master somewhere else on the bus
	// addresses us and wishes to read data from us

	//indicate that this service is running
	slaveServiceStatus = 1;

	// counter variable
	u08 i;

	// copy the local buffer to the transmit buffer
	for(i=0; i<length; i++)
	{
```

```
                *transmitData++ = headingStr[i];
        }

//      timeOfFixStr[0]++;

        //incidate that this service is done
        slaveServiceStatus = 0;

        return length;
}

void i2cSlaveReceiveService(u08 receiveDataLength, u08* receiveData)
{
        u08 i;

        slaveRecieveCounter++;

        // this function will run when a master somewhere else on the bus
        // addresses us and wishes to write data to us



        // copy the received data to a local buffer
        for(i=0; i<receiveDataLength; i++)
        {
                localBuffer[i] = *receiveData++;
        }
        localBufferLength = receiveDataLength;

        localBuffer[i] = 0;

}
```

Appendix O-22 Joystick Coding

```
/*
 * Created on Feb 14, 2005
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
package com.netrol.Data;

/**
 * @author nmehdizadeh
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
public class ControlCommand extends SingleValueCommnicationPacket {

	public ControlCommand()
	{
		setType('C');
	}

}



/*
 * Created on Feb 19, 2005
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
package com.netrol;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Vector;

/**
 * @author nmehdizadeh
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
public class DataBaseAccess {

	public static void insertRobotCommands(String command) throws Exception
	{
		Connection con = openSQLConnection();
		String sQuery = "insert into robotcomanddata (command) values('" +
command + "')";
		Statement stmt = con.createStatement();
		try
		{
			int updatedRows = stmt.executeUpdate(sQuery);
		}
		finally
		{
			stmt.close();
			con.close();
```

```
		}
	}

	public static void insertRobotStatus(String statusData) throws Exception
	{
		Connection con = openSQLConnection();
		String sQuery = "insert into robotstatusdata (robotstatus)
values('" + statusData + "')";
		Statement stmt = con.createStatement();
		try
		{
			int updatedRows = stmt.executeUpdate(sQuery);
		}
		finally
		{
			stmt.close();
			con.close();
		}
	}

	public static Vector fetchRobotStatusData(String sQuery)
	{
		String query = "select * from robotstatusdata where " + sQuery;
		Connection con = openSQLConnection();
		Vector result = new Vector();

		Statement stmt = null;
		ResultSet rs = null;

		try {
			stmt = con.createStatement();
			rs = stmt.executeQuery(query);
			if (rs != null)
			{
				while(rs.next())
				{
					result.add(rs.getString("robotstatus"));
				}
			}
		} catch (SQLException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
		finally
		{
			try {
				stmt.close();
				con.close();
			} catch (Exception e1) {
				// TODO Auto-generated catch block
				//e1.printStackTrace();
			}

		}

		return result;
	}

	public static Connection openSQLConnection() {
		String errMsg = "Unable to open SQL connection.";
		// Connection reference
		Connection sqlconn = null;
		try {
```

```
                    // Load database driver
                    Class.forName("com.mysql.jdbc.Driver");
                    // Make connection
                    sqlconn = DriverManager

      .getConnection("jdbc:mysql://localhost/test?user=root&password=admin");
              } catch (Exception excp) {
                    excp.printStackTrace();
              }
              return sqlconn;
      }
}



/*
 * Created on Feb 6, 2005
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
package com.netrol;

import java.util.Vector;

import com.netrol.Data.ControlCommand;
import com.netrol.Data.MultiValueCommnicationPacket;

/**
 * @author nmehdizadeh
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
public class DataPipe {

      public static RobotWritePort robot = new RobotWritePort();

      public static void storeDataFromRobotToDatabase(byte[] rawData)
      {
              try {
                    DataBaseAccess.insertRobotStatus(new String(rawData));
              } catch (Exception e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
              }
      }

      public static void sendCommandToRobot(MultiValueCommnicationPacket
newData)
      {

              int iNumOfData = newData.size();

              for (int i=0; i < iNumOfData; i++)
              {
                    String command = newData.getStringCommandAt(i);
                    System.out.println("row at " + i + " is " + command);
                    try {
                          writeDataToRobot(command);
                          DataBaseAccess.insertRobotCommands(command);
                    } catch (Exception e) {
                          // TODO Auto-generated catch block
```

```
                        e.printStackTrace();
                }
        }
    }

    public static void sendControlToRobot(ControlCommand cmd)
    {
        try {
            String command = cmd.toString();
            writeDataToRobot(command);
            DataBaseAccess.insertRobotCommands(command);
            System.out.println(command);
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
    public static Vector getData(String sQuery)
    {
        return DataBaseAccess.fetchRobotStatusData(sQuery);
    }

    private static void writeDataToRobot(String data)
    {
        robot.writeData(data);
    }

}



/*
 * Created on Feb 6, 2005
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
package com.netrol;

import java.io.BufferedReader;
import java.io.FileReader;
import java.util.Random;

/**
 * @author nmehdizadeh
 *
 * TODO To change the template for this generated type comment go to Window -
 * Preferences - Java - Code Style - Code Templates
 */
public class DataReader implements Runnable {

    private String dataFile = "c:\\data.txt";

    public static boolean runFlag = true;

    Random random = new Random(System.currentTimeMillis());

    static {
        DataReader dataReader = new DataReader();
        Thread thread = new Thread(dataReader);
        thread.start();

    }
```

```
	public void run() {
		try {
			this.startReading();
		} catch (Exception e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
	}

	public void startReading() throws Exception {
		FileReader fl = new FileReader(dataFile);
		BufferedReader br = new BufferedReader(fl);
		String readData = "";
		while (runFlag && readData != null) {
			int readLines = getRandom();
			readData = br.readLine();
			for (int i = 0; i < readLines && readData != null; i++) {
				System.out.println("Read " + readData);
				//Messages dataRow = new Messages();
				//dataRow.setRawData(readData);
				//MessageCache.addDataFields(dataRow);
				DataBaseAccess.insertRobotCommands(readData);
				readData = br.readLine();
			}
			Thread.sleep(getRandom() * 1000);
		}
		br.close();
		fl.close();
	}

	public int getRandom() {
		return random.nextInt(6);
	}

}



/*
 * Created on Feb 20, 2005
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
package com.netrol.dummyRobot;


import java.util.Enumeration;
import java.util.Hashtable;
import java.util.Vector;

import javax.comm.CommPortIdentifier;
import javax.comm.SerialPort;

import screem.netrol.communication.ChannelException;
import screem.netrol.communication.event.DataTurbineListener;
import screem.netrol.communication.event.ErrorEvent;
import screem.netrol.communication.event.ErrorListener;
import screem.netrol.communication.event.SerialPortConnectionListener;
import screem.netrol.serialport.SerialPortConnection;
import screem.netrol.serialport.SerialPortException;
import screem.netrol.serialport.event.SerialPortDataEvent;
```

```
import screem.netrol.serialport.event.SerialPortDataListener;

/**
* The <code>Robot</code> class manages the connection to the DataTurbine,
communication between the robot and
* the opearator and maintains the interface between the robot hardware and the
computer. The robot hardware and
* computer communicate on the computer's serial (COM) port(s). This
communication interface is implemented in the
* <code>SerialPortConnection</code> class, setting the communication parameters
and the I/O streams.
* <p>
* Following the <code>Channel</code> naming scheme defined by the DataTurbine
API, the
* <code>edu.scu.engr.screem.robitics</code> package defines a scheme to allow
for manipulating data
* channels. A channel is uniquely idetified in the DataTurbine with a form:
* <p>
* <i>&ltIP_Address&gt:&ltPort&gt/&ltdatapath&gt/&ltchannel_name&gt</i>
* <p>
* The <i>IP Address</i> is a standard address and the <i>port</i> used is 3333,
the default. The <i>datapath</i> is used
* to specify the source of where data is coming from and in the case of the
operator, uses its identification as the
* datapath when it puts data on the DataTurbine. When the operator looks for
data, it searches channels with the robot
* name in the datapath.
* <p>
* Communication between the robot and operator occur on 4 functional modes:
Control, Sensor, ACK and ERROR.
* Data flows in two directions, robot-to-operator and operator-to-robot. A
format is added to channel name to reflect the
* channel functions, of the form: <i>&ltfunction&gt:&ltname&gt</i> where
function is 'Control,' 'Sensor,' 'ACK,'
* or 'ERROR.' The <i>name</i> should reflects the application of the channel.
*
* @author Mike Rasay
* @author Ognjen Petrovic
* @author SCREEM (Santa Clara University)
**/
public class DummyRobot
	implements SerialPortDataListener
{

	/**
	* Name of the robot; used to establish the channel datapaths.
	**/
	private String			identification;
	/**
	* Name of the operator channel datapath that the robot looks for control
data on.
	**/
	private String			operatorIdentification;
	/**
	* IP address of the DataTurbine that the robot and the operator
communicate on.
	**/
	private String			ipAddress;
	/**
	* State of the connection to the DataTurbine: <code>false</code>-Not
Connected;
	* <code>true</code>-Connected.
	**/
```

```
	private boolean			connected = false;

