



Project Number: IIH-0002

# Optimization of an Autonomous Underwater Vehicle

A Major Qualifying Project  
submitted to the Faculty of  
WORCESTER POLYTECHNIC INSTITUTE  
in partial fulfillment of the requirements for the  
Degree of Bachelor of Science

by

Joseph Baker, Robotics Engineering

---

Christopher Frumento, Mechanical Engineering

---

Jacob Grzyb, Robotics Engineering

---

Taylor North, Mechanical Engineering

---

Date: March 4, 2011

Approved:

---

Professor Islam Hussein, Major Advisor

## Keywords

---

Professor William Michalson, Co-Advisor

1. Autonomous
2. Submarine
3. Hydrodynamics

*This report represents work of WPI undergraduate students submitted to the faculty as evidence of a degree requirement. WPI routinely publishes these reports on its web site without editorial or peer review. For more information about the projects program at WPI, see <http://www.wpi.edu/Academics/Projects>.*

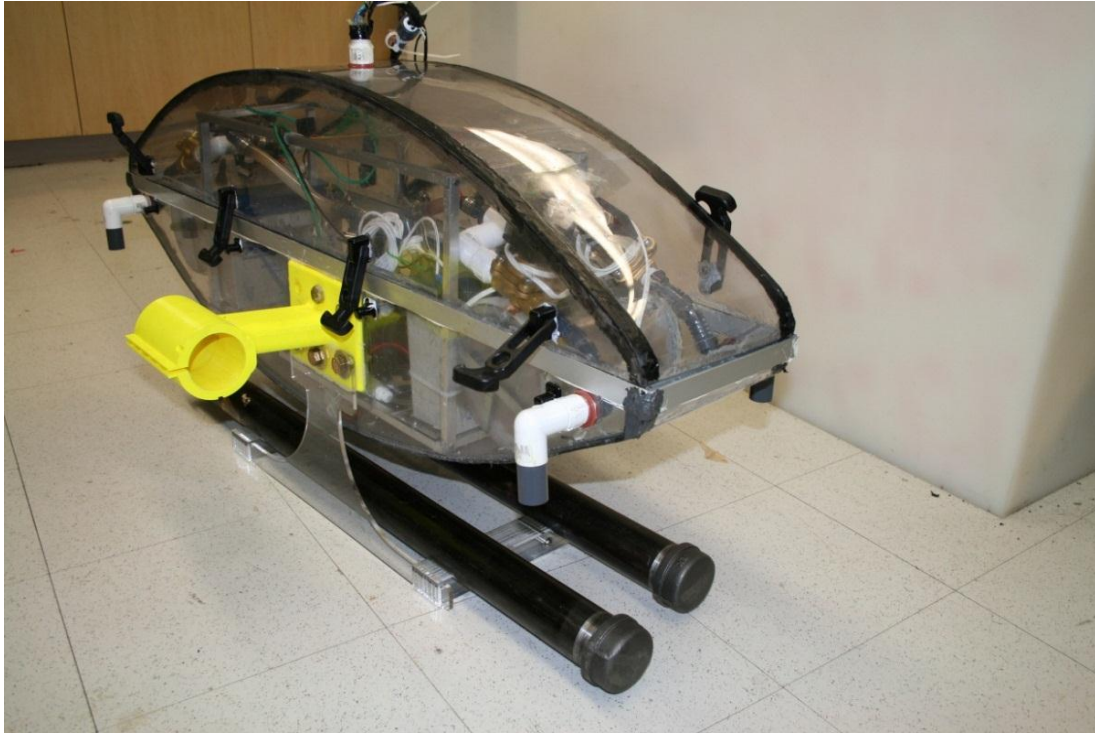


Figure 1: AUV with Ballast Holder

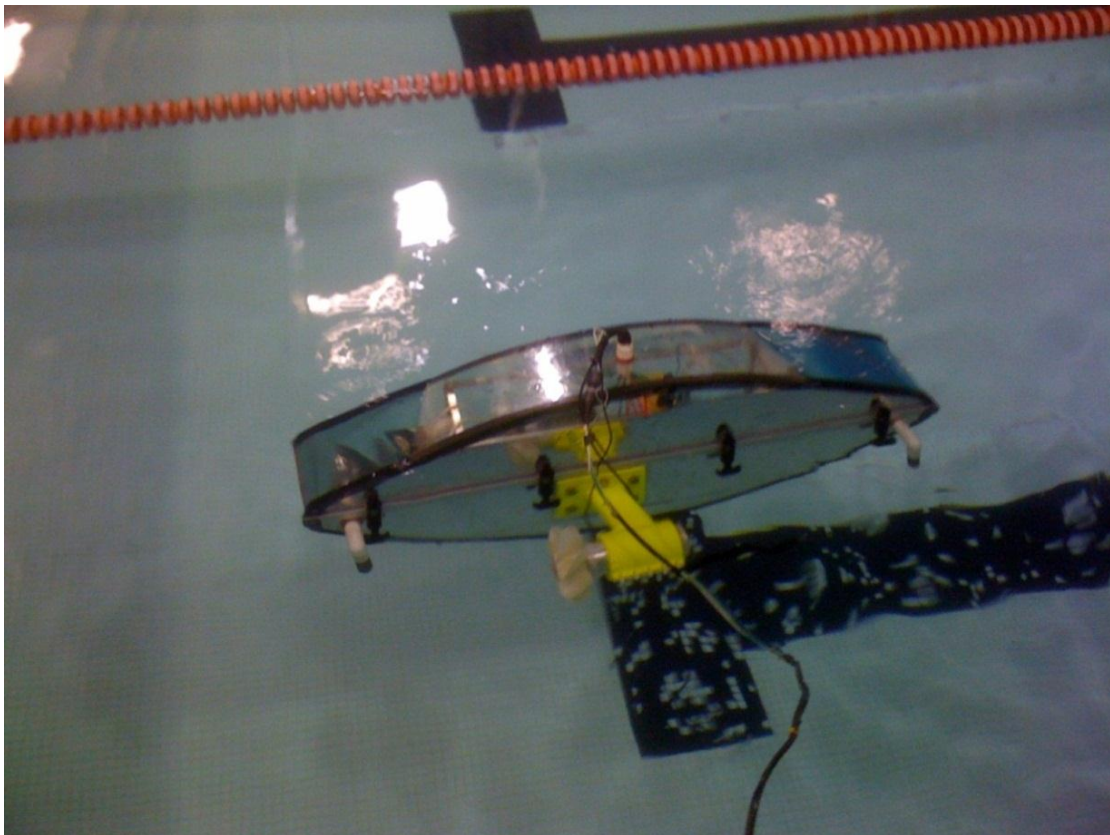


Figure 2: Leak Test Fall 2010

## **Abstract**

With autonomous vehicles becoming more prevalent in ocean applications, a necessity for affordable autonomous underwater vehicle (AUV) research is clear. The 2010-2011 WPI AUV MQP continued the work of previous project groups in optimization of AUV systems and control algorithms. This project also focused on upgrading the onboard control electronics, as well as the implementation and integration of an acoustic modem for tether-less underwater communication. An external ballast holder was designed, constructed and installed to increase diving performance and reliability of the AUV. A focus late in the project was underwater communication and networking.

## Executive Summary

The overall goal of this project is to develop a fleet of autonomous vehicles to operate cooperatively on an acoustic network. The use of acoustic signals for underwater applications is still a developing field. Acoustic signals are a convenient way to send data from one point to another underwater without the use of tethers.

The MQP teams at WPI have been tasked with the construction and implementation of an Autonomous Underwater Vehicle or AUV. The 2010-2011 MQP team focused mostly on optimizing the current AUV built by the previous teams. The team put the most effort in the implementation of new electronic hardware. The old PCB and MSP430 were replaced with a new designed daughter card and PIC32.

Other work went into the control and maintenance of current systems. Simple dynamic equations were developed and programmed so an open loop control was established. The acoustic modems were delivered and are ready to be implemented into the AUV. A new static ballast holder was designed and built for ease of moving the sub.

Additional work is required in integrating a sensor system in the vehicle. Once complete, the closed loop code can be completed and the vehicle will be fully autonomous. These additional efforts will eventually allow the vehicle to perform more complex missions such as searching for foreign objects and mapping seafloors, lakebeds and riverbeds. This will also allow communicating with other autonomous vehicles thus completing the overall goal of the project.

## **Acknowledgements**

We would like to thank Professor Islam Hussein and Professor William Michalson for their guidance throughout the duration of this project. We would like to thank Pat Morrison, Tom Angeliotti, Joe St. Germain, Adriana Hera, and Neil Whitehouse for their technical assistance. We would also especially like to thank Professor Fischer for his guidance in the design of our new controller PCB.

## Nomenclature

<b>ANx</b>	PIC32 ADC Input x
<b>A</b>	Amperes
<b>AFL</b>	Front-left actuator
<b>AFR</b>	Front-right actuator
<b>ARL</b>	Rear-left actuator
<b>ARR</b>	Rear-right actuator
<b>AUV</b>	Autonomous Underwater Vehicle
<b>MHz</b>	Megahertz
<b>MCU</b>	Microcontroller Unit
<b>PCB</b>	Printed Circuit Board
<b>kB</b>	Kilobyte
<b>ADC</b>	Analog to Digital Converter
<b>DAC</b>	Digital to Analog Converter
<b>SPI</b>	Serial Peripheral Interface
<b>I<sup>2</sup>C</b>	Inter Integrated Circuit
<b>IMU</b>	Inertial Measurement Unit
<b>I/O</b>	Input/Output
<b>ksps</b>	kilo samples per second
<b>PWM</b>	Pulse-Width Modulation
<b>Rxn</b>	PIC32 I/O Port x, Pin n (i.e. RB5 = Port B, Pin 5)
<b>SFE</b>	SparkFun Electronics
<b>SVM</b>	Supply Voltage Monitor
<b>USB</b>	Universal Serial Bus
<b>UART</b>	Universal Asynchronous Receiver/Transmitter
<b>V</b>	Volts
<b>W</b>	Watts
<b>WPI</b>	Worcester Polytechnic Institute

## Table of Contents

Abstract.....	3
Executive Summary.....	4
Acknowledgements.....	5
Nomenclature .....	6
List of Figures .....	12
List of Tables .....	13
List of Equations.....	13
Introduction .....	14
Background .....	16
AUV History .....	16
Submarine Ballast.....	17
Acoustic Underwater Communication.....	17
Methodology.....	19
Ballast Holder Design .....	19
Electronics Enclosure for WPI AUV .....	21
AUV Dynamics Equations and Simulation.....	22
Horizontal .....	23
Pitch .....	23
Roll .....	23
Yaw .....	24
Vertical.....	24
Electronics Refit.....	26
Microcontroller Selection .....	27
Power Systems.....	29
Sensor Systems .....	31
Motion Control systems .....	33
Communications Systems.....	35
Silkscreen Labels .....	36

Errata .....	36
Programming/Control .....	37
AUVLib .....	37
2009-2010 Final Mission Rewrite .....	40
Results .....	41
Simulations .....	41
Electronics .....	42
Programming .....	44
Modem Operations .....	44
Recommendations .....	46
Hull Redesign .....	46
Ballast System .....	47
Controller PCB .....	47
Fuses .....	47
3.3V/5V Rail Regulation and Protection .....	48
Voltage Reference Generation .....	48
Supply Voltage Monitor .....	48
RDS Header Placement .....	49
MOSFET Current Limiting .....	50
ISCP Programming Header .....	50
Appendix A: PCB Design and Fabrication Guide .....	51
Accessing the Eagle Files in SVN .....	51
Eagle Setup .....	52
PCB Fabrication .....	52
Appendix B: Vehicle Operation Guide .....	54
Vehicle Power and Battery Charging .....	54
Programming .....	55
MPLAB and C32 Compiler .....	55
Setting up Eclipse for AUV Development .....	55



Making a new Project using AUVLib.....	58
Downloading Code .....	61
Warning .....	61
Software Considerations .....	62
Electrical Subsystem Connections.....	62
Sealing the Vehicle .....	62
Transporting the Vehicle.....	63
Appendix C: Modem Operation Guide .....	64
Modem Setup.....	64
PuTTY Setup.....	66
Modem Modes of Operation .....	68
Sending and Receiving Commands .....	69
Troubleshooting .....	70
Appendix D: MATLAB Code for Motion Simulations.....	71
Simple Motions .....	71
Vertical Maneuver.....	72
Ballast .....	72
Ballast ODE .....	75
Dive Maneuver Lookup Table Generator.....	76
Appendix E: Controller Revision 2.0 Schematics .....	78
PIC32 Connector (upper half).....	78
PIC32 Connector (lower half) .....	79
Compass Amplifiers.....	80
.....	80
Compass and Compass Set/Reset Circuit.....	81
External Communications .....	82
Gyro.....	83
Motors.....	84
Power Connections .....	85

Pump .....	86
Solenoids and Actuators .....	87
Temperature and Pressure Sensors .....	88
Voltage Reference Amplifier and 5V Regulator .....	89
Appendix F: AUVLib Documentation .....	90
File List.....	90
auv.h File Reference .....	91
Detailed Description .....	91
system.h File Reference .....	92
Defines.....	92
Functions .....	92
Variables .....	92
Detailed Description .....	93
Define Documentation .....	93
Function Documentation.....	93
communication/acoustic_modem.h File Reference .....	95
Defines.....	95
Functions .....	95
Variables .....	95
Detailed Description .....	95
Define Documentation .....	95
Function Documentation.....	95
communication/uart.h File Reference .....	96
Functions .....	96
Detailed Description .....	96
control/base_control.h File Reference .....	97
Defines.....	97
Functions .....	97
Detailed Description .....	99

Define Documentation .....	99
Function Documentation.....	99
control/open_loop_maneuvers.h File Reference .....	105
Defines .....	105
Functions .....	105
Detailed Description .....	105
Function Documentation.....	105
sensors/analog_sensors.h File Reference.....	108
Defines .....	108
Functions .....	108
Variables .....	108
Detailed Description .....	108
Define Documentation .....	108
sensors/gyro.h File Reference.....	109
Functions .....	109
Detailed Description .....	109
Function Documentation.....	109
References .....	110

## List of Figures

Figure 1: AUV with Ballast Holder.....	2
Figure 2: Leak Test Fall 2010.....	2
Figure 3: MIT Odyssey IV.....	16
Figure 4: Acoustic Modem .....	18
Figure 5: Final Ballast Holder Assembly.....	20
Figure 6: CAD Rendering of Electronics Enclosure .....	21
Figure 7: Coordinate System.....	22
Figure 8: PIC32 Starter Kit (Top) .....	29
Figure 9: PIC32 Starter Kit (Bottom) .....	29
Figure 10: Horizontal Maneuver (3 meters) .....	41
Figure 11: Dive Maneuver (5 meters).....	42
Figure 12: PCB Top .....	43
Figure 13: PCB Bottom .....	43
Figure 14: Command Window .....	44
Figure 15: Current modem setup.....	44
Figure 16: Prototype Fiber Glass Hull .....	47
Figure 17: Improved SVM Circuit .....	49
Figure 18: TortoiseSVN Checkout .....	51
Figure 19: Eagle Configuration.....	52
Figure 20: 12V 3A battery charger for the AUV batteries .....	54
Figure 21: Adding SVN Repository to Eclipse .....	57
Figure 22: New Eclipse Project.....	58
Figure 23: Eclipse Project Include Paths .....	59
Figure 24: Eclipse Project Library Paths.....	60
Figure 25: PIC32 Starter Kit power input schematic.....	61
Figure 26: First modem .....	64
Figure 27: Second modem .....	65
Figure 28: Modem transducers.....	65
Figure 29: Hydrophones.....	66
Figure 30: PuTTY Configuration .....	67
Figure 31: PuTTY Serial Configuration .....	67
Figure 32: Modem Switches .....	68
Figure 33: Modem startup .....	69
Figure 34: Modem transmission .....	69
Figure 35: Modem Reception .....	70

## List of Tables

Table 1: Eagle Custom Device Library.....	28
Table 2: Comparison of MSP430 and PIC32.....	28
Table 3: Current Load Distribution .....	30
Table 4: Fuses.....	31
Table 5: 3-axis Compass.....	33
Table 6: Thruster Connections.....	33
Table 7: Ballast System Connections .....	34
Table 8: Motor Connections .....	34
Table 9: Acoustic Communication Control .....	38
Table 10: System Health Checks .....	39

## List of Equations

Equation 1: Force Equation.....	22
Equation 2: Velocity Equation.....	22
Equation 3: Distance Equation.....	22
Equation 4: Horizontal Force Equation.....	23
Equation 5: Pitching Moment Equation.....	23
Equation 6: Rolling Moment Equation.....	24
Equation 7: Yawing Moment Equation .....	24
Equation 8: Vertical Force.....	25
Equation 9: Mass Flow-rate .....	25
Equation 10: Bernoulli Equation .....	25
Equation 11: Pressure.....	25

## Introduction

The 2010-2011 Autonomous Vehicle (AUV) Major Qualifying Project (MQP) team began the third year of work on the WPI AUV during A-Term of 2010. The previous project groups had designed and built an AUV. At the culmination of the 2009-2010 project team's work resulted in preliminary autonomous mission testing of the WPI AUV at the WPI pool. The 2008-2009 AUV project team designed and built many of the components and systems of the AUV. The 2009-2010 project team worked to optimize the onboard systems of the AUV and work towards complete autonomy of the AUV. This year's project team began by focusing on the problems that became evident in the final mission testing of the AUV in the spring of 2010. As work began, further room for optimization was discovered and the goals of the project shifted to further optimization, replacement, and design of AUV components.

An initial goal of the project team was to recreate the final mission of the 2009-2010 MQP and to troubleshoot and remedy the problems with the mission test in the spring of 2010. During that test the AUV failed to submerge below the surface. The 2010-2011 project team focused on the ballast system itself as well as the electronic control module. These two areas were the main focus of the 2010-2011 project group.

One early goal of the group was to have the AUV operating autonomously by the end of A-term such that the group would have video of the mission test to present at the Workshop on Underwater Networks (WUWNet) at Woods Hole Oceanographic Institute (WHOI). It became clear early in the project that a full scale water test would not be possible during a-term because of electronics problems. However, the group did present the WPI AUV at WHOI and learned a great deal from touring the facilities there.

For most of the duration of the project, the team split the work into three major categories: an electronics group, a mechanical systems group, and a controls group. The electronics group spent a majority of the time working on the trouble shooting of the onboard electronics and the eventual redesign of the onboard computer board. The mechanical systems group focused on the optimization and repair of onboard systems and the development of an external ballast holder. While the controls group focused on simulating the motion of the AUV

as well as the program coding for AUV maneuvers. The project team found this to be a very productive way to accomplish many tasks in an efficient manner.

In the future, the goal for the WPI AUV project is to have an operational fleet of AUVs which can operate together autonomously and share information to accomplish a mission. With this being the overarching goal, there is great opportunity for AUV development and experimentation using the WPI AUV as a cost effective platform as the project moves forward. The 2010-2011 project team made significant progress in ensuring that the WPI AUV is a reliable tool for future teams to complete further experiments and testing.

## Background

### AUV History

AUV development began in the 1960s at the University of Washington and made very large leaps in the early 1990s with the backing of Massachusetts Institute of Technology (MIT) and the Office of Naval Research (ONR). The MIT AUV Lab created the “Odyssey Class” AUV which could be out-fitted for numerous underwater missions. MIT is has now developed the Odyssey IV, their latest iteration of AUV. Woods Hole Oceanographic Institute (WHOI) has also played a large role in the development of AUV’s. Their fleet of AUVs are the cutting edge of underwater autonomy and with the worldwide capability WHOI has, their work is some of the most highly respected oceanographic research in the world. This important research drives AUV development forward every year and with more sophisticated electronics the range of what these vehicles can do increases multifold.



Figure 3: MIT Odyssey IV

AUVs have numerous purposes. Most involve research but others can involve search and rescues, object identification, water sampling, mapping, or for our purposes here at WPI, communication and control research. AUVs can go places where is to too dangerous or expensive for humans to go. They can take numerous samples for research without human



interaction. Although AUVs are expensive pieces of hardware, the information they gather can be invaluable.

## **Submarine Ballast**

In order for a submarine to operate in an efficient manner, its overall density must be equal to that of the surrounding water. Most submarines can control their density through the use of a dynamic ballast system. Water ballast is taken aboard or deposited into the surrounding water to change the density and allow the submarine to manipulate its depth. Ideally, a submarine should be designed such that its natural density is slightly less than the density of the water it is in, such that it does not require a substantial amount of dynamic ballast to submerge. Static ballast, or dead weight, can also be utilized to change the natural buoyancy of the submarine so that the size of the submarine's ballast tanks can be reduced, opening up more room inside the submarine for other systems.

## **Acoustic Underwater Communication**

The use of acoustic signals for underwater applications has been around for many years however the use of acoustic signals for communication and data transfer is still a developing, dynamic field. These signals can be a convenient way to send data from one point to another underwater. Applications of this technology are diverse. Acoustic communication can be utilized to communicate from ship or buoy to a submarine, data transfer from unmanned or autonomous underwater vehicles, or for a group of AUV's to communicate with each other to work cooperatively to accomplish a task as is the eventual, long term goal for this project.

There are some drawbacks to acoustic communication in water versus traditional terrestrial communication. Overall, subsurface communication is slow due to the speed of sound in water which is approximately 1500 meters per second while communications above the surface can travel at the speed of light ( $3 \times 10^8$  meters per second). The relatively slow speed of sound in water limits the amount of data that can be transferred in a given time, resulting in a low rate of data transfer. However these sacrifices in performance are well worth the advantages of being able to transfer data without tethers underwater.

The AUV team has received two acoustic modems and two pairs of hydrophones from Professor Hussein's colleagues at the University of Connecticut. The modems take data input from a computer, convert the data into acoustic signals and transmit them through the hydrophones. The modems work both as transmitters and receivers so that the modem in the AUV will be able to receive acoustic signals as well as transmit data back to a receiving station.



**Figure 4: Acoustic Modem**

The eventual goal of this project is to have a fleet of AUV's that will be able to work cooperatively to accomplish a task. This will require a sophisticated communication based heavily on acoustic underwater signals. Needless to say, the acoustic modems are an integral part of this goal.