	/**
	* Links the serial port name (for example "COM1"; "COM#") with the
<code>SerialPortConnection</code>.
	* Key: (String) serial port name; Value: (SerialPortConnection)
	**/
	private Hashtable		serialPorts = new Hashtable();
	/**
	* Links the sink channel with the serial port that passes control data to
the robot.
	* Key: (String) channel name; Value: (String) serial port name
	**/
	private Hashtable		sinkHash = new Hashtable();
	/**
	* Link the robot data (sensor or motor state data) on the serial port(s)
with the channel(s) that communicate
	* to the operator.
	* Key: (String) serial port name; Value: (String) channel name
	**/
	private Hashtable		sourceHash = new Hashtable();

	/**
	* List of <code>DataTurbineListener</code>s
	**/
	private Vector			dataTurbineListeners = new Vector();
	/**
	*List of <code>ErrorListener</code>s.
	**/
	private Vector			errorListeners = new Vector();
	/**
	* List of <code>SerialPortConnectionListener</code>s.
	**/
	private Vector			serialPortConnectionListeners = new Vector();

	//CONSTRUCTORS
	/**
	* Class constructor
	**/
	public DummyRobot() {}

	/*
	 public Robot(String identification, String ipAddress, Map setupMap,
Hashtable serialPorts) {
		this.identification = identification.toUpperCase();
		this.ipAddress = ipAddress;
		this.setupMap = setupMap;
		this.serialPorts = serialPorts;
	}

	//PUBLIC METHODS

	//EVENT LISTENER ADDITION-REMOVAL
	/**
	* Add a <code>DataTurbineListener</code> to the listener list.
	* @param listener <code>DataTurbineListener</code> to be added to the
list
	* @see screem.netrol.communication.event.DataTurbineListener
	**/
	public synchronized void addDataTurbineListener(DataTurbineListener
listener) {
		if(dataTurbineListeners.contains(listener)) { return; }
		dataTurbineListeners.addElement(listener);
```

```
	}

	/**
	* Add a <code>ErrorListener</code> to the listener list.
	* @param listener <code>ErrorListener</code> to be added to the list.
	* @see screem.netrol.communication.event.ErrorListener
	**/
	public synchronized void addErrorListener(ErrorListener listener) {
		if(errorListeners.contains(listener)) { return; }
		errorListeners.addElement(listener);
	}

	/**
	* Add a <code>SerialPortConnectionListener</code> to the listener list.
	* @param listener <code>SerialPortConnectionListner</code> to be added to
the list.
	* @see screem.netrol.serialport.SerialPortConnection
	**/
	public synchronized void
addSerialPortConnectionListener(SerialPortConnectionListener listener) {
		if(serialPortConnectionListeners.contains(listener)) { return; }
		serialPortConnectionListeners.addElement(listener);
	}

	/**
	* Remove a <code>DataTurbineListener</code> from the listener list.
	* @param listener <code>DataTurbineListener</code> to be removed from the
list.
	**/
	public synchronized void removeDataTurbineListener(DataTurbineListener
listener) {
		dataTurbineListeners.removeElement(listener);
	}

	/**
	* Remove a <code>ErrorListener</code> from the listener list.
	* @param listener <code>ErrorListener</code> to be removed from the list.
	**/
	public synchronized void removeErrorListener(ErrorListener listener) {
		errorListeners.removeElement(listener);
	}

	/**
	* Remove a <code>SerialPortConnectionListener</code> from the listener
list.
	* @param listener <code>SerialPortConnection</code> to be removed from
the list.
	**/
	public synchronized void
removeSerialPortConnectionListener(SerialPortConnectionListener listener) {
		serialPortConnectionListeners.removeElement(listener);
	}

	/**
	* Adds a control channel to the source and ACK channel to the sink then
synchs the serial port to the
	* channels.
	* @param channelName application of the channel for example motor
control; arm control
	* @param portName name of the port to synch with the control data for
example "COM1"
	* @param access read/write access given to the serial port
	* @param baudRate baud rate configuration
```

```
	 * @param dataBits data bit configuration
	 * @param flowControl flow control configuration
	 * @param parity parity setting
	 * @param stopBits stop bit configuration
	 * @param bufferSize is the size of data chunks from COM port
	 * @param sleeptime is the waiting time for data on COM port
	 * @throws ChannelException if the channel name is already established or
if a connection has not been
	 *	made
	 * @throws SerialPortException if the port cannot be opened
	 **/
	public void addControlChannel(String channelName, String portName, String
access,
			String baudRate, String dataBits, String flowControl,
String parity,
			String stopBits, String bufferSize, String sleeptime)
				throws SerialPortException {
		try {
			//add channels to source (response map)
			//set this up after the map has been sent to the operator
			//ADD PORT DATA TO THE HASHTABLES

			String[] ports = getAvailablePorts();
			for (int i = 0; i < ports.length; i++) {
				System.out.println("Ports are " + ports[i]);
			}
			SerialPortConnection port = new
SerialPortConnection("Control:" + channelName,
				portName, access, baudRate, dataBits, flowControl,
parity, stopBits, bufferSize,sleeptime);
			port.open();
			port.addSerialPortListener(this);
			serialPorts.put(portName, port);
			sourceHash.put(port.getPortName(), "ACK:" + channelName);
			sinkHash.put("Control:" + channelName, port.getPortName());
		}
		catch(SerialPortException ex) {
			throw new SerialPortException(ex.getMessage() + ":
\n\tSerial Port Error");
		}
	}

	/**
	 * Adds an 'ACK' (output) channel to the source and the corresponding
'Control' (input) channel to the sink
	 * and maps the I/O to a specific <code>SerialPortConnection</code>.
	 * @param channelName name of the application associated with
communication channel. The
	 *	function prefix is added to this name.
	 * @param port the SerialPortConnection associated with the communication
channels
	 * @throws SerialPortException if the serial port cannot be opened
	 * @throws ChannelException if a connection is not established or if a
duplicate channel name is given
	 * @see #sink
	 * @see #source
	 * @see screem.netrol.serialport.SerialPortConnection
	 **/
	public void addControlChannel(String channelName, SerialPortConnection
port)
			throws SerialPortException {
		try {
			//add channels to source (response map)
```

```
                        //set this up after the map has been sent to the operator
                        //ADD PORT DATA TO THE HASHTABLES
                        port.open();
                        port.addSerialPortListener(this);
                        serialPorts.put(port.getPortName(), port);
                        sourceHash.put(port.getPortName(), "ACK:" + channelName);
                        sinkHash.put("Control:" + channelName, port.getPortName());
                }
                catch(SerialPortException ex) {
                        throw new SerialPortException(ex.getMessage());
                }

        }

        //

        /**
        * Adds a sensor channel to the source then synchs the serial port to the
channel.
        * @param channelName application of the channel for example motor
control; arm control
        * @param portName name of the port to synch with the control data for
example "COM1"
        * @param access read/write access given to the serial port
        * @param baudRate baud rate configuration
        * @param dataBits data bit configuration
        * @param flowControl flow control configuration
        * @param parity parity setting
        * @param stopBits stop bit configuration
        * @param bufferSize is the size of data chunks from COM port
        * @param sleeptime is the waiting time for data on COM port
        * @throws ChannelException if the channel name is already established or
if a connection has not been
        *      made
        * @throws SerialPortException if the port cannot be opened
        **/
        public void addSensorChannel(String channelName, String portName, String
access, String baudRate,
                        String dataBits, String flowControl, String parity, String
stopBits,
                        String bufferSize,String sleeptime)
                                throws SerialPortException {
                try {

                        //OPEN THE PORT AND ADD INFO TO THE HASHTABLES
                        SerialPortConnection port = new
SerialPortConnection(channelName, portName,
                                        access, baudRate, dataBits, flowControl,
parity, stopBits,bufferSize,sleeptime);
                        port.open();
                        port.addSerialPortListener(this);
                        serialPorts.put(port.getPortName(), port);
                        sourceHash.put(port.getPortName(), "Sensor:" +
channelName);
                }
                catch(SerialPortException ex) {
                        throw new SerialPortException(ex.getMessage());
                }

        }

        /**
```

```
	* Adds a 'Sensor' (output) channel to the source and maps it to the
serial port that receives data from the
	* robot.
	* @param channelName name of the application associated with the
communication channel. The
	*	function prefix is added to this name.
	* @param port SerialPortConnection interface that maintains streams of
sensor data
	* @throws SerialPortException if the serial port cannot be opened
	* @throws ChannelException if the connection to the DataTurbine is not
established or if a duplicate
	*	channel name is given.
	**/
	public void addSensorChannel(String channelName, SerialPortConnection
port) throws
			SerialPortException {
		try {
			//OPEN THE PORT AND ADD INFO TO THE HASHTABLES
			port.open();
			port.addSerialPortListener(this);
			serialPorts.put(port.getPortName(), port);
			sourceHash.put(port.getPortName(), "Sensor:" +
channelName);
		}
		catch(SerialPortException ex) {
			throw new SerialPortException(ex.getMessage());
		}
	}

	/**
	* Sends bytes of data to the operator
	* @param channelName The name of the channel to send the data on. The
name must follow the form:
	*	<i>&ltfunction&gt:&ltname&gt</i>.
	* @param data byte array which is sent to the operator
	**/

	/**
	* Sets the sink map so that all the appropriate channels are listened on
to receive control and error data from
	* the operator.
	* @see #connect()
	**/
	//ACCESSOR MEHTODS
	/**
	* Returns the robot's identification
	* @return if the identification has not been set, the value will be
<code>null</code> otherwise,
	*	it will be the set identification.
	**/
	public String getIdentification() { return identification; }