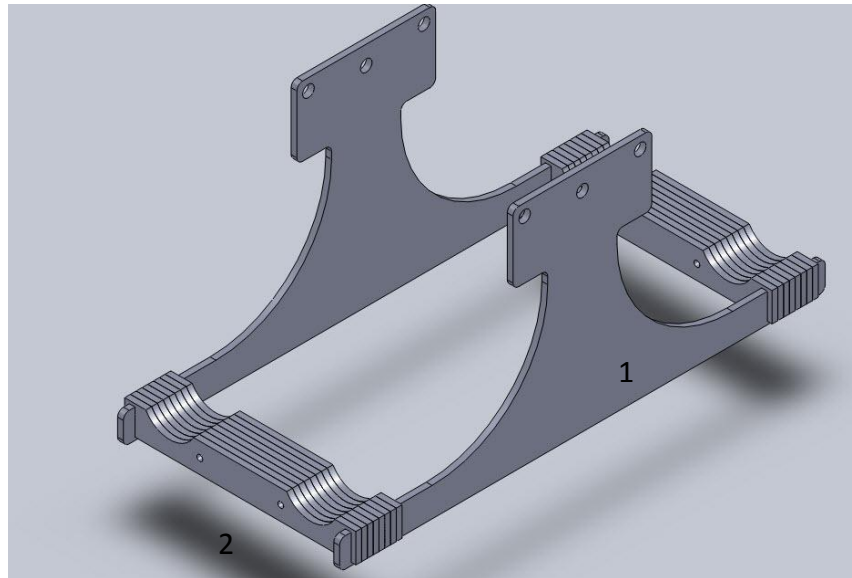
## Methodology

### Ballast Holder Design

For the WPI AUV, the project team decided to design an external static ballast system. The large interior volume of the AUV requires over 100lbs of lead ballast to achieve neutral buoyancy; where the weight of the submarine is equal to the buoyant force of the submarine when it is just submerged at the surface. Previously, the lead ballast was placed in bags and latex balloons, which had to be placed at specific locations within the hull. This confined the space within the AUV for other systems and interfered with wiring and plumbing for the current infrastructure. Therefore there was a need for a new static ballast design. It was decided that most of the static ballast weight should be affixed semi-permanently to the exterior of the hull, while about 10lbs would remain inside of the submarine and will be easily removed such that the amount of static ballast can be easily changed with the addition of new components to keep the natural density of the AUV constant.

The ballast holder design consists of three components:

1. Ballast Holder Bracket-Attaches to Hull and supports Ballast Holder
2. Ballast Holder- Connect to both ballast holder brackets and support both ballast tubes.
3. Ballast Tubes-2" Diameter, 4' Long lead shot filled tubes



**Figure 5: Final Ballast Holder Assembly**

The assembly was designed in such a way so that all of the parts could be quickly made using a laser cutter. Quarter-inch clear Acrylic was the material of choice because of its compatibility with the laser cutter, esthetic qualities and low cost.

Other important design qualities were the ease of attaching/removing the ballast tubes to the AUV. The tubes sit in semicircular grooves in the ballast holder. The force of gravity keeps the tubes secure during all normal operation and then can be easily lifted off of the holder to make launching and retrieval of the AUV from the water easier.

## Electronics Enclosure for WPI AUV

Previously, the control electronics were simply suspended from the inner frame of the hull by zip ties. With a great deal of expensive and complicated electronics now being onboard the WPI AUV, there was a definite need for a way to protect and properly mount these crucial components inside the hull of the AUV. The enclosure houses an acoustic modem and control electronics securely to the internal framework of the AUV. The enclosure reduces the chances of electronics damage in the event of minor water intrusion. Please note, however, this enclosure was not designed to protect the onboard electronics from catastrophic hull failure and therefore is not completely waterproof.

The electronics enclosure for the WPI AUV was completed in the middle of C-Term 2011. The interior of the rectangular case measures 10x9x3 inches which allows ample room for the acoustic modem, the AUV's control electronics board, and clearance for all wire connections. The enclosure was made of  $\frac{1}{4}$  inch clear acrylic for simple fabrication using WPI's laser cutter. The seams were bonded using a special acrylic adhesive. The top is held in place with two elastic shock chords and has a plastic gasket to ensure a good fit. The shock cord lid system allow for the top of the enclosure to be removed entirely to allow for complete access to the enclosure's contents. The box will be mounted vertically in the AUV. The fore and aft faces of the enclosure have slots, such that all the necessary ports on the modem and board may be accessed. One end of a slotted side features a 1.7 x 1.7 inch cut-out to allow for large serial port connectors to be connected. After the electronics are mounted permanently within the enclosure, acrylic inserts may be bonded into place in the slots to further reduce the chances of water intrusion.

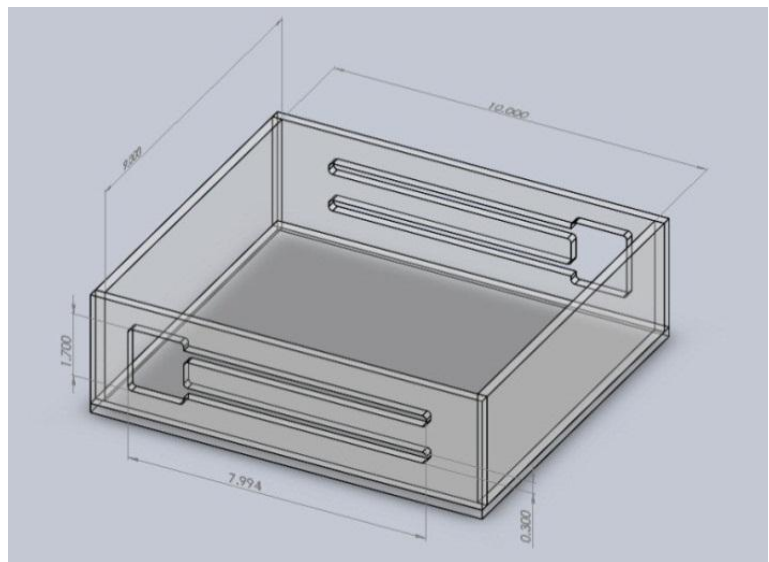


Figure 6: CAD Rendering of Electronics Enclosure

## AUV Dynamics Equations and Simulation

Basic motion equations were derived to allow for open-loop control of the AUV. Using these equations the AUV could be controlled both autonomously and remotely. A coordinate system with the z-axis acting on the gravitational axis allows the AUV can be positioned in the water, while the equations will be simulated in an orthogonal coordinate system.

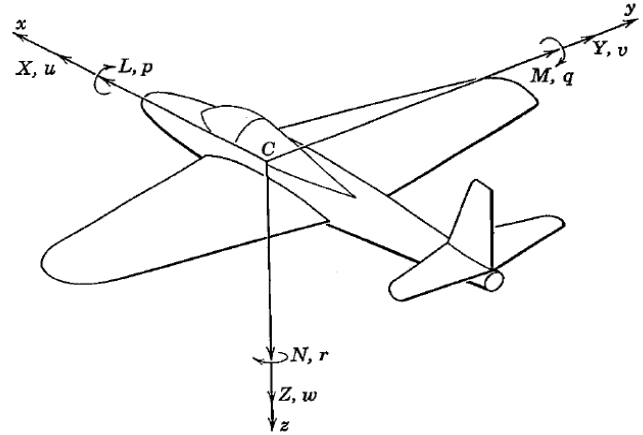


Figure 7: Coordinate System

Using Newton's second Law it is possible to derive equations for the distance in terms of time. The force and mass remains constant thus the acceleration would also be constant. Every motion can be solved this way with the exception of the Z-motion. By integrating the acceleration with time twice, a simple distance verse time equation achieved.

Equation 1: Force Equation

$$\frac{F}{m} = a$$

Equation 2: Velocity Equation

$$\int a dt = at + C1 = v$$

Equation 3: Distance Equation

$$\int v dt = \frac{at^2}{2} + C1 t + C2 = x$$

Each motion is broken into two phases, an acceleration from zero velocity and a deceleration back to zero velocity. Using initial conditions each phase can be broken down and analyzed. In the first phase, the constants of integration would be zero. In the second phase C1 would be the velocity at the end of the first phase, and C2 is the distance at the end of the first

phase. The only difference between each motion is the acceleration. The MATLAB code for each motion can be found in Appendix D.

## Horizontal

The two main thrusters mounted on the side of the AUV are able to provide horizontal motion. Firing them simultaneously will create the force necessary to propel the AUV. The first phase will have the AUV propelled either forward or backwards from a stationary position. In the second phase, the thruster will be fire in the opposite direction to slow the AUV back down to rest.

**Equation 4: Horizontal Force Equation**

$$X = m \frac{d^2x}{dt^2}$$

## Pitch

Pitch is controlled by firing the two water jet thrusters located on the bow or the stern of the AUV. The maneuver is done by firing the ones located on one side, and then firing the others located on the opposite side. Pitch could also be achieved by filling one of the ballast tanks, though the ballast tanks are hard to model.

**Equation 5: Pitching Moment Equation**

$$M = Iy \frac{dq}{dt}$$

## Roll

Roll is achieved almost exactly the same way as pitching. The only difference is that the two thrusters to be used have to be located on either port or starboard side of the AUV. Though this motion can be achieved it recommended that it not be purposely attempted. Rolling the sub would put strain on the Lexan hull and possibly break it apart. Rolling and pitching should be used solely for maintaining proper orientation of the sub.

#### Equation 6: Rolling Moment Equation

$$L = I_x \frac{dp}{dt}$$

#### Yaw

Yaw is performed by firing the main thrusters in opposite directions. Firing the thrusters in this fashion will turn the AUV. The first phase will initiate the turn by speeding the sub clockwise or counter-clockwise. The second phase the thruster will be fire in the opposite direction so the rate of AUV rotation will return to zero.

#### Equation 7: Yawing Moment Equation

$$N = I_z \frac{dr}{dt}$$

#### Vertical

The vertical maneuver is simulated in MATLAB to determine the timing of several components. Unlike the other motions the vertical down motion combines the pump and the water-jet thrusters to position the vehicle at a desired depth. The maneuver is fairly complicated to calculate. Some of our assumptions were to neglect disturbances in the water and head-loss through the pipes. An important assumption that is made is that the water the AUV moves through is an incompressible fluid. This assumption holds true for the depths that this AUV is designed for (12ft or bottom of WPI pool). The density of water at these depths is assumed to be constant.

To start the maneuver, first the ballast tanks must be filled; this will increase the density of the sub. Next, the ballast tanks are emptied using the pressure in the ballast tank. Finally, the thrusters fire to reduce downward velocity to zero. To analyze this maneuver, start with the force equation used in the other maneuvers, but here there are two forces that make up the buoyant force. The pump has a constant mass flow rate so we can assume the sub sinks relatively in line with the pump. The pressure calculation was done in MATLAB but is not



necessary. Then the equations that determine the mass flow rate in and out of the sub are used.

All the equations below are variable to change with depth. The most effective means to solve this is to use numerical methods by integrating and then interpolating to an answer. After running through multiple calculations it was determined that if the ballast tanks are pressurized enough the ballast tanks will empty relatively quickly and are fairly constant. The final phase of this motion use water-jet thrusters to slow to a stop.

**Equation 8: Vertical Force**

$$m \frac{d^2z}{dt^2} = B_y = m g - \rho g (\text{Vol.})$$

**Equation 9: Mass Flow-rate**

$$\frac{dm}{dt} = \rho VA$$

**Equation 10: Bernoulli Equation**

$$\frac{P_1}{\rho} + \frac{V_1^2}{2} = \frac{P_2}{\rho} + \frac{V_2^2}{2}$$

**Equation 11: Pressure**

$$P = P_{atm} + \rho g z$$

## Electronics Refit

Given the myriad of issues with the existing microcontroller board, it was deemed to not be a reliable means of controlling the vehicle in its current state of repair. The initial plan at this point was simply to revise the existing board design to correct the errors in it and order a new PCB to be populated based on the revised design. Unfortunately, Pads Logic, the program that was used to design the original controller, is no longer available on the ECE lab computers and there are no other design programs or converters capable of handling these files. This made it necessary to recreate the entire board from scratch based on the schematics in the appendix of the 2008-2009 MQP final report.