	/**
	* Returns the operator's identification; the datapath searched for
incoming data.
	* @return if the operator's identification is not set, the value is
<code>null</code> otherwise, it
	*	will be the set identification.
	**/
	public String getOperatorIdentification() { return
operatorIdentification; }

	/**
```

```
	* Returns the IP address of the DataTurbine
	* @return if the IP address is not set, the value is <code>null</code> ;
if the address is set but the
	*	connection is not established, the value is the set address.
	**/
	public String getIPAddress() { return ipAddress; }

	/**
	* Returns a hash of control channel-serial port pairings.
	* @return the hash reflects the channels and the the serial ports setup
for control on the robot side.
	* @see #sinkHash
	**/
	public Hashtable getControlChannelPortPairs() {
		return sinkHash;
	}

	/**
	* Returns the state of the connection to the DataTurbine
	* @return <code>false</code>-not connected; <code>true</code>-is
connected
	**/
	public boolean isConnected() { return connected; }

	/**
	* Sets the identification of the robot.
	* @param identification name of the robot; the datapath that the operator
will search for data.
	*	The name is set to upper case to eliminate case sensitivity and
avoid the possibility of
	*	channels being lost due to the case of the datapath.
	**/
	public void setIdentification(String identification) {
this.identification = identification.toUpperCase(); }

	/**
	* Sets the identification of the operator in control of the robot; the
datapath searched for incoming
	* control data.
	* @param opeartorIdentification name of the operator
	**/
	public void setOperatorIdentification(String operatorIdentification) {
		this.operatorIdentification =
operatorIdentification.toUpperCase();
	}

	/**
	* Sets the IP address of the DataTurbine to connect to.
	* @param ipAddress The IP addres of the DataTurbine; can be "localhost".
	**/
	public void setIPAddress(String ipAddress) { this.ipAddress = ipAddress;
}

	//SERIAL PORT METHODS
	/**
	* Returns a list of all the ports available on the machine
	* @return the list of channels is a complete list of all the serial ports
on the machine, even the ones that
	*	are in use by another application.
	**/
	public String[] getAvailablePorts() {
		Vector list = new Vector();
		Enumeration portList;
```

```
		portList = CommPortIdentifier.getPortIdentifiers();
		while(portList.hasMoreElements()) {
			CommPortIdentifier portId =
(CommPortIdentifier)portList.nextElement();
			if(portId.getPortType() == CommPortIdentifier.PORT_SERIAL)
{
				list.addElement(portId.getName());
			}
		}
		//toArray is a method provided by Java2
		//return list.toArray();

		String[]portArray = new String[list.size()];
		for(int i=0; i<list.size(); i++) {
			portArray[i] = (String)list.elementAt(i);
		}
		return portArray;
	}

	/**
	* Returns a hash of serial ports being used by the robot.
	* @return the hash of serial ports in use.
	**/
	public Hashtable getPortUsage() { return serialPorts; }

	/**
	* Closes a serial port
	* @param portName name of the port to be closed (for example "COM1")
	**/
	public void closeSerialPort(String portName) {
		if(serialPorts.contains(portName)) {
			SerialPortConnection port =
(SerialPortConnection)serialPorts.get(portName);
			String appName = port.getAppName();
			port.removeSerialPortListener(this);
			port.close();
			serialPorts.remove(portName);
		}
	}

	/**
	* Sends data to the robot
	* @param portName the name of the port to put data on (for example
"COM1")
	* @param data byte arrray to be sent to the robot hardware via the serial
port.
	**/
	public void putDataOnPort(String portName, byte[] data) {
		SerialPortConnection port =
(SerialPortConnection)serialPorts.get(portName);
		try {
			port.send(data);
		} catch(SerialPortException e) {
			fireErrorEvent(e.getMessage());
		}
	}

	/**
	* Notifies the error listeners of an error that will either occur on the
DataTurbine or the serial port(s).
	* @param msg the error message
	* @see #errorListeners
	* @see screem.netrol.communication.event.ErrorEvent
```

```
	* @see screem.netrol.communication.event.ErrorListener
	**/
	protected void fireErrorEvent(String msg) {
		Vector listenerList;
		synchronized(this) {
			listenerList = (Vector)errorListeners.clone();
		}
		if(listenerList.size() == 0) { return; }
		ErrorEvent event = new ErrorEvent(this, msg);
		for(int i=0; i<listenerList.size(); i++) {
			ErrorListener listener =
				(ErrorListener)listenerList.elementAt(i);
			listener.errorOccurred(event);
		}
	}

	/* (non-Javadoc)
	 * @see
screem.netrol.serialport.event.SerialPortDataListener#dataFromRobotEvent(screem
.netrol.serialport.event.SerialPortDataEvent)
	 */
	public void dataFromRobotEvent(SerialPortDataEvent e) {

		byte[] buffer = e.getBuffer();
		System.out.println("Got data at port at " +
System.currentTimeMillis());
		//putDataOnPort("COM1", buffer);
		System.out.println(new String(buffer));
	}

	/* (non-Javadoc)
	 * @see
screem.netrol.serialport.event.SerialPortDataListener#dataToRobotEvent(screem.n
etrol.serialport.event.SerialPortDataEvent)
	 */
	public void dataToRobotEvent(SerialPortDataEvent arg0) {
		// TODO Auto-generated method stub

	}

	public StringBuffer sbDataRead = new StringBuffer();

	public static void main(String[] args) {
		DummyRobot dummy = new DummyRobot();
		try {
			dummy.addControlChannel("NetrolWeb", "COM1", "READ_WRITE",
"9600", SerialPort.DATABITS_8 + "",SerialPort.FLOWCONTROL_NONE + "",
SerialPort.PARITY_ODD + "", SerialPort.STOPBITS_1 + "", "256", "5");
		} catch (SerialPortException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
	}
}



import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
import java.net.*;
import java.io.*;
```

```
/**
 *
 * @author nmehdizadeh
 * This listens for certain keystrokes or joystick movements, highlights the
 * corresponding arrow, and submits the fact that that arrow was highlighted to
 * an online script via HTTP.
 */

public class KeyboardListener extends Applet {
	// These are the four arrow keys. This should not be changed.
	public static final int UP = Event.UP, RIGHT = Event.RIGHT,
			DOWN = Event.DOWN, LEFT = Event.LEFT;

	// These tell what keys will activate what arrows. Each variable
corresponds
	// to one arrow, and one direction. It is an array of keys which will
	// activate that arrow.
	public static final int[] NORTH_KEYS = { UP, 'i' },
			NORTHEAST_KEYS = { 'o' }, EAST_KEYS = { RIGHT, 'l' },
			SOUTHEAST_KEYS = { ',' }, SOUTH_KEYS = { DOWN, 'm' },
			SOUTHWEST_KEYS = { 'n' }, WEST_KEYS = { LEFT, 'j' },
			NORTHWEST_KEYS = { 'u' };

	// These are the values that correspond to a given direction. When the
	// given direction is selected (by pressing the key or moving the
joystick),
	// the corresponding value (defined here) will be sent to the SUBMIT_URL.
	// These are the default values, they may be changed through parameters
of
	// the same name (for example, if the parameter NORTH_VAL exists, then
	// that value will be changed to the parameter's value. The values should
	// all be integers.
	private int NORTH_VAL = 1, NORTHEAST_VAL = 2, EAST_VAL = 3,
			SOUTHEAST_VAL = 4, SOUTH_VAL = 5, SOUTHWEST_VAL = 6,
WEST_VAL = 7,
			NORTHWEST_VAL = 8;

	// This is the host that serves the script, and the name of the script
file
	// (relative to the web base). For example, if you could access the
script
	// by going to http://www.example.com/asub/script.php, then host
	// should be "www.example.com" and file should be "/asub/script.php"
	// These are default values, and should be changed (presuming it
	// necessary) through parameters of the same name. For example, you
	// will probably have a parameter SCRIPT_FILE which defines where the
file
	// is on the machine.
	private String SCRIPT_HOST = "localhost",
			SCRIPT_FILE = "/NetrolNew/applet.jsp";

	// These two arrays tell the x and y position (upper-left hand
coordinate)
	// of the images. The indexes correspond to the direction according the
	// the value [DIRECTION]_INDEX (where DIRECTION is substituted with the
	// given direction name)
	public static final int[] IMAGE_X_POS = { 60, 100, 100, 100, 60, 20, 20,
20 },
			IMAGE_Y_POS = { 20, 20, 60, 100, 100, 100, 60, 20 };

	// These are the indexes in the arrow arrays of the directions, and in
the
	// coordinate arrays.
```

```
	public static final int NORTH_INDEX = 0, NORTHEAST_INDEX = 1,
			EAST_INDEX = 2, SOUTHEAST_INDEX = 3, SOUTH_INDEX = 4,
			SOUTHWEST_INDEX = 5, WEST_INDEX = 6, NORTHWEST_INDEX = 7;

	// The HTTP port on the host machine
	public static final int HTTP_PORT = 8080;

	// The number of arrows that there are
	public static final int NUM_ARROWS = 8;

	// The arrow images - one array of the images when they are being
	// highlighted (selected), and one when they are in their normal state.
	private Image[] selectedArrows;

	private Image[] unselectedArrows;

	// These tell whether an arrow is selected or unselected.
	private boolean[] selected;

	// The url object corresponding to the URL that the images are kept at.
	// Within this directory, images should be named 0Selected.jpg and
	// 0Unselected.jpg for the north arrows, 1Selected.jpg and
1Unselected.jpg
	// for the northeast arrows, etc all the way through 7Selected.jpg and
	// 7Unselected.jpg for the northwest arrows.
	private URL imageURL;

	/**
	 * Run when the applet begins. Loads the graphics.
	 */
	public void init() {
		// Load parameters if they exist.
		String v = getParameter("NORTH_VAL");
		if (v != null)
			NORTH_VAL = Integer.parseInt(v);

		v = getParameter("NORTHEAST_VAL");
		if (v != null)
			NORTHEAST_VAL = Integer.parseInt(v);

		v = getParameter("EAST_VAL");
		if (v != null)
			EAST_VAL = Integer.parseInt(v);

		v = getParameter("SOUTHEAST_VAL");
		if (v != null)
			SOUTHEAST_VAL = Integer.parseInt(v);

		v = getParameter("SOUTH_VAL");
		if (v != null)
			SOUTH_VAL = Integer.parseInt(v);

		v = getParameter("SOUTHWEST_VAL");
		if (v != null)
			SOUTHWEST_VAL = Integer.parseInt(v);

		v = getParameter("WEST_VAL");
		if (v != null)
			WEST_VAL = Integer.parseInt(v);

		v = getParameter("NORTHWEST_VAL");
		if (v != null)
			NORTHWEST_VAL = Integer.parseInt(v);
```

```
		v = getParameter("SCRIPT_HOST");
		if (v != null)
			SCRIPT_HOST = v;

		v = getParameter("SCRIPT_FILE");
		if (v != null)
			SCRIPT_FILE = v;

		// All arrows are initially unselected
		selected = new boolean[NUM_ARROWS];
		for (int i = 0; i < selected.length; i++) {
			selected[i] = false;
		}

		// initialize the URL
		try {
			imageURL = new URL("http://" + SCRIPT_HOST +
":8080/NetrolNew/images/");
		} catch (MalformedURLException e) {
			// TODO Auto-generated catch block
			// e.printStackTrace();
		}

		// Initialize the image arrays
		selectedArrows = new Image[NUM_ARROWS];
		unselectedArrows = new Image[NUM_ARROWS];

		// Get all of the images
		for (int i = 0; i < NUM_ARROWS; i++) {
			// Get the selected and unselected images for this arrow.
			selectedArrows[i] = getImage(imageURL, i + "Selected.jpg");
			unselectedArrows[i] = getImage(imageURL, i +
"Unselected.jpg");
		}
	}

	/**
	 * Called when the keyboard key is pressed or joystick moved. Highlights
the
	 * corresponding arrow, and sends the corresponding value to the online
	 * script.
	 *
	 * @param index
	 *            the index in the arrow arrays and selected boolean array
	 *            corresponding to the arrow/direction that was selected
	 * @param val
	 *            the value to send to the online script
	 */
	public void select(int index, int val) {
		// Select the arrow
		selected[index] = true;

		// Repaint with the arrow selected
		repaint();

		// Send the HTTP request, entering the val in the query string.
		Socket s = null;
		try {
			URL u = getDocumentBase();

			// s = new Socket( SCRIPT_HOST, HTTP_PORT );
```

```
                    /*
                     * OutputStream out = s.getOutputStream(); PrintWriter
writer = new
                     * PrintWriter( out, false ); writer.print("GET " +
SCRIPT_FILE +
                     * "?action=control&control="+ val + "\r\n");
                     */

                    s = new Socket(SCRIPT_HOST, 8080);
                    OutputStream out = s.getOutputStream();
                    PrintWriter writer = new PrintWriter( out, false );
                    writer.print("GET " + SCRIPT_FILE +
"?action=control&control="+ val + "\r\n");

/*                  u = new URL("http://localhost:8080/NetrolNew/applet.jsp"
                                + "?action=control&control=" + val);
                    HttpURLConnection huc = (HttpURLConnection)
u.openConnection();
                    huc.setRequestMethod("GET");
                    huc.connect();
                    OutputStream os = huc.getOutputStream();
                    int code = huc.getResponseCode();

                    InputStream  rawInStream = huc.getInputStream();

            // get response
            BufferedReader rdr = new BufferedReader(new
InputStreamReader(rawInStream));
            String line;

            while ((line = rdr.readLine()) != null) {}
                    huc.disconnect();
*/
                    // getAppletContext().showDocument(u);
                    writer.flush();

                    s.close();
             }
             /*
              * catch( UnknownHostException e ) { System.out.println("Can't
connect
              * to hose machine: Invalid host."); System.out.println(
e.getMessage() ); }
              */catch (IOException e) {
             }

      }

      /**
       * This is called when a key is released. If it was a key which
highlighted
       * a button, this method will dehighlight it.
       *
       * @param keyEvent
       *             the event which occurred
       * @param key
       *             the key which was released
       */
      public boolean keyUp(Event keyEvent, int key) {
            // Look at each direction. If the key was in that direction,
dehighlight
            // the corresponding arrow.
            if (contains(NORTH_KEYS, key))
```

```
                selected[NORTH_INDEX] = false;

            else if (contains(NORTHEAST_KEYS, key))
                selected[NORTHEAST_INDEX] = false;

            else if (contains(EAST_KEYS, key))
                selected[EAST_INDEX] = false;

            else if (contains(SOUTHEAST_KEYS, key))
                selected[SOUTHEAST_INDEX] = false;

            else if (contains(SOUTH_KEYS, key))
                selected[SOUTH_INDEX] = false;

            else if (contains(SOUTHWEST_KEYS, key))
                selected[SOUTHWEST_INDEX] = false;

            else if (contains(WEST_KEYS, key))
                selected[WEST_INDEX] = false;

            else if (contains(NORTHWEST_KEYS, key))
                selected[NORTHWEST_INDEX] = false;

            else
                return true;

            // If we got here, then a key was dehighlighted. Update the
picture.
            repaint();

            return true;
      }

      /**
       * This is called to paint the applet window.
       *
       * @param g
       *             the graphics object to draw to
       */
      public void paint(Graphics g) {
            for (int i = 0; i < NUM_ARROWS; i++) {
                g.drawImage(
                              (selected[i] ? selectedArrows[i] :
unselectedArrows[i]),
                              IMAGE_X_POS[i], IMAGE_Y_POS[i], this);
            }
      }

      /**
       * This is called when a key is pressed. It checks to see if the key
       * corresponds to a direction, and highlights the arrow and sends the
value
       * to the script if it does.
       *
       * @param keyEvent
       *             the event which occurred
       * @param key
       *             the key that was pressed
       */
      public boolean keyDown(Event keyEvent, int key) {
            // Look at each direction. If the value was in that direction,
            // activate and send it.
            if (contains(NORTH_KEYS, key))
```

```
                        select(NORTH_INDEX, NORTH_VAL);

                else if (contains(NORTHEAST_KEYS, key))
                        select(NORTHEAST_INDEX, NORTHEAST_VAL);

                else if (contains(EAST_KEYS, key))
                        select(EAST_INDEX, EAST_VAL);

                else if (contains(SOUTHEAST_KEYS, key))
                        select(SOUTHEAST_INDEX, SOUTHEAST_VAL);

                else if (contains(SOUTH_KEYS, key))
                        select(SOUTH_INDEX, SOUTH_VAL);

                else if (contains(SOUTHWEST_KEYS, key))
                        select(SOUTHWEST_INDEX, SOUTHWEST_VAL);

                else if (contains(WEST_KEYS, key))
                        select(WEST_INDEX, WEST_VAL);

                else if (contains(NORTHWEST_KEYS, key))
                        select(NORTHWEST_INDEX, NORTHWEST_VAL);

                return true;
        }

        /**
         * Determines if 'val' is in the array 'array'.
         *
         * @param array
         *             an array of integers. They do not have to be sorted.
         * @param val
         *             the val to look for in array
         * @return true if val is in array, false if it is not
         */
        public boolean contains(int[] array, int val) {
                // If the array is null, then val is obviously not in it.
                if (array == null)
                        return false;

                // go through the array, and look at each element
                for (int i = 0; i < array.length; i++) {
                        // If this element is the value we are looking for, then
the val
                        // is in the array, so return true.
                        if (array[i] == val)
                                return true;
                }

                // None of the elements were the value we were looking for,
                // so return false.
                return false;
        }
}



/*
 * Created on Feb 14, 2005
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
```

```
package com.netrol.Data;

/**
 * @author nmehdizadeh
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
public class ManualCommand extends SingleValueCommnicationPacket {

	public ManualCommand()
	{
		setType('M');
	}

}



/*
 * Created on Feb 6, 2005
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
package com.netrol;

import java.util.Vector;

/**
 * @author nmehdizadeh
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
public class MessageCache {
	private static Vector dataList = new Vector();
	private static int counter = 0;

	public static Messages getNextDataField()
	{
		Messages value = null;
		synchronized (dataList) {
			if (dataList.size() >= counter)
			{
				value = (Messages)dataList.get(counter);
				counter +=1;
			}
		}
		return null;

	}

	public static Vector getAllRows()
	{
		// to load that class
		boolean val = DataReader.runFlag;

		Vector cloneVector = (Vector)dataList.clone();
		return cloneVector;

	}
	public static void addDataFields(Messages newDataFeilds)
```

```
	{
		synchronized (dataList) {
			dataList.add(newDataFeilds);
		}

	}
}



/*
 * Created on Feb 6, 2005
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
package com.netrol;

import java.util.StringTokenizer;