The new PCB was designed in the free version of a program called Eagle. The free version limits PCB dimensions to 4 x 3 inches (the old PCB was 5.5 x 4 inches), which made the layout of the board more challenging as all component placement had to be optimized for space reasons. Purchasing a single-user student license (\$125) would have increased this limitation to 6.3 x 3.9 inches had the need arose.

Not all of the devices used are available in the default devices libraries included with the program. Some of these could be found in other user-created libraries hosted by Cadsoft (the company that develops Eagle) as well as a large library supplied by SparkFun Electronics which contains most of the components they sell. However others had to be created by hand for the project. A device in Eagle consists of an electrical circuit symbol, a physical package footprint and a mapping of pin connections between the two. Devices were created using existing symbols and/or packages when possible, often originating from the SFE library and the included package reference library. A device and its associated package and symbol have to be located in the same library file, so a custom library for the AUV components (auv.lbr) was created for this purpose. A list of custom devices and the source of their packages/symbols is shown below. Note that in copying devices from one library to another, some extraneous symbols/packages are copied as well. Anything in the custom library not listed here was not used.

Device	Symbol source	Package source
A3A-12DA-2SV	sparkfun.lbr	WPI
A3A-12PA-2SV	sparkfun.lbr	WPI
DMG9926USD	transistor-fet.lbr	ref-packages.lbr
FDB5800	sparkfun.lbr	ref-packages.lbr
FUSE	fuse.lbr	WPI
FX10A-120S/12-SV(71)*	conn-hirose-pic.lbr/WPI	conn-hirose-pic.lbr
HMC1043	WPI	sparkfun.lbr/WPI
IRF7509	transistor-fet.lbr	WPI
MSI1451	WPI	WPI
OPA358	WPI	ref-packages.lbr

**Table 1: Eagle Custom Device Library**

\*This device was originally in conn-hirose-pic.lbr; the symbol was edited so that the pin names reflected the signals present on the PIC32MX360F512L.

## Microcontroller Selection

Recreating the entire board opened up a number of design decisions that otherwise would not have been altered on the previous board. First and foremost among these was the choice of microcontroller. Had the old Pads Logic files simply been revised, swapping out the microcontroller would have been out of the question due to the extra time investment required to properly reroute all of the signal connections. After consulting with Professor Fischer about various options available (including dropping in one of the new RBE Development Boards), the decision was made to upgrade the microcontroller to a PIC32. The increased processing power over the old MSP430 will allow more to be done at the microcontroller level that previously required the use of the PC/104 stack, the presence of which places an additional load on the batteries of up to 30W. Shown below are the relevant specifications of the PIC32MX360F512L and the old MCU, an MSP430F233:

Specification	MSP430F233	PIC32MX360F512L
Architecture	16-bit	32-bit
CPU Frequency	16 MHz	80 MHz
Program Memory (Flash)	8 kB	512 kB
Data Memory (RAM)	2 kB	32 kB
I/O Pins	40	85
Timers	2	5
PWM Outputs	6	5
UART Channels	1	2
SPI Channels	1	2
I <sup>2</sup> C Channels	1	2
ADC Inputs	8	16
ADC Sample Rate	200 ksps	500 ksps
ADC Resolution	12-bit	10-bit
Power consumption	1.27 mW	181.5 mW

Table 2: Comparison of MSP430 and PIC32

A PIC32 Starter Kit was purchased, which is a 2x2" PCB containing a PIC32MX360F512L MCU, power LED, clock circuitry, and programming circuitry. Two other variants of the Starter Kit are available which also have a second USB port or Ethernet port respectively. Neither of these was deemed necessary after confirming that the acoustic modem being developed by the UConn research group uses an RS-232 serial interface. Rather than being soldered directly to the controller PCB, this external board has a 120-pin socket which can be connected to anything with the mating connector. There are two advantages to this configuration. First, the full controller board does not require physical access to a computer to be programmed; the Starter Kit board can simply be unplugged and programmed anywhere without removing the daughter card from the vehicle. Second, any future revisions of the daughter card will not need to purchase a new MCU or any of the other components on the Starter Kit board. The theoretical future revision of the daughter card would have the same mating socket connector, and the

Starter Kit would be connected in the same fashion as with the current design. The programming interface is a mini-USB port which requires no external debugger or RS232 adapter.



**Figure 8: PIC32 Starter Kit (Top)**



**Figure 9: PIC32 Starter Kit (Bottom)**

## Power Systems

Several revisions were made to the power systems of the controller from the previous version.

## Power Inputs

The previous controller called for a separate 12V input to each of the four 12V rails as well as 5V and 3.3V inputs from a Pico-ATX computer power supply. It quickly became apparent, especially given the incident which damaged the Pico-ATX supply, that this was not an ideal system. The new controller does not require the Pico-ATX supply at all. There is a single 12V banana clip input and another banana clip input which connects to the ground plane on the bottom of the PCB. This single 12V input is connected to all four separately fused 12V rails, two of which are on each side of the board. One of these is connected to the input pin of an LM7805 voltage regulator, which outputs 5V to the PIC32 Starter Kit and the pump signal

amplifier. The onboard circuitry on the PIC32 Starter Kit then outputs 3.3V. The end result is that six power input connections are replaced with one.

### 12V Bus Current Requirements

One of the critical flaws in the original controller was that the 12V bus traces were not wide enough to carry the required amount of current, resulting in the wire around the edge of the board soldered to each of the MOSFETs. Careful attention was paid to the width of each 12V trace to ensure that no further burnouts occur. Calculation of the correct minimum trace widths requires knowledge of not only the current loads being placed on the trace, but also the fabrication process being used. Cheaper fabrication processes like the one used for the new controller use a thinner layer of copper on the surface of the PCB, reducing the cross-sectional area of the trace and thus the amount of current it can carry. All trace width requirements were calculated for a trace thickness of 1 ounce of copper per square foot, which is the value that was used for fabrication. With more expensive fabrication processes, trace thicknesses of up to 5 ounces of copper per square foot can be achieved, thus reducing width requirements.

The four 12V rails on the new controller have been designed to evenly distribute the current load as much as possible such that no particular bus trace has to be excessively wide. The loads have been split as shown below.

<b>Bus</b>	<b>Connection</b>
12V1	Motors
12V2	Aft ballast solenoids Aft maneuvering thruster actuators
12V3	Fore ballast solenoids Fore maneuvering thruster actuators
12V4	Pump All 5/3.3V systems

**Table 3: Current Load Distribution**

## Fuses

Shortly before ordering the controller PCB, it was discovered that the fuse holders and matching fuses used on the original controller are no longer sold. Similar Littelfuse 564 series fuse holders and matching 396 series fuses were chosen as a replacement. The rated current values for each 12V rail are shown below.

Bus	Fuse Value
12V1 (Motors)	6.3A
12V2 (Aft)	5A
12V3 (Fore)	5A
12V4 (Pump/PIC)	5A

Table 4: Fuses

## Sensor Systems

### Cabin Temperature

The cabin temperature sensor circuit was unchanged from the original schematics other than the addition of power filtering capacitors. The need for this sensor was in question given the results of the heat generation tests in the original report. However, it was decided to keep the sensor in case of heat building up in the new electronics enclosure. This sensor was connected to input AN1 on the ADC.

### 3-Axis Compass

The original controller schematics called for a HMC1052 2-axis compass. However, if the board were to be mounted on its side as it is with the new enclosure, this compass would be crippled. Therefore, it was replaced with a HMC1043, a similar 3-axis model. All three axes are connected to allow the PCB to be mounted in any orientation as shown below.

Signal	ADC Input
COMPX	AN8
COMPY	AN9
COMPZ	AN5

**Table 5: 3-Axis Compass**

## 6-Axis IMU

The new revision of the controller PCB fully supports the ADIS16354 6-axis IMU which was purchased by the first MQP group. The two headers on the board (Hirose part # A3A-12PA-2SV) are the same pin headers on the IMU's breakout board. Mating connectors were purchased to be made into two small ribbon cables to connect the controller to the gyro breakout board. A separate 1-pin header is also connected to the IMU's ADC input to allow an external input to be connected to the ADC without interfering with the primary header connection.

## Pressure

The original controller made provisions for multiple pressure sensors for depth as well as the ballast tanks. However, these were never implemented. The sensors were never populated on the original controller PCB, and no design was ever conceived for connecting the transducers to the ballast tanks. Given this and the size restrictions of the new PCB, the ballast pressure sensors were removed. Although the depth pressure transducer was also never connected, a component had been made for the purpose which never had a hole drilled in the hull for it. The depth pressure sensor output is connected to the AN3 input.

## Supply Voltage

The supply voltage monitor circuit was not placed on the PCB itself for lack of a free ADC input pin near the 12V rails. However, the circuit was simple enough that it could be mounted on the prototyping board already in the submarine that the pump relay is mounted on. This



location gives easy access to a +12V connection. The analog voltage output is connected to the gyro ADC input header.

### Water Leak Sensor

Given the incomplete nature of the schematics in the old report and that it would be ideal to have a different circuit for each transducer; the water leak sensor circuit was not implemented on the board itself.

### Motion Control systems

With the new revision of the controller, the choice of MOSFETs used to control the maneuvering thruster actuators and ballast system was once again re-examined with the goal of optimizing the amount of physical space required on the board by these systems without compromising the functionality of the design. After researching the detailed specifications of several candidate components, the DMG9926USD was selected to replace the FDB5800 MOSFETs that were used by the previous group. Each DMG9926USD on the controller takes over the role of two FDB5800s in a SOP-8L package, which takes up less space than a single FDB5800.

### Maneuvering Thrusters

The maneuvering thruster jet actuators are connected as follows:

Signal	Connection
AFL	RB14
AFR	RB13
ARL	RA5
ARR	RA4

**Table 6: Thruster Connections**

## Ballast System Connections

The ballast solenoids and pump are connected as follows:

Signal	Connection
FFS	RB12
FDS	RB11
RFS	RA1
RDS	RA0
Pump	RA14

Table 7: Ballast System Connections

## Motors

The motor driving circuitry is largely unchanged from the original schematics other than the addition of power-filtering capacitors for the LMD18200T H-Bridge drivers. Signals to/from the H-Bridges are connected as follows. Note: For signals that do not use a general-purpose I/O port (PWM output signals from the PIC32 Output Compare pins and analog voltage inputs from the motor current sensors), the alternate functions of these pins (general-purpose I/O pins) are also given when applicable.

Signal	Connection
MODIR	RD6
MOBRAKE	RD7
MOPWM	OC5 (RD4)
MOTMP	RD3
MOCURRENT	AN15
M1DIR	RD2
M1BRAKE	RD1
M1PWM	OC1 (RD0)
M1TMP	RA3
M1CURRENT	AN0 (RB0)

Table 8: Motor Connections

## Communications Systems

Several interfaces were used on the controller to allow for external communication with as many different current and future devices as possible.

### UART 1 Header

The 3-pin header for UART 1 provides access to basic transmit/receive functionality as well as a ground pin. No clear-to-send/ready-to-send functionality is available.