/**
 * @author nmehdizadeh
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
public class Messages {

	  private String logitude;
	  private String latitude;
	  private String depth;
	  private String buoyency;
	  private String delimiter = " ";

public void setRawData(String rawData)
{
	 StringTokenizer stTok = new StringTokenizer(rawData, delimiter);
	 setLogitude(stTok.nextToken());
	 setLatitude(stTok.nextToken());
	 setDepth(stTok.nextToken());
	 setBuoyency(stTok.nextToken());
}

public String getRawData()
{
	return getLogitude() + " " + getLatitude() + " " + getDepth() + " " +
getBuoyency();
}

/**
 * @return Returns the buoyency.
 */
public String getBuoyency() {
	return buoyency;
}
/**
 * @param buoyency The buoyency to set.
 */
public void setBuoyency(String buoyency) {
	this.buoyency = buoyency;
}
/**
 * @return Returns the depth.
```

```
 */
public String getDepth() {
	return depth;
}
/**
 * @param depth The depth to set.
 */
public void setDepth(String depth) {
	this.depth = depth;
}
/**
 * @return Returns the latitude.
 */
public String getLatitude() {
	return latitude;
}
/**
 * @param latitude The latitude to set.
 */
public void setLatitude(String latitude) {
	this.latitude = latitude;
}
/**
 * @return Returns the logitude.
 */
public String getLogitude() {
	return logitude;
}
/**
 * @param logitude The logitude to set.
 */
public void setLogitude(String logitude) {
	this.logitude = logitude;
}
}



/*
 * Created on Feb 14, 2005
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
package com.netrol.Data;

import java.util.ArrayList;

/**
 * @author nmehdizadeh
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
public class MultiValueCommnicationPacket {

	public int size()
	{
		int retValue = 0;
		if (dataList != null)
		{
			retValue = dataList.size();
		}
```

```
		return retValue;
	}

	public void addData(int targetLongitude, int targetLatitude)
	{

		SingleAutoCommand newCommand = new SingleAutoCommand();
		newCommand.setLatitude(targetLatitude);
		newCommand.setLongitude(targetLongitude);
		dataList.add(newCommand);
	}

	public ArrayList getCommands() throws Exception
	{
		return dataList;
	}

	public SingleAutoCommand getCommandAt(int index)
	{

		return (SingleAutoCommand)dataList.get(index);
	}

	public String getStringCommandAt(int index)
	{
		SingleAutoCommand command = getCommandAt(index);
		String rawCommand = "$" + getType() + "," + dataList.size() + ","
+ index + "," + command.getLongitude() + "," + command.getLatitude();
		String stringCommand = rawCommand + "*" + getCheckSum(rawCommand);

		return stringCommand;
	}

	private String getCheckSum(String command)
	{
		int returnValue = 0;
		if (command.length() >0)
		{
			returnValue = command.charAt(0);
		}
		for (int i = 1; i < command.length(); i++)
		{
			returnValue = returnValue ^ command.charAt(i);
		}
		// convert to hex
		return (Integer.toString(returnValue, 16)).toUpperCase();
	}

	public void setType(char type)
	{
		commandType = type;;
	}

	public char getType()
	{
		return commandType;
	}

	public ArrayList dataList = new ArrayList();


	public char commandType = 'A';
/*	private ArrayList longitude = new ArrayList();
```

```
      private ArrayList latitude = new ArrayList();
*/
}
/*
 * Created on Feb 20, 2005
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
package com.netrol;


import java.util.Enumeration;
import java.util.Hashtable;
import java.util.Vector;

import javax.comm.CommPortIdentifier;

import screem.netrol.communication.ChannelException;
import screem.netrol.communication.event.DataTurbineListener;
import screem.netrol.communication.event.ErrorEvent;
import screem.netrol.communication.event.ErrorListener;
import screem.netrol.communication.event.SerialPortConnectionListener;
import screem.netrol.serialport.SerialPortConnection;
import screem.netrol.serialport.SerialPortException;
import screem.netrol.serialport.event.SerialPortDataEvent;
import screem.netrol.serialport.event.SerialPortDataListener;

/**
* The <code>Robot</code> class manages the connection to the DataTurbine,
communication between the robot and
* the opearator and maintains the interface between the robot hardware and the
computer. The robot hardware and
* computer communicate on the computer's serial (COM) port(s). This
communication interface is implemented in the
* <code>SerialPortConnection</code> class, setting the communication parameters
and the I/O streams.
* <p>
* Following the <code>Channel</code> naming scheme defined by the DataTurbine
API, the
* <code>edu.scu.engr.screem.robitics</code> package defines a scheme to allow
for manipulating data
* channels. A channel is uniquely idetified in the DataTurbine with a form:
* <p>
* <i>&ltIP_Address&gt:&ltPort&gt/&ltdatapath&gt/&ltchannel_name&gt</i>
* <p>
* The <i>IP Address</i> is a standard address and the <i>port</i> used is 3333,
the default. The <i>datapath</i> is used
* to specify the source of where data is coming from and in the case of the
operator, uses its identification as the
* datapath when it puts data on the DataTurbine. When the operator looks for
data, it searches channels with the robot
* name in the datapath.
* <p>
* Communication between the robot and operator occur on 4 functional modes:
Control, Sensor, ACK and ERROR.
* Data flows in two directions, robot-to-operator and operator-to-robot. A
format is added to channel name to reflect the
* channel functions, of the form: <i>&ltfunction&gt:&ltname&gt</i> where
function is 'Control,' 'Sensor,' 'ACK,'
* or 'ERROR.' The <i>name</i> should reflects the application of the channel.
*
**/
```

```
public class Robot
	implements SerialPortDataListener
{

	/**
	* Name of the robot; used to establish the channel datapaths.
	**/
	private String		identification;
	/**
	* Name of the operator channel datapath that the robot looks for control
data on.
	**/
	private String		operatorIdentification;
	/**
	* IP address of the DataTurbine that the robot and the operator
communicate on.
	**/
	private String		ipAddress;
	/**
	* State of the connection to the DataTurbine: <code>false</code>-Not
Connected;
	* <code>true</code>-Connected.
	**/
	private boolean		connected = false;

	/**
	* Links the serial port name (for example "COM1"; "COM#") with the
<code>SerialPortConnection</code>.
	* Key: (String) serial port name; Value: (SerialPortConnection)
	**/
	private Hashtable	serialPorts = new Hashtable();
	/**
	* Links the sink channel with the serial port that passes control data to
the robot.
	* Key: (String) channel name; Value: (String) serial port name
	**/
	private Hashtable	sinkHash = new Hashtable();
	/**
	* Link the robot data (sensor or motor state data) on the serial port(s)
with the channel(s) that communicate
	* to the operator.
	* Key: (String) serial port name; Value: (String) channel name
	**/
	private Hashtable	sourceHash = new Hashtable();

	/**
	* List of <code>DataTurbineListener</code>s
	**/
	private Vector		dataTurbineListeners = new Vector();
	/**
	*List of <code>ErrorListener</code>s.
	**/
	private Vector		errorListeners = new Vector();
	/**
	* List of <code>SerialPortConnectionListener</code>s.
	**/
	private Vector		serialPortConnectionListeners = new Vector();

	//CONSTRUCTORS
	/**
	* Class constructor
	**/
	public Robot() {}
```

```
	/*
	 public Robot(String identification, String ipAddress, Map setupMap,
Hashtable serialPorts) {
		this.identification = identification.toUpperCase();
		this.ipAddress = ipAddress;
		this.setupMap = setupMap;
		this.serialPorts = serialPorts;
	}

	//PUBLIC METHODS

	//EVENT LISTENER ADDITION-REMOVAL
	/**
	* Add a <code>DataTurbineListener</code> to the listener list.
	* @param listener <code>DataTurbineListener</code> to be added to the
list
	* @see screem.netrol.communication.event.DataTurbineListener
	**/
	public synchronized void addDataTurbineListener(DataTurbineListener
listener) {
		if(dataTurbineListeners.contains(listener)) { return; }
		dataTurbineListeners.addElement(listener);
	}

	/**
	* Add a <code>ErrorListener</code> to the listener list.
	* @param listener <code>ErrorListener</code> to be added to the list.
	* @see screem.netrol.communication.event.ErrorListener
	**/
	public synchronized void addErrorListener(ErrorListener listener) {
		if(errorListeners.contains(listener)) { return; }
		errorListeners.addElement(listener);
	}

	/**
	* Add a <code>SerialPortConnectionListener</code> to the listener list.
	* @param listener <code>SerialPortConnectionListner</code> to be added to
the list.
	* @see screem.netrol.serialport.SerialPortConnection
	**/
	public synchronized void
addSerialPortConnectionListener(SerialPortConnectionListener listener) {
		if(serialPortConnectionListeners.contains(listener)) { return; }
		serialPortConnectionListeners.addElement(listener);
	}

	/**
	* Remove a <code>DataTurbineListener</code> from the listener list.
	* @param listener <code>DataTurbineListener</code> to be removed from the
list.
	**/
	public synchronized void removeDataTurbineListener(DataTurbineListener
listener) {
		dataTurbineListeners.removeElement(listener);
	}

	/**
	* Remove a <code>ErrorListener</code> from the listener list.
	* @param listener <code>ErrorListener</code> to be removed from the list.
	**/
	public synchronized void removeErrorListener(ErrorListener listener) {
		errorListeners.removeElement(listener);
```