### UART 2 Port

The DB9 RS-232 serial port is connected to UART 2. This port is capable of clear-to-send/ready-to-send functionality. The primary purpose of this port is to interface with the acoustic modems.

### I<sup>2</sup>C Header

The I<sup>2</sup>C header is connected to the I<sup>2</sup>C1 pins on the PIC32. This header is not currently used, but is available for future expansion.

### SPI 2 Header

Upon discovering that the PIC32 has two discrete SPI busses, it was decided that the gyro should be allocated a dedicated bus to itself. While researching all of the components used in the original controller schematics, it was found shortly thereafter that there are currently no other SPI devices present in the AUV. To allow for future expansion of the capabilities of the AUV, a 4-pin header was added to interface with the second SPI bus.

### Port E General Purpose I/O

Pins 0-7 of Port E were allocated a 4x2 pin header for digital I/O. These pins have no relevant alternate functions. The location of the header is ideal for connection of slave select lines from external SPI devices connected to the SPI2 bus.

## Silkscreen Labels

The previous controller marked the actuator connection headers with the default names (J16, J17, etc.), which made it necessary to refer to documentation in the previous reports to connect the actuators to the controller properly. This has been corrected with the new revision. Every header on the controller has the proper signal name clearly marked on the PCB silkscreen. To aid in troubleshooting, the status LEDs on each 12V rail have markings on the silkscreen indicating which components that status LED corresponds to. To prevent compass malfunctions due to improper mounting of the PCB, simple “FORE” and “AFT” labels were added to the silkscreen.

## Errata

All of the following errata have been corrected in the Eagle files located in the Subversion repository. There is a file release on the SourceForge page called Revision 2.1 of the controller which has these fixes – the Revision 2.1 PCB has not been ordered, but if a new one is purchased with the Revision 2.1 files, they will be corrected.

## Pressure Sensor

The pressure sensor pad layout is reversed on the PCB as-ordered and the side mounting pads are missing. A similar mistake was made as with the previous revision of the PCB.

## C35 Label

The label for C35 is incorrectly marked as C32 on the silkscreen label. The real C32 is also marked as being C32.

## Gyro Slave Select

Pin 3 of the GYROJ1 header is connected to the PIC32’s SS1 pin, which is the slave select input that is used to set the PIC32 as an SPI slave. This trace has been rerouted to RG15 in the Eagle files. As an interim fix, the trace has been ripped up on the current PCB and the gyro SS pin is wired to RE3 on the external header.

## Programming/Control

### AUVLib

AUVLib is a library of reusable microcontroller code for the AUV which is distinctly separated from the mission code. It contains functionality to make all of the hardware function, but does not execute any commands on its own. The full documentation can be found in Appendix F.

One of the problems with the existing code for the old controller was it required the programmer to have an intimate knowledge of the schematics of the controller board itself. For example, to control the pump, the programmer had to be aware that the pump, by convention, should be plugged into a header marked J14, which maps to pin 1 of I/O port 2 on the MSP430. This system made it very easy to make mistakes when writing code. One of the goals of AUVLib is to provide several layers of abstraction between the programmer writing mission code and the specific electrical connections on the controller board such that writing mission code requires no understanding of the electrical circuits and relatively little programming background. The control functionality of AUVLib was designed from the ground up so that each level of control functions adds another layer of abstraction.

### Basic Control

The basic control layer provides the most basic abstraction. It consists of a series of functions that simply turn actuators on and off and nothing more. The importance of this layer is that these functions are the only ones that directly interact with the I/O pins that control the actuators.

### Open-Loop Maneuvers

This layer performs single maneuvers (forward movement, yaw, vertical movement). It uses the basic control functions in conjunction with the calculations used in the Matlab simulations to allow the AUV to move a specified amount in the given direction. In the case of the dive maneuver, a lookup table had to be created because the calculations required cannot

be performed in a timely fashion using the microcontroller. The script used to generate this table can be found at the end of Appendix D.

### Closed-Loop Maneuvers

These functions will use the open-loop maneuver layer as a starting point. They also implement a software PID controller based on sensor data from the external pressure sensor, IMU, and compass. The PID controller will then make additional open-loop maneuver function calls as needed to make corrections.

### Communication

Text commands sent over the acoustic modem from a base station can be used to control the AUV. This interface can also be used to transmit sensor data and debugging information from the AUV back to the base station. The AUV will respond to commands to perform both open and closed-loop maneuvers, but not basic control functions. The syntax mimics that of the actual control functions.

Command code	Resultant function call
~OH_x;	open_horizontal_maneuver(x);
~OS_x;	open_surface_maneuver(x);
~OD_x;	open_dive_maneuver(x);
~OY_x;	open_yaw_maneuver(x);
~OP_x;	open_pitch_maneuver(x);
~RO_x;	open_roll_maneuver(x);
~CH_x;	closed_horizontal_maneuver(x);
~CS_x;	closed_surface_maneuver(x);
~CD_x;	closed_dive_maneuver(x);
~CY_x;	closed_yaw_maneuver(x);
~CP_x;	closed_pitch_maneuver(x);
~CR_x;	closed_roll_maneuver(x);

Table 9: Acoustic Communication Control

## System Health Checks

In order to help prevent/mitigate any problems which may occur within the AUV, the controller constantly monitors several internal system health sensors. If a problem is detected, the controller takes action to mitigate the problem where possible and attempts to communicate the nature of the problem over the acoustic modem. In the event of a critical failure, the controller will empty the ballast tanks to ensure that the vehicle can be recovered from the water as quickly as possible.

<b>Fault condition</b>	<b>Error text</b>	<b>Action(s) Taken</b>
Motor 0 temperature flag	M0 DRIVER OVERHEATING. BRAKING M0.	Apply M0 brake Set M0 duty cycle to 0%
Motor 1 temperature flag	M1 DRIVER OVERHEATING. BRAKING M1.	Apply M1 brake Set M1 duty cycle to 0%
Supply voltage below 10V	BATTERY LOW.	Restrict motors to 50% thrust (NOTE: This will break open-loop control)
Supply voltage below 9V	BATTERY CRITICAL, SURFACING.	Empty ballast tanks
Cabin temperature above 75°C	CABIN OVERHEATING. SURFACING AND SHUTTING DOWN SYSTEMS.	Empty ballast tanks Shut down heat-producing systems
Water leak*	<Report location of leak in the case of multiple leak sensors>	Empty ballast tanks Fire maneuvering jets if upright
External pressure above <unsafe level>	SAFE EXTERNAL PRESSURE LEVELS EXCEEDED, SURFACING.	Empty ballast tanks Fire maneuvering jets if upright
Gyro status bit errors	To be determined.	To be determined.
Gyro over range error	To be determined.	To be determined.

**Table 10: System Health Checks**

\*Sensor hardware not fully implemented at the time of writing

## Timing

The code written for the old controller did not make use of any of the MSP430's onboard timer circuits for accurate clocking of events, instead making use of a function which takes roughly the given amount of time to execute by repeatedly performing a mathematical operation with a known runtime on the given processor. In order to accurately execute open-loop maneuvers as well as checking sensor values, etc. at a known frequency, the PIC32's Timer 3 was used to provide a millisecond-accurate timestamp.

## 2009-2010 Final Mission Rewrite

As a simple test/demo of AUVLib, the final mission code from the previous MQP was rewritten using the new library. In the process of examining the old code in close detail in order to translate it into the new function calls, several errors were found which contributed to the partial failure of the final mission. The nature of most of the errors was such that they would not have occurred had basic on/off-level control functions had been written instead of directly setting the I/O ports to hexadecimal values. For example, at one point in the old code, the following 3 lines are called in sequence. The comments explaining what they do were added in the process of translating them to AUVLib function calls – the only possible explanation for this set of commands being performed is that it was not the intended commands, but the mistake was not caught because the code is not self-explanatory.

```
P1OUT|= 0x78; // Turn all 4 maneuvering jet actuators on
P5OUT|= 0x0F; // Turn FDS/RDS and FFS on (what?)
P2OUT |= 0x02; // Turn pump on
```



## Results

### Simulations

Each simple equation of motion was solved for distance in terms of time. The time gathered would then be used in determining the sub-systems run time. These equations were simulated in MATLAB to determine runtime for the AUV under ideal circumstances. Gathering these timings is important for setting up an open loop control of the AUV. The open loop control would give the team control of the AUV in a closed environment, like the swimming pool, to certain behavior of the sub and its systems.

As stated in the dynamics section each motion was broken down into two phases with the exception of the ballast, which has three. Using data gathered by the previous MQP teams the equations were defined and solved in the MATLAB. The code located in Appendix D, solves each maneuver. By a distance to travel the code will break the total distance up into parts and determine timings for each component of the AUV. Below are examples of the horizontal and vertical maneuver at a given distance and depth.

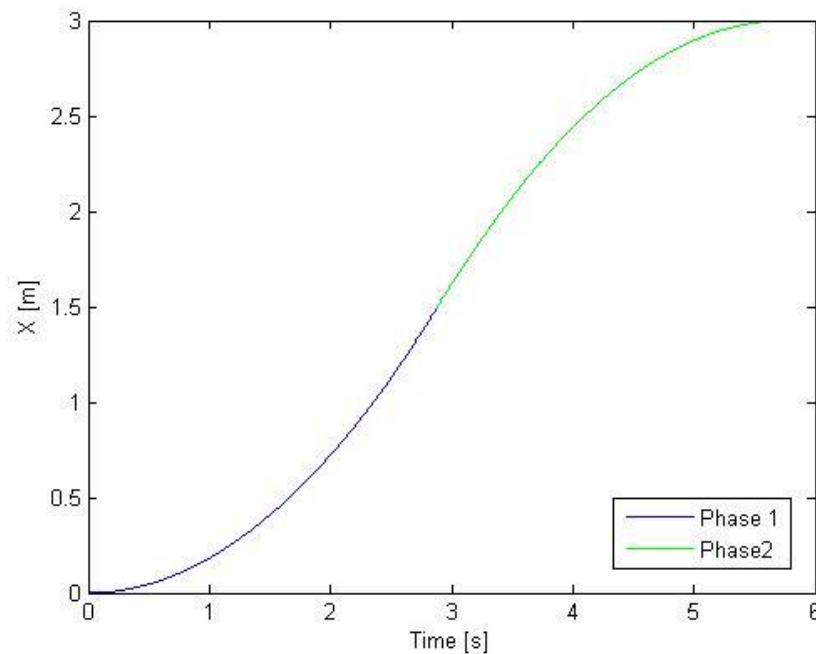
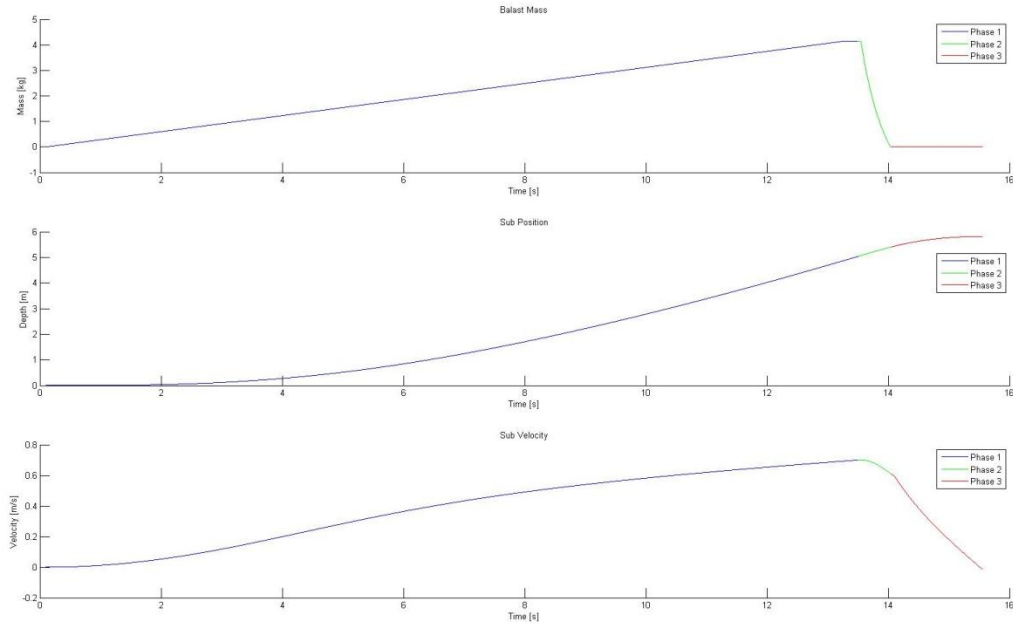