```
	}

	/**
	* Remove a <code>SerialPortConnectionListener</code> from the listener
list.
	* @param listener <code>SerialPortConnection</code> to be removed from
the list.
	**/
	public synchronized void
removeSerialPortConnectionListener(SerialPortConnectionListener listener) {
		serialPortConnectionListeners.removeElement(listener);
	}

	/**
	* Adds a control channel to the source and ACK channel to the sink then
synchs the serial port to the
	* channels.
	* @param channelName application of the channel for example motor
control; arm control
	* @param portName name of the port to synch with the control data for
example "COM1"
	* @param access read/write access given to the serial port
	* @param baudRate baud rate configuration
	* @param dataBits data bit configuration
	* @param flowControl flow control configuration
	* @param parity parity setting
	* @param stopBits stop bit configuration
	* @param bufferSize is the size of data chunks from COM port
	* @param sleeptime is the waiting time for data on COM port
	* @throws ChannelException if the channel name is already established or
if a connection has not been
	*	made
	* @throws SerialPortException if the port cannot be opened
	**/
	public void addControlChannel(String channelName, String portName, String
access,
			String baudRate, String dataBits, String flowControl,
String parity,
			String stopBits, String bufferSize, String sleeptime)
				throws SerialPortException {
		try {
			//add channels to source (response map)
			//set this up after the map has been sent to the operator
			//ADD PORT DATA TO THE HASHTABLES
			SerialPortConnection port = new
SerialPortConnection("Control:" + channelName,
				portName, access, baudRate, dataBits, flowControl,
parity, stopBits, bufferSize,sleeptime);
			port.open();
			port.addSerialPortListener(this);
			serialPorts.put(portName, port);
			sourceHash.put(port.getPortName(), "ACK:" + channelName);
			sinkHash.put("Control:" + channelName, port.getPortName());
		}
		catch(SerialPortException ex) {
			throw new SerialPortException(ex.getMessage() + ":
\n\tSerial Port Error");
		}
	}

	/**
	* Adds an 'ACK' (output) channel to the source and the corresponding
'Control' (input) channel to the sink
```

```
	* and maps the I/O to a specific <code>SerialPortConnection</code>.
	* @param channelName name of the application associated with
communication channel. The
	*      function prefix is added to this name.
	* @param port the SerialPortConnection associated with the communication
channels
	* @throws SerialPortException if the serial port cannot be opened
	* @throws ChannelException if a connection is not established or if a
duplicate channel name is given
	* @see #sink
	* @see #source
	* @see screem.netrol.serialport.SerialPortConnection
	**/
	public void addControlChannel(String channelName, SerialPortConnection
port)
			throws SerialPortException {
		try {
			//add channels to source (response map)
			//set this up after the map has been sent to the operator
			//ADD PORT DATA TO THE HASHTABLES
			port.open();
			port.addSerialPortListener(this);
			serialPorts.put(port.getPortName(), port);
			sourceHash.put(port.getPortName(), "ACK:" + channelName);
			sinkHash.put("Control:" + channelName, port.getPortName());
		}
		catch(SerialPortException ex) {
			throw new SerialPortException(ex.getMessage());
		}

	}

	//

	/**
	* Adds a sensor channel to the source then synchs the serial port to the
channel.
	* @param channelName application of the channel for example motor
control; arm control
	* @param portName name of the port to synch with the control data for
example "COM1"
	* @param access read/write access given to the serial port
	* @param baudRate baud rate configuration
	* @param dataBits data bit configuration
	* @param flowControl flow control configuration
	* @param parity parity setting
	* @param stopBits stop bit configuration
	* @param bufferSize is the size of data chunks from COM port
	* @param sleeptime is the waiting time for data on COM port
	* @throws ChannelException if the channel name is already established or
if a connection has not been
	*      made
	* @throws SerialPortException if the port cannot be opened
	**/
	public void addSensorChannel(String channelName, String portName, String
access, String baudRate,
			String dataBits, String flowControl, String parity, String
stopBits,
			String bufferSize,String sleeptime)
				throws SerialPortException {
		try {

			//OPEN THE PORT AND ADD INFO TO THE HASHTABLES
```

```
                    SerialPortConnection port = new
SerialPortConnection(channelName, portName,
                                  access, baudRate, dataBits, flowControl,
parity, stopBits,bufferSize,sleeptime);
                    port.open();
                    port.addSerialPortListener(this);
                    serialPorts.put(port.getPortName(), port);
                    sourceHash.put(port.getPortName(), "Sensor:" +
channelName);
             }
             catch(SerialPortException ex) {
                    throw new SerialPortException(ex.getMessage());
             }

      }

      /**
      * Adds a 'Sensor' (output) channel to the source and maps it to the
serial port that receives data from the
      * robot.
      * @param channelName name of the application associated with the
communication channel. The
      *      function prefix is added to this name.
      * @param port SerialPortConnection interface that maintains streams of
sensor data
      * @throws SerialPortException if the serial port cannot be opened
      * @throws ChannelException if the connection to the DataTurbine is not
established or if a duplicate
      *      channel name is given.
      **/
      public void addSensorChannel(String channelName, SerialPortConnection
port) throws
                    SerialPortException {
             try {
                    //OPEN THE PORT AND ADD INFO TO THE HASHTABLES
                    port.open();
                    port.addSerialPortListener(this);
                    serialPorts.put(port.getPortName(), port);
                    sourceHash.put(port.getPortName(), "Sensor:" +
channelName);
             }
             catch(SerialPortException ex) {
                    throw new SerialPortException(ex.getMessage());
             }
      }

      /**
      * Sends bytes of data to the operator
      * @param channelName The name of the channel to send the data on. The
name must follow the form:
      *      <i>&ltfunction&gt:&ltname&gt</i>.
      * @param data byte array which is sent to the operator
      **/

      /**
      * Sets the sink map so that all the appropriate channels are listened on
to receive control and error data from
      * the operator.
      * @see #connect()
      **/
      //ACCESSOR MEHTODS
      /**
      * Returns the robot's identification
```

```
	* @return if the identification has not been set, the value will be
<code>null</code> otherwise,
	*	it will be the set identification.
	**/
	public String getIdentification() { return identification; }

	/**
	* Returns the operator's identification; the datapath searched for
incoming data.
	* @return if the operator's identification is not set, the value is
<code>null</code> otherwise, it
	*	will be the set identification.
	**/
	public String getOperatorIdentification() { return
operatorIdentification; }

	/**
	* Returns the IP address of the DataTurbine
	* @return if the IP address is not set, the value is <code>null</code> ;
if the address is set but the
	*	connection is not established, the value is the set address.
	**/
	public String getIPAddress() { return ipAddress; }

	/**
	* Returns a hash of control channel-serial port pairings.
	* @return the hash reflects the channels and the the serial ports setup
for control on the robot side.
	* @see #sinkHash
	**/
	public Hashtable getControlChannelPortPairs() {
		return sinkHash;
	}

	/**
	* Returns the state of the connection to the DataTurbine
	* @return <code>false</code>-not connected; <code>true</code>-is
connected
	**/
	public boolean isConnected() { return connected; }

	/**
	* Sets the identification of the robot.
	* @param identification name of the robot; the datapath that the operator
will search for data.
	*	The name is set to upper case to eliminate case sensitivity and
avoid the possibility of
	*	channels being lost due to the case of the datapath.
	**/
	public void setIdentification(String identification) {
this.identification = identification.toUpperCase(); }

	/**
	* Sets the identification of the operator in control of the robot; the
datapath searched for incoming
	* control data.
	* @param opeartorIdentification name of the operator
	**/
	public void setOperatorIdentification(String operatorIdentification) {
		this.operatorIdentification =
operatorIdentification.toUpperCase();
	}
```

```
	/**
	* Sets the IP address of the DataTurbine to connect to.
	* @param ipAddress The IP addres of the DataTurbine; can be "localhost".
	**/
	public void setIPAddress(String ipAddress) { this.ipAddress = ipAddress;
}

	//SERIAL PORT METHODS
	/**
	* Returns a list of all the ports available on the machine
	* @return the list of channels is a complete list of all the serial ports
on the machine, even the ones that
	*	are in use by another application.
	**/
	public String[] getAvailablePorts() {
		Vector list = new Vector();
		Enumeration portList;
		portList = CommPortIdentifier.getPortIdentifiers();
		while(portList.hasMoreElements()) {
			CommPortIdentifier portId =
(CommPortIdentifier)portList.nextElement();
			if(portId.getPortType() == CommPortIdentifier.PORT_SERIAL)
{
				list.addElement(portId.getName());
			}
		}
		//toArray is a method provided by Java2
		//return list.toArray();

		String[]portArray = new String[list.size()];
		for(int i=0; i<list.size(); i++) {
			portArray[i] = (String)list.elementAt(i);
		}
		return portArray;
	}

	/**
	* Returns a hash of serial ports being used by the robot.
	* @return the hash of serial ports in use.
	**/
	public Hashtable getPortUsage() { return serialPorts; }

	/**
	* Closes a serial port
	* @param portName name of the port to be closed (for example "COM1")
	**/
	public void closeSerialPort(String portName) {
		if(serialPorts.contains(portName)) {
			SerialPortConnection port =
(SerialPortConnection)serialPorts.get(portName);
			String appName = port.getAppName();
			port.removeSerialPortListener(this);
			port.close();
			serialPorts.remove(portName);
		}
	}

	/**
	* Sends data to the robot
	* @param portName the name of the port to put data on (for example
"COM1")
	* @param data byte arrray to be sent to the robot hardware via the serial
port.
```