Figure 10: Horizontal Maneuver (3 meters)



**Figure 11: Dive Maneuver (5 meters)**

## Electronics

The Revision 2 controller PCB was largely a success. Although there is still some room for improvement in the design, the Revision 2.0 PCB is a good starting point for a fully functioning AUV where the control electronics are no longer a serious liability. The Revision 2.0 PCB was used to successfully demonstrate operation of the port thruster operating under the control of microcontroller code. The control electronics are no longer in danger of catastrophic failures under normal operating conditions due to trace burnouts or components functioning outside of their safe operating conditions. Setup of the AUV systems as well as the associated programming is now more intuitive with specific systems being permanently assigned to specific signal headers. RS232 serial communication support allows for integration of the acoustic modems, which was not possible with the original PCB. Upgrading the MCU to a PIC32 has allowed the control electronics to operate independently of a higher-power computer like the PC/104 stack. Shown below is the mostly-populated PCB.

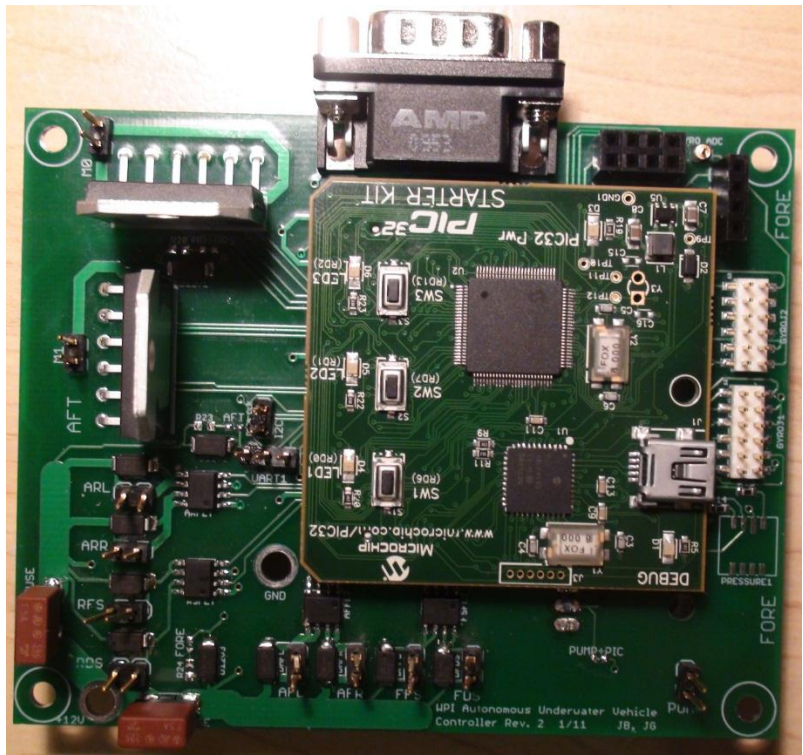


Figure 12: PCB Top

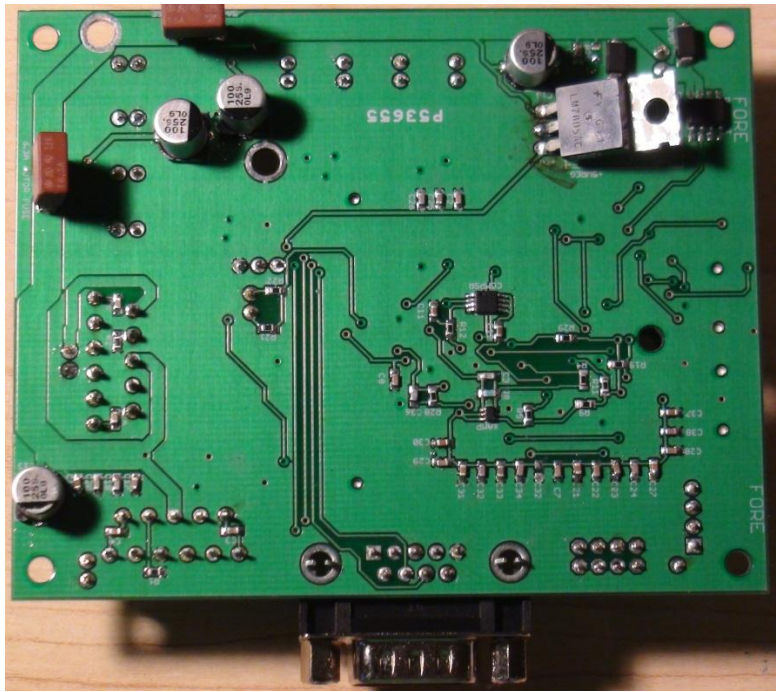


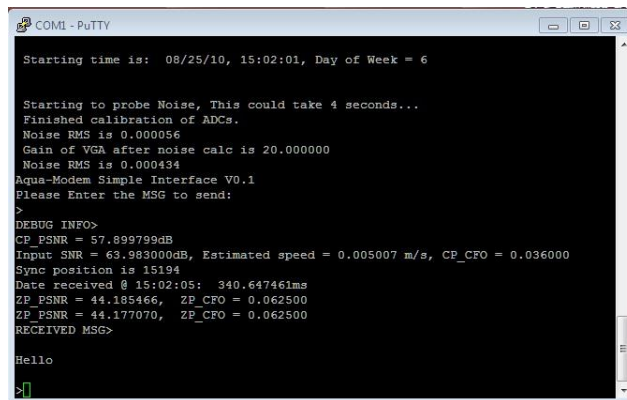
Figure 13: PCB Bottom

## Programming

AUVLib, while not entirely implemented at the time of writing, is on track to provide a stable library for future development of the AUV using good software engineering practices. Basic motion control and simple open-loop maneuvers are fully implemented. Implementation of the onboard internal sensor systems has begun. The current top priority is to make use the acoustic modems for external communication. This will allow the AUV to perform commands remotely as well as communicate important system health information about the vehicle and other sensor data back to the base station. Once that is complete, more sensors will be brought online in order to facilitate closed-loop control of the vehicle. The basic framework for all of these systems has been designed; the low-level code just needs to be implemented.

## Modem Operations

The acoustic modems from UConn were successfully setup and operated from the CAN MUVE lab at WPI. Both modems worked as expected and were able to send and receive data from one another. Below is a printout from a message received from one modem to another. The message sent was "Hello". As shown below currently there is a significant amount of debug information that is printed out with the received message. In future software upgrades to the modems this will be corrected.



```
COM1 - PuTTY
Starting time is: 08/25/10, 15:02:01, Day of Week = 6

Starting to probe Noise, This could take 4 seconds...
Finished calibration of ADCs.
Noise RMS is 0.000056
Gain of VGA after noise calc is 20.000000
Noise RMS is 0.000434
Aqua-Modem Simple Interface V0.1
Please Enter the MSG to send:
>
DEBUG INFO>
CP_PSNR = 57.892799dB
Input SNR = 63.983000dB, Estimated speed = 0.005007 m/s, CP_CFO = 0.036000
Sync position is 15194
Date received @ 15:02:05: 340.647461ms
ZP_PSNR = 44.185466, ZP_CFO = 0.062500
ZP_PSNR = 44.177070, ZP_CFO = 0.062500
RECEIVED MSG>
Hello
>
```

Figure 14: Command Window

The modems are currently set up on a table in the lab. This can be seen in the picture below. Because there was no aquarium in the CAN MUVE lab the hydrophone are currently in a bucket of water. Both modems are connected to a computer for testing purposes. In the future one modem will be mounted in the AUV and the serial connection from the modem will go to the new daughter card to facilitate communication to the AUV.



**Figure 15: Current Modem Setup**

## Recommendations

### Hull Redesign

One future aspect of the project that might need to be redone is the hull. Other designs of the hull that have been tossed around are a torpedo like shape and a modular tube structure. A torpedo shape would give the AUV a more hydrodynamic shape that would greatly reduce the drag that the current hull suffers from. The modular design would allow for quick swap of components or hull segments on the fly.

The new design concept of the hull is a football-like shape made out of fiber glass. The football shape would provide a comparable coefficient of drag to that of a torpedo. Another design feature would be to create a flange to seal the hull. Currently the sub is sealed by fitting the top of the hull in a gasket and rubbing Vaseline over edge to create a perfect seal. The flange would press to flat surfaces to gather and the force created would be enough to seal it. This would theoretically be easier and more reliable than reapplying Vaseline every time.

Fiber glass would be used to construct the hull because it is easier to work with than Lexan. Originally the idea was to use Lexan and mold it the same way glass bubbles can be made. Heating the Lexan and then blowing air under it by placing a mold ring on top of the Lexan would form the spherical shape of the hull. This idea, while interesting, is not optimal. Fiber-glass like that used on sailboats and the AUVs at Bluefin Robotics is much simpler to work with. Simply create a mold and then form the fiber glass over it. Another positive aspect of using fiber glass is the fact that fiber glass bonds to fiber glass, so any repairs or alterations that need to be made to the hull can be done easily, whereas Lexan might require a new hull to be made.

Figure 16 shows a mockup hull that was built with a catamaran type ballast system that was developed. Another external ballast system that was considered was a keel design like those found on sailboats.



Figure 16: Prototype Fiber Glass Hull

## Ballast System

Over the course of this project a new ballast system was often discussed. During the trip to WHOI the group was given a tour of some of their AUV labs. One AUV was a glider that utilized a piston like ballast system. A screw-drive motor is used to increase and decrease the volume and the ballast tank. The screw-drive creates linear motion that is used to draw water in and expel water out of the tank. Although the WPI AUV is not a glider this system could be used. This system would be very precise and provide extremely accurate depth control.

## Controller PCB

A future revision of the PCB would benefit from the following changes.

## Fuses

Rather than using a traditional fuse which has to be replaced by hand after a single fault event, the fuses could be replaced with a PTC (Positive Temperature Coefficient) resettable fuse. These devices act as a highly specialized variable resistor which switches from a very low resistance state to a very high resistance state when input current goes above a certain level. The fuse stays in this high-resistance state until power is fully cycled.

### 3.3V/5V Rail Regulation and Protection

A more reliable way to supply 3.3V would be to use a regulator connected to the 12V input (similar to the 5V regulator) instead of using the PIC32 Starter Kit 3.3V output pins to reduce load on the 5V regulator. In addition, the existing LM7805 linear regulator could be replaced with a V7805 switching regulator (in conjunction with a V7803 for the 3.3V rail). These switching regulators are more efficient as well as being compact. These components should be placed on the top side of the board for thermal management reasons.

Given the lack of a reverse voltage protection diode for the +5V\_EXT input on the PIC32 Starter Kit board, such a Schottky diode (or diodes as trace layout dictates) should be placed on the daughtercard to prevent the situation mentioned in the Programming section of the Vehicle Operation Guide.

### Voltage Reference Generation

Instead of using a combination of the PIC32's onboard reference generator and an amplifier circuit, more accurate reference voltages can be generated with voltage reference diode ICs.

### Supply Voltage Monitor

Given the changes in Revision 2.1 of the PCB, the SVM circuit can be fit on the PCB. This was not done for the current revision of the board due to the lack of free I/O pins on the bottom half of the Hirose connector. There are two possible ways to free up an ADC input given the current layout:

- Shift the FDS signal up to RE9, and then shift FFS and AFR up accordingly to move AFL off of AN14.
- Use the AN2 pin since the SS1 input should never be used.

The simple voltage divider currently used for the off-PCB SVM circuit is not a very power-efficient system. It constantly draws about 3.3 mW, even when not being sampled. A revised SVM circuit is shown below which only draws current when sampling. The SVM\_ON



signal (connected to a digital I/O pin on the PIC32) in this revised circuit must be switched on in order to sample. In addition, it may be worthwhile to install a microcontroller voltage supervisory circuit such as an MCP809 or MCP810 for additional protection against brownouts.

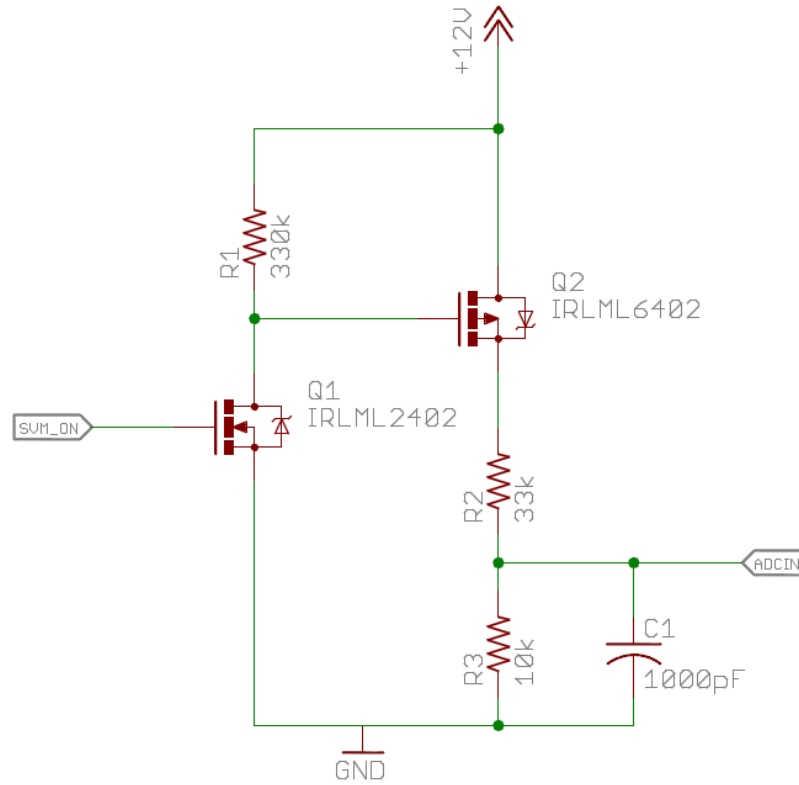


Figure 17: Improved SVM Circuit

### RDS Header Placement

Upon populating the board, it was found that the RDS header is in a very cramped location with the 12V input banana clip in place. While there is no imminent danger of a short circuit when the board is already connected, it is a concern when connecting the board. For a future revision of the PCB, simply swap the position of the diode and pin header for each aft actuator to remedy this situation.

## MOSFET Current Limiting

Professor Fischer recommended adding current-limiting resistors as a standard practice as a standard practice when driving high-current MOSFETs with a microcontroller. Further investigation regarding the values and exact location in the circuit of these resistors is required.

## ISCP Programming Header

Adding a 6-pin header connection for the PIC32's ISCP (In-Circuit Serial Programming) interface would allow code to be downloaded to the PIC32 with a debugger such as a PICkit2 without using the Starter Kit USB connection and thus eliminating many of the software limitations outlined in the device programming section of Appendix B.

## Appendix A: PCB Design and Fabrication Guide

### Accessing the Eagle Files in SVN

The first step is to access the files in the SVN repository outside of Eclipse (Eclipse should not be used here at all). This requires a program called TortoiseSVN, which is available at <http://tortoisesvn.net/downloads.html>. After installing TortoiseSVN, create an empty folder somewhere convenient on your computer and then right-click on it and click “SVN Checkout”. Fill in the directory paths as follows. Note that the empty folder in the example below is called “Eagle”; if that is the case then the default directory chosen would be “C:\Users\JOE\Documents\Classes\MQP\Eagle\Eagle”.

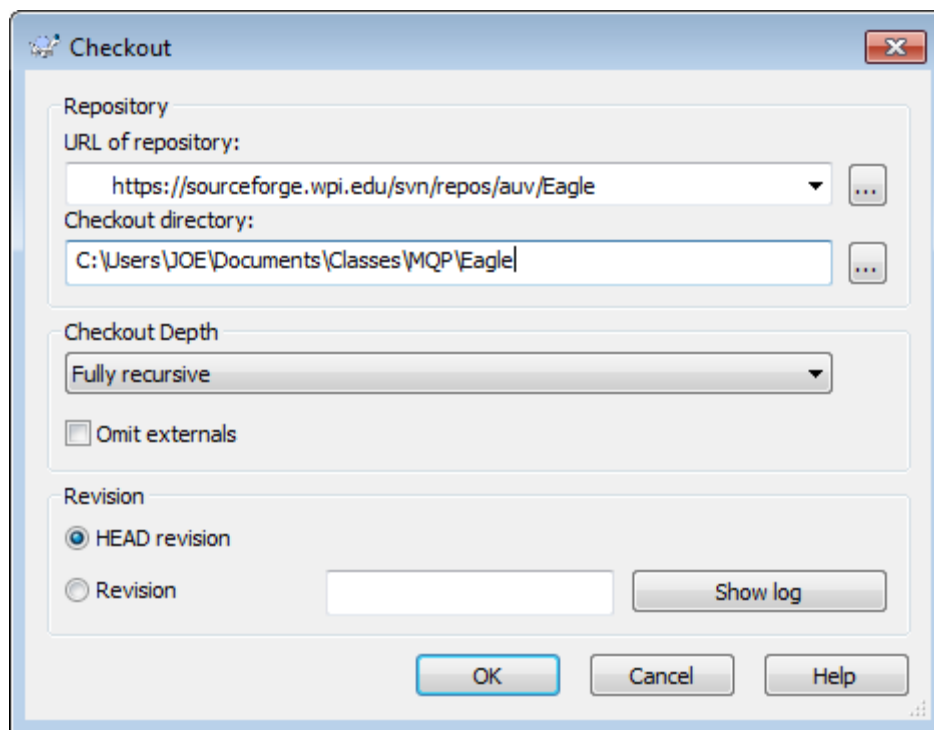


Figure 18: TortoiseSVN Checkout

## Eagle Setup

Download Eagle from <http://www.cadsoft.de/download.htm> and install it using the freeware license option when prompted. When Eagle is run, the Control Panel window is opened. As an initial configuration step, some directories in the files checked out from SVN have to be added to Eagle's directory paths. Add the Libraries, Design Rules, CAM Jobs, and Projects directories from SVN to the path as shown below.

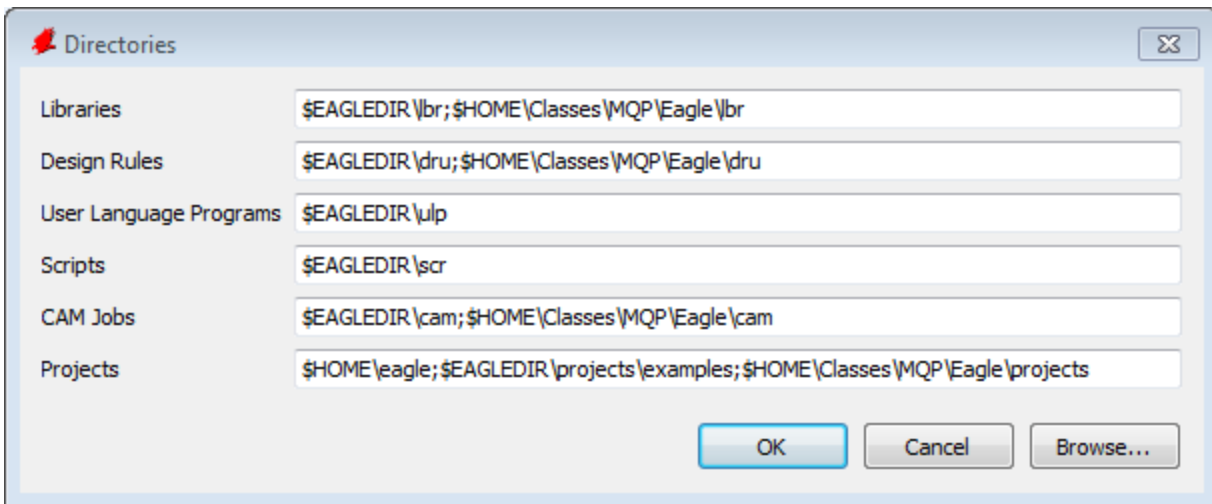


Figure 19: Eagle Configuration

At this point, read over the following tutorials as a primer:

- Eagle Schematics: <http://www.sparkfun.com/tutorials/108>
- Eagle PCB Layout: <http://www.sparkfun.com/tutorials/109>
- Creating parts in Eagle: <http://www.sparkfun.com/tutorials/110>
- Better PCBs in Eagle: <http://www.sparkfun.com/tutorials/115>

## PCB Fabrication

Once changes to the schematic/layout are finalized, it's time to create the files that need to be sent to the fabrication house. Gerber files contain a set of coordinates that directly control the hardware that creates the PCB. From the board layout window, open the CAM Processor, which will open in a new window. In the CAM Processor window, open the sfe-

gerb274x.cam in the 'cam' folder. This will add nine new tabs to the window for each layer. At this point, click the Process Job button in the bottom-right. The following 7 files need to be placed in a zip archive:

- daughtercard.gbl (Bottom copper)
- daughtercard.gbo (Bottom silkscreen)
- daughtercard.gbs (Bottom soldermask)
- daughtercard.gtl (Top copper)
- daughtercard.gto (Top silkscreen)
- daughtercard.gts (Top soldermask)
- daughtercard.txt (Drill file)

All previous PCBs have been fabricated by Advanced Circuits (<http://4pcb.com/>). Use their FreeDFM pre-order file check before placing an order. Once this has been cleared, place your order using the \$33 2-layer board special offer, and make use of the student discount to remove the minimum order size.

## Appendix B: Vehicle Operation Guide

This guide is based on the guide in last year's report with revisions made to reflect changes that have been made to the vehicle since the previous report was written.

### Vehicle Power and Battery Charging

The vehicle is powered by two 12V batteries hooked up in parallel. There is a power switch located near the aft battery that will allow the PCB and components to be connected without unwanted power.