```
	**/
	public void putDataOnPort(String portName, byte[] data) {
		SerialPortConnection port =
(SerialPortConnection)serialPorts.get(portName);
		try {
			port.send(data);
		} catch(SerialPortException e) {
			fireErrorEvent(e.getMessage());
		}
	}

	/**
	* Notifies the error listeners of an error that will either occur on the
DataTurbine or the serial port(s).
	* @param msg the error message
	* @see #errorListeners
	* @see screem.netrol.communication.event.ErrorEvent
	* @see screem.netrol.communication.event.ErrorListener
	**/
	protected void fireErrorEvent(String msg) {
		Vector listenerList;
		synchronized(this) {
			listenerList = (Vector)errorListeners.clone();
		}
		if(listenerList.size() == 0) { return; }
		ErrorEvent event = new ErrorEvent(this, msg);
		for(int i=0; i<listenerList.size(); i++) {
			ErrorListener listener =
				(ErrorListener)listenerList.elementAt(i);
			listener.errorOccurred(event);
		}
	}

	/* (non-Javadoc)
	 * @see
screem.netrol.serialport.event.SerialPortDataListener#dataFromRobotEvent(screem
.netrol.serialport.event.SerialPortDataEvent)
	 */
	public void dataFromRobotEvent(SerialPortDataEvent e) {

		byte[] buffer = e.getBuffer();
		DataPipe.storeDataFromRobotToDatabase(buffer);

	}

	/* (non-Javadoc)
	 * @see
screem.netrol.serialport.event.SerialPortDataListener#dataToRobotEvent(screem.n
etrol.serialport.event.SerialPortDataEvent)
	 */
	public void dataToRobotEvent(SerialPortDataEvent arg0) {
		// TODO Auto-generated method stub

	}

	public StringBuffer sbDataRead = new StringBuffer();

}



/*
 * Created on Feb 24, 2005
```

```
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
package com.netrol;

/*
 * @(#)SimpleWrite.java 1.12 98/06/25 SMI
 *
 * Copyright 2003 Sun Microsystems, Inc. All rights reserved. SUN
 * PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
 *
 * Sun grants you ("Licensee") a non-exclusive, royalty free, license to use,
 * modify and redistribute this software in source and binary code form,
 * provided that i) this copyright notice and license appear on all copies of
 * the software; and ii) Licensee does not utilize the software in a manner
 * which is disparaging to Sun.
 *
 * This software is provided "AS IS," without a warranty of any kind. ALL
 * EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY
 * IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR
 * NON-INFRINGEMENT, ARE HEREBY EXCLUDED. SUN AND ITS LICENSORS SHALL NOT BE
 * LIABLE FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING
 * OR DISTRIBUTING THE SOFTWARE OR ITS DERIVATIVES. IN NO EVENT WILL SUN OR ITS
 * LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR DIRECT,
 * INDIRECT, SPECIAL, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER
 * CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF THE USE OF
 * OR INABILITY TO USE SOFTWARE, EVEN IF SUN HAS BEEN ADVISED OF THE
POSSIBILITY
 * OF SUCH DAMAGES.
 *
 * This software is not designed or intended for use in on-line control of
 * aircraft, air traffic, aircraft navigation or aircraft communications; or in
 * the design, construction, operation or maintenance of any nuclear facility.
 * Licensee represents and warrants that it will not use or redistribute the
 * Software for such purposes.
 */
import java.io.*;
import java.util.*;

import javax.comm.*;

/**
 * Class declaration
 *
 *
 * @author
 * @version 1.10, 08/04/00
 */
public class RobotWritePort implements SerialPortEventListener {
       static Enumeration portList;

       static CommPortIdentifier portId;

       static SerialPort serialPort;

       static OutputStream outputStream;

       static InputStream inputStream;

       static boolean outputBufferEmptyFlag = false;

       static String defaultPort = "COM1";
```

```
	static boolean isPortOpen = false;

	static boolean isRead = true;

	/**
	 * Method declaration
	 *
	 *
	 * @param args
	 *
	 * @see
	 */
	public void writeData(String messageString) {
		boolean portFound = false;

		messageString = messageString + "\r\n";
		portList = CommPortIdentifier.getPortIdentifiers();

		while (portList.hasMoreElements()) {
			portId = (CommPortIdentifier) portList.nextElement();

			if (portId.getPortType() == CommPortIdentifier.PORT_SERIAL)
{

				if (portId.getName().equals(defaultPort)) {
					System.out.println("Found port " +
defaultPort);

					portFound = true;

					try {
						if (!isPortOpen) {
							serialPort = (SerialPort)
portId.open(
									"SimpleWrite",
2000);
							isPortOpen = true;


	serialPort.addEventListener(this);

	serialPort.notifyOnDataAvailable(true);
						}
					} catch (PortInUseException e) {
						System.out.println("Port in use.");

						continue;
					} catch (TooManyListenersException e) {
						// TODO Auto-generated catch block
						e.printStackTrace();
					}

					try {
						outputStream =
serialPort.getOutputStream();
					} catch (IOException e) {
					}

					try {
						serialPort.setSerialPortParams(9600,
								SerialPort.DATABITS_8,
SerialPort.STOPBITS_1,
```

```
                                                SerialPort.PARITY_NONE);
                                    } catch (UnsupportedCommOperationException e)
{
                                    }

                                    try {
                                          serialPort.notifyOnOutputEmpty(true);
                                    } catch (Exception e) {
                                          System.out.println("Error setting event
notification");
                                          System.out.println(e.toString());
                                          System.exit(-1);
                                    }

                                    System.out.println("Writing \"" +
messageString + "\" to "
                                                + serialPort.getName());

                                    try {
                                          System.out.println("Writing to port
started at " + System.currentTimeMillis());

      outputStream.write(messageString.getBytes());
                                          System.out.println("Writing to port
ended at " + System.currentTimeMillis());
                                    } catch (IOException e) {
                                    }

                                    /*
                                     * try { Thread.sleep(2000); // Be sure data
is xferred
                                     * before // closing } catch (Exception e) { }
                                     */

                              }
                        }
            }

            if (!portFound) {
                  System.out.println("port " + defaultPort + " not found.");
            }
      }

    public void serialEvent(SerialPortEvent event) {

            switch (event.getEventType()) {

            case SerialPortEvent.BI:

            case SerialPortEvent.OE:

            case SerialPortEvent.FE:

            case SerialPortEvent.PE:

            case SerialPortEvent.CD:

            case SerialPortEvent.CTS:

            case SerialPortEvent.DSR:

            case SerialPortEvent.RI:
```

```
            case SerialPortEvent.OUTPUT_BUFFER_EMPTY:
                break;

            case SerialPortEvent.DATA_AVAILABLE:
                byte[] readBuffer = new byte[20];

                try {
                   if (inputStream == null)
                   {
                          inputStream = serialPort.getInputStream();
                   }
                   while (inputStream.available() > 0) {
                       int numBytes = inputStream.read(readBuffer);
                   }

                   if(isRead)
                   {
                          System.out.println("Recived data back from robot at "
+ System.currentTimeMillis());
                   }
                   isRead = !isRead;
                   //System.out.print(new String(readBuffer));
                } catch (IOException e) {}

                break;
            }

    }


}


package com.netrol.dummyRobot;

/*
 * @(#)SimpleRead.java    1.12 98/06/25 SMI
 *
 * Copyright 2003 Sun Microsystems, Inc. All rights reserved.
 * SUN PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
 *
 * Sun grants you ("Licensee") a non-exclusive, royalty free, license
 * to use, modify and redistribute this software in source and binary
 * code form, provided that i) this copyright notice and license appear
 * on all copies of the software; and ii) Licensee does not utilize the
 * software in a manner which is disparaging to Sun.
 *
 * This software is provided "AS IS," without a warranty of any kind.
 * ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES,
 * INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE HEREBY EXCLUDED. SUN AND
 * ITS LICENSORS SHALL NOT BE LIABLE FOR ANY DAMAGES SUFFERED BY
 * LICENSEE AS A RESULT OF USING, MODIFYING OR DISTRIBUTING THE
 * SOFTWARE OR ITS DERIVATIVES. IN NO EVENT WILL SUN OR ITS LICENSORS
 * BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR DIRECT,
 * INDIRECT, SPECIAL, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES,
 * HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING
 * OUT OF THE USE OF OR INABILITY TO USE SOFTWARE, EVEN IF SUN HAS BEEN
 * ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.
 *
 * This software is not designed or intended for use in on-line control
 * of aircraft, air traffic, aircraft navigation or aircraft
 * communications; or in the design, construction, operation or
```

```
 * maintenance of any nuclear facility. Licensee represents and
 * warrants that it will not use or redistribute the Software for such
 * purposes.
 */
import java.io.*;
import java.util.*;
import javax.comm.*;

/**
 * Class declaration
 *
 *
 * @author
 * @version 1.8, 08/03/00
 */
public class SimpleRead implements Runnable, SerialPortEventListener {
    static CommPortIdentifier portId;
    static Enumeration          portList;
    InputStream                 inputStream;
    SerialPort                  serialPort;
    Thread                readThread;

    /**
     * Method declaration
     *
     *
     * @param args
     *
     * @see
     */
    public static void main(String[] args) {
    boolean               portFound = false;
    String                defaultPort = "COM1";

       if (args.length > 0) {
           defaultPort = args[0];
       }

       portList = CommPortIdentifier.getPortIdentifiers();

       while (portList.hasMoreElements()) {
           portId = (CommPortIdentifier) portList.nextElement();
           if (portId.getPortType() == CommPortIdentifier.PORT_SERIAL) {
              if (portId.getName().equals(defaultPort)) {
                  System.out.println("Found port: "+defaultPort);
                  portFound = true;
                  SimpleRead reader = new SimpleRead();
              }
           }
       }
       if (!portFound) {
           System.out.println("port " + defaultPort + " not found.");
       }

    }

    /**
     * Constructor declaration
     *
     *
     * @see
     */
    public SimpleRead() {
```

```
   try {
       serialPort = (SerialPort) portId.open("SimpleReadApp", 2000);
   } catch (PortInUseException e) {}

   try {
       inputStream = serialPort.getInputStream();
   } catch (IOException e) {}

   try {
       serialPort.addEventListener(this);
   } catch (TooManyListenersException e) {}

   serialPort.notifyOnDataAvailable(true);

   try {
       serialPort.setSerialPortParams(9600, SerialPort.DATABITS_8,
                                  SerialPort.STOPBITS_1,
                                  SerialPort.PARITY_NONE);
   } catch (UnsupportedCommOperationException e) {}

   readThread = new Thread(this);

   readThread.start();
}

/**
 * Method declaration
 *
 *
 * @see
 */
public void run() {
   try {
       Thread.sleep(20000);
   } catch (InterruptedException e) {}
}

/**
 * Method declaration
 *
 *
 * @param event
 *
 * @see
 */
public void serialEvent(SerialPortEvent event) {
   switch (event.getEventType()) {

   case SerialPortEvent.BI:

   case SerialPortEvent.OE:

   case SerialPortEvent.FE:

   case SerialPortEvent.PE:

   case SerialPortEvent.CD:

   case SerialPortEvent.CTS:

   case SerialPortEvent.DSR:

   case SerialPortEvent.RI:
```

```
       case SerialPortEvent.OUTPUT_BUFFER_EMPTY:
           break;

       case SerialPortEvent.DATA_AVAILABLE:
           byte[] readBuffer = new byte[20];

           try {
              while (inputStream.available() > 0) {
                  int numBytes = inputStream.read(readBuffer);
              }

              System.out.print(new String(readBuffer));
           } catch (IOException e) {}

           break;
       }
    }

}
/*
 * Created on Feb 24, 2005
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
package com.netrol.dummyRobot;

/*
 * @(#)SimpleWrite.java   1.12 98/06/25 SMI
 *
 * Copyright 2003 Sun Microsystems, Inc. All rights reserved.
 * SUN PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
 *
 * Sun grants you ("Licensee") a non-exclusive, royalty free, license
 * to use, modify and redistribute this software in source and binary
 * code form, provided that i) this copyright notice and license appear
 * on all copies of the software; and ii) Licensee does not utilize the
 * software in a manner which is disparaging to Sun.
 *
 * This software is provided "AS IS," without a warranty of any kind.
 * ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES,
 * INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE HEREBY EXCLUDED. SUN AND
 * ITS LICENSORS SHALL NOT BE LIABLE FOR ANY DAMAGES SUFFERED BY
 * LICENSEE AS A RESULT OF USING, MODIFYING OR DISTRIBUTING THE
 * SOFTWARE OR ITS DERIVATIVES. IN NO EVENT WILL SUN OR ITS LICENSORS
 * BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR DIRECT,
 * INDIRECT, SPECIAL, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES,
 * HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING
 * OUT OF THE USE OF OR INABILITY TO USE SOFTWARE, EVEN IF SUN HAS BEEN
 * ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.
 *
 * This software is not designed or intended for use in on-line control
 * of aircraft, air traffic, aircraft navigation or aircraft
 * communications; or in the design, construction, operation or
 * maintenance of any nuclear facility. Licensee represents and
 * warrants that it will not use or redistribute the Software for such
 * purposes.
 */
import java.io.*;
import java.util.*;
import javax.comm.*;
```

```
/**
 * Class declaration
 *
 *
 * @author
 * @version 1.10, 08/04/00
 */
public class SimpleWrite {
    static Enumeration          portList;
    static CommPortIdentifier portId;
    static String         messageString = "Hello, world!";
    static SerialPort           serialPort;
    static OutputStream       outputStream;
    static boolean        outputBufferEmptyFlag = false;
    /**
     * Method declaration
     *
     *
     * @param args
     *
     * @see
     */
    public static void main(String[] args) {
      boolean portFound = false;
      String  defaultPort = "COM1";

      if (args.length > 0) {
          defaultPort = args[0];
      }

      portList = CommPortIdentifier.getPortIdentifiers();

      while (portList.hasMoreElements()) {
          portId = (CommPortIdentifier) portList.nextElement();

          if (portId.getPortType() == CommPortIdentifier.PORT_SERIAL) {

             if (portId.getName().equals(defaultPort)) {
                 System.out.println("Found port " + defaultPort);

                 portFound = true;

                 try {
                    serialPort =
                        (SerialPort) portId.open("SimpleWrite", 2000);
                 } catch (PortInUseException e) {
                    System.out.println("Port in use.");

                    continue;
                 }

                 try {
                    outputStream = serialPort.getOutputStream();
                 } catch (IOException e) {}

                 try {
                    serialPort.setSerialPortParams(9600,
                                                   SerialPort.DATABITS_8,
                                                   SerialPort.STOPBITS_1,
                                                   SerialPort.PARITY_NONE);
                 } catch (UnsupportedCommOperationException e) {}
```

```
                try {
                   serialPort.notifyOnOutputEmpty(true);
                } catch (Exception e) {
                   System.out.println("Error setting event notification");
                   System.out.println(e.toString());
                   System.exit(-1);
                }


                System.out.println(
                   "Writing \""+messageString+"\" to "
                   +serialPort.getName());

                try {
                   outputStream.write(messageString.getBytes());
                } catch (IOException e) {}

                try {
                   Thread.sleep(2000);  // Be sure data is xferred before
closing
                } catch (Exception e) {}
                serialPort.close();
                System.exit(1);
             }
          }
       }

       if (!portFound) {
           System.out.println("port " + defaultPort + " not found.");
       }
    }


}



/*
 * Created on Feb 14, 2005
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
package com.netrol.Data;

/**
 * @author nmehdizadeh
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
public class SingleAutoCommand {

       /**
        * @return Returns the latitude.
        */
       public int getLatitude() {
              return latitude;
       }
       /**
        * @param latitude The latitude to set.
        */
```

```
	public void setLatitude(int latitude) {
		this.latitude = latitude;
	}
	/**
	 * @return Returns the longitude.
	 */
	public int getLongitude() {
		return longitude;
	}
	/**
	 * @param longitude The longitude to set.
	 */
	public void setLongitude(int longitude) {
		this.longitude = longitude;
	}

	int longitude;
	int latitude;

}

/*
 * Created on Feb 14, 2005
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
package com.netrol.Data;

import java.util.HashMap;

/**
 * @author nmehdizadeh
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
public abstract class SingleValueCommnicationPacket {
	public SingleValueCommnicationPacket()
	{

	}

	public String toString()
	{
		String rawCommand = "$" + getType() + "," +
getHorizontalDisplacement() + "," + getVerticalDisplacement();
		String command = rawCommand + "*" + getCheckSum(rawCommand);
		return command;
	}

	public void setData(int targetLongitude, int targetLatitude)
	{
		setHorizontalDisplacement(targetLatitude);
		setVerticalDisplacement(targetLongitude);
	}

	public int getVerticalDisplacement()
	{
		Integer hz = (Integer)vars.get("vertical");
		return hz.intValue();
	}
```

```
	public void setVerticalDisplacement(int val)
	{
		vars.put("vertical", new Integer(val));
	}

	public int getHorizontalDisplacement()
	{
		Integer hz = (Integer)vars.get("horizontal");
		return hz.intValue();
	}

	public void setHorizontalDisplacement(int val)
	{
		vars.put("horizontal", new Integer(val));
	}

	public void setType(char type)
	{
		vars.put("type", new Character(type));
	}

	public char getType()
	{
		Character charType = (Character)vars.get("type");
		return charType.charValue();
	}

	protected String getCheckSum(String command)
	{
		int returnValue = 0;
		if (command.length() >0)
		{
			returnValue = command.charAt(0);
		}
		for (int i = 1; i < command.length(); i++)
		{
			returnValue = returnValue ^ command.charAt(i);
		}
		//convert to hex
		return (Integer.toString(returnValue, 16)).toUpperCase();
	}

	public String data = null;
	public HashMap vars = new HashMap();
}



/*
 * Created on Feb 14, 2005
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
package com.netrol.Data;

/**
 * @author nmehdizadeh
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
public class UpdateResponse extends SingleValueCommnicationPacket {
```

```
	public UpdateResponse()
	{
		setType('U');
	}

	public String toString()
	{
		String rawCommand = "$" + getType() + "," + getSpeed() + "," +
getHorizontalDisplacement() + "," + getVerticalDisplacement() + "," +
getBatteryLife();
		String command = rawCommand + "*" + getCheckSum(rawCommand);
		return command;
	}

	public void setSpeed(int sp)
	{
		speed = sp;
	}

	public int getSpeed()
	{
		return speed;
	}

	public void setBatteryLife(int bl)
	{
		batteryLife = bl;
	}

	public int getBatteryLife()
	{
		return batteryLife;
	}

	private int speed = 0;
	private int batteryLife = 0;

}
```