Included in the CAN MUVE laboratory should be a 12V 3A constant current charger. The leads from the charger unit should be connected to the respective positive and negative terminals on a single battery pack in the AUV. Note: RED designates positive connections while BLACK is negative. Because both batteries are connected in parallel with each other, both batteries will charge simultaneously even though leads are connected to only a single pack. Total charge time will vary depending on usage. Leaving the AUV to charge for a full day however should ensure a complete charge of the system.

The charger uses a self-terminating trickle charge method to charge the batteries, so leaving the charger connected for long periods of time will not harm the system. The charger will automatically stop charging the batteries when they are at capacity and a small LED on the unit will turn green. A picture of the Soneil battery charger is included in Figure 20.



Figure 20: 12V 3A battery charger for the AUV batteries

## Programming

### MPLAB and C32 Compiler

MPLAB is Microchip's PIC-specific IDE. Although it should not be used in favor of Eclipse for development as it does not provide reliable SVN support, it needs to be installed because it includes the Microchip C32 compiler, associated libraries, and tools to download code to the PIC32. Microchip does offer the C32 compiler as a separate download, but it does not include the programming tools. At the time of writing, MPLAB X, a Netbeans-based replacement for MPLAB, is available as a beta release. In the future, MPLAB X will likely surpass Eclipse as the preferred development tool.

Do not use the software on the CD included with the PIC32 Starter Kit, as it is out of date and will cause AUVLib to fail to compile. MPLAB can be found on the Microchip website at [http://www.microchip.com/stellent/idcplg?IdcService=SS\\_GET\\_PAGE&nodeId=81](http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=81). The version of the C32 compiler used to write AUVLib (v1.11b Lite) can also be found on the Documents page on SourceForge if needed for compatibility reasons.

### Setting up Eclipse for AUV Development

The base Eclipse for C/C++ Developers installation can be acquired from <http://www.eclipse.org/downloads/>. If any sort of incompatibility issues are encountered with newer versions of Eclipse, all development for this project was done using Eclipse 3.5.

### PIC C Builder Plugin

This plugin allows Eclipse to use the Microchip C32 compiler. The .jar file, which can be obtained from <http://sourceforge.net/projects/piccbuilder/>, needs to be placed in the Eclipse\plugins directory.

### Subversive Plugin

Subversive provides access to SVN repositories from within Eclipse. This is the preferred method of using SVN when writing code in Eclipse, although TortoiseSVN will also work.

Subversive can be obtained from <http://www.eclipse.org/subversive/downloads.php>, with

detailed installation instructions located at

<http://www.eclipse.org/subversive/documentation/gettingStarted/aboutSubversive/install.php>.

### Eclox Plugin

Eclox is an Eclipse plugin to build Doxygen documentation such as the AUVLib documentation found in Appendix F. Eclox is available at

<http://home.gna.org/eclox/#download>.

### Checking out AUVLib from SVN

In Eclipse, click Window->Show View->Other... and select "SVN Repositories" from the SVN folder. There should now be a tab on the bottom pane for SVN Repositories. Add a new repository location with the URL <https://sourceforge.wpi.edu/svn/repos/auv> as shown below and click Finish.



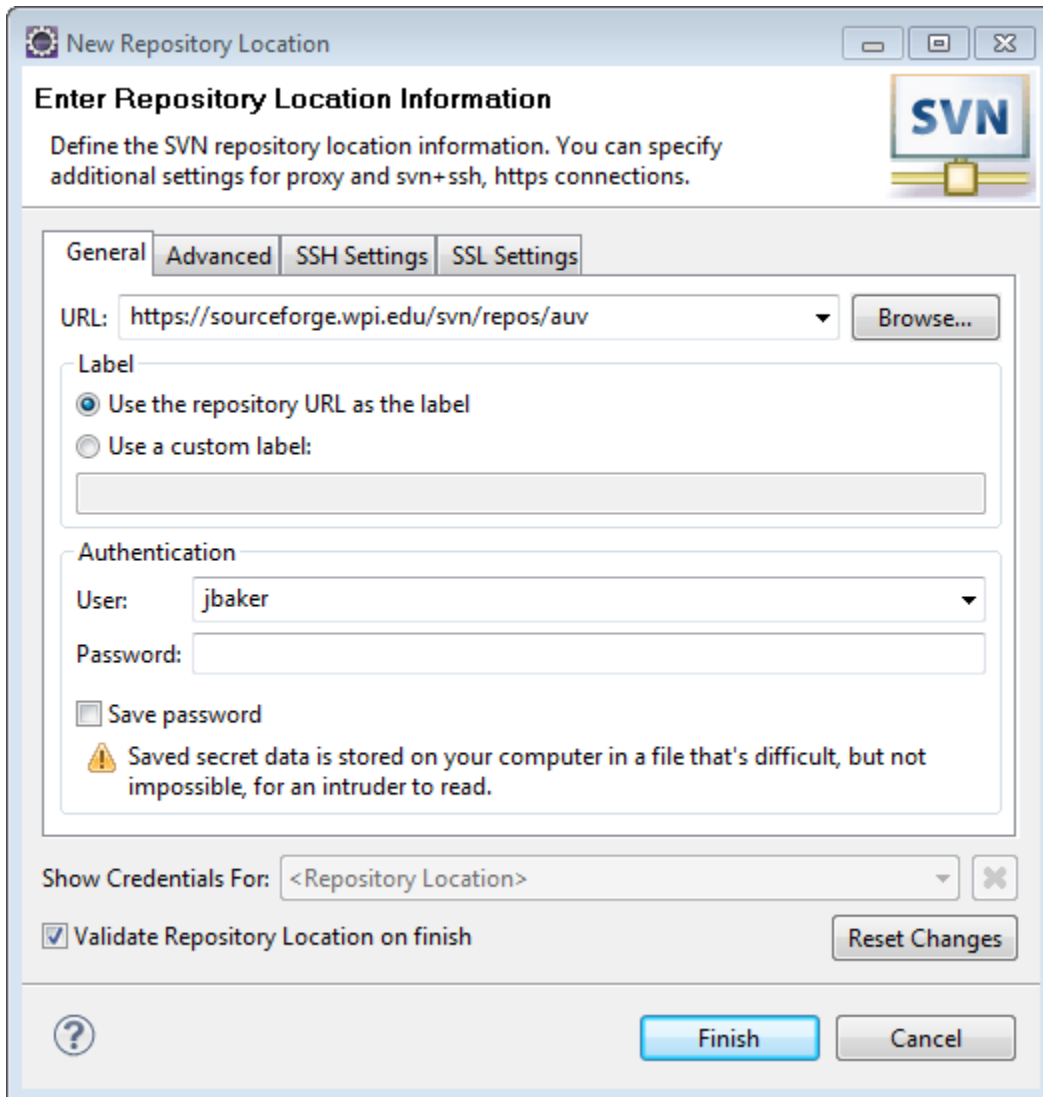


Figure 21: Adding SVN Repository to Eclipse

Expand the repository, right click on AUVLib, and click Check Out. The AUVLib project should now be in the Project Explorer tab on the left side of the Eclipse window.

## Making a new Project using AUVLib

Make a new C project, and select “PIC Cross Target Application” with the PIC C32 toolchain if shown below.

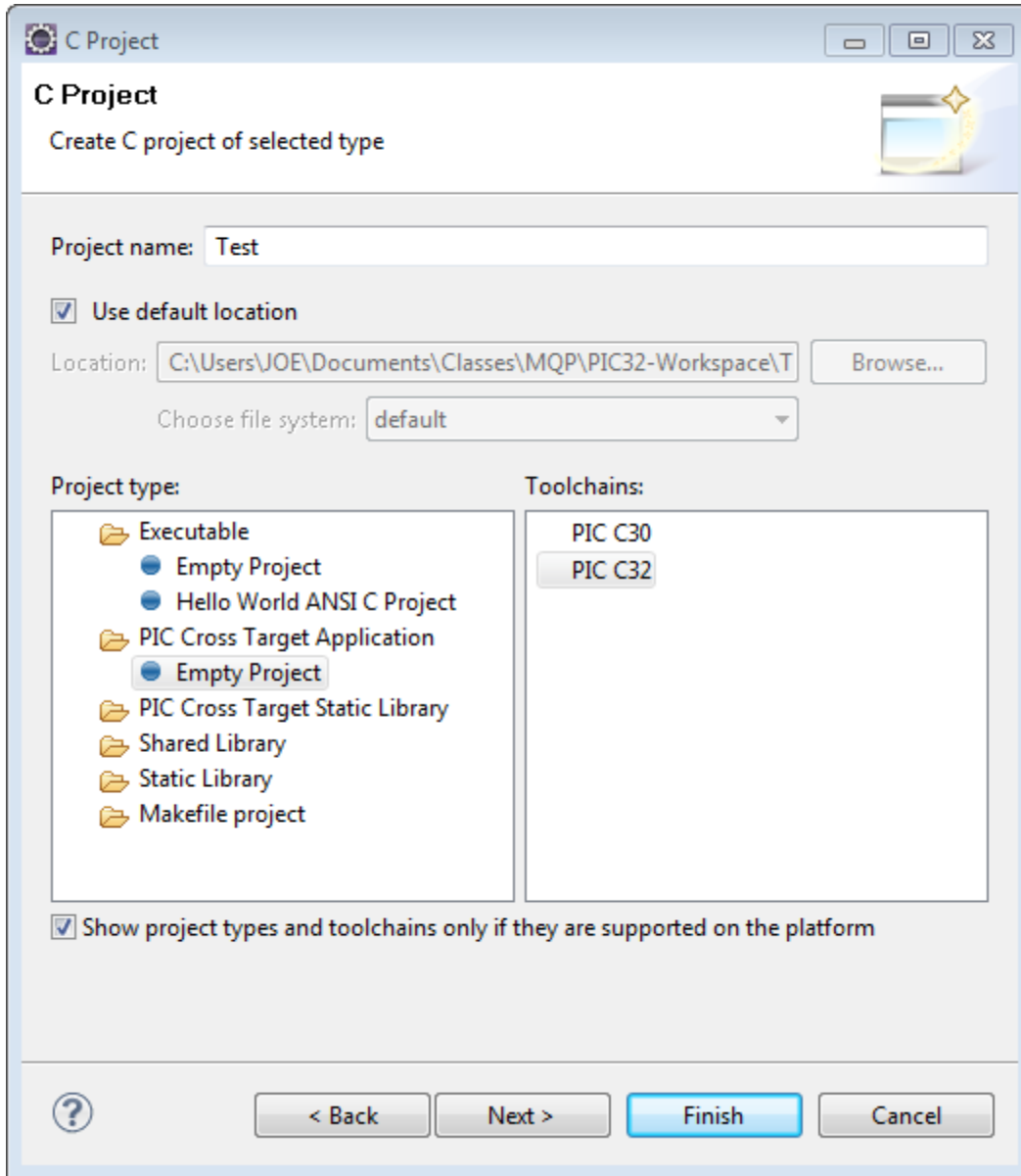


Figure 22: New Eclipse Project

At this point, click “Next”, not “Finish”. On the following screen, make sure Release is checked and Debug is unchecked. After clicking “Finish”, right click on the project folder in the Project Explorer tab on the left side of the screen and click “Properties”. Under C/C++ Build-

>Settings, set the target processor to PIC32MX360F512L. While still in C/C++Build->Settings, set up the PIC C32 C Compiler directories as shown for the PIC32 and AUVLib headers.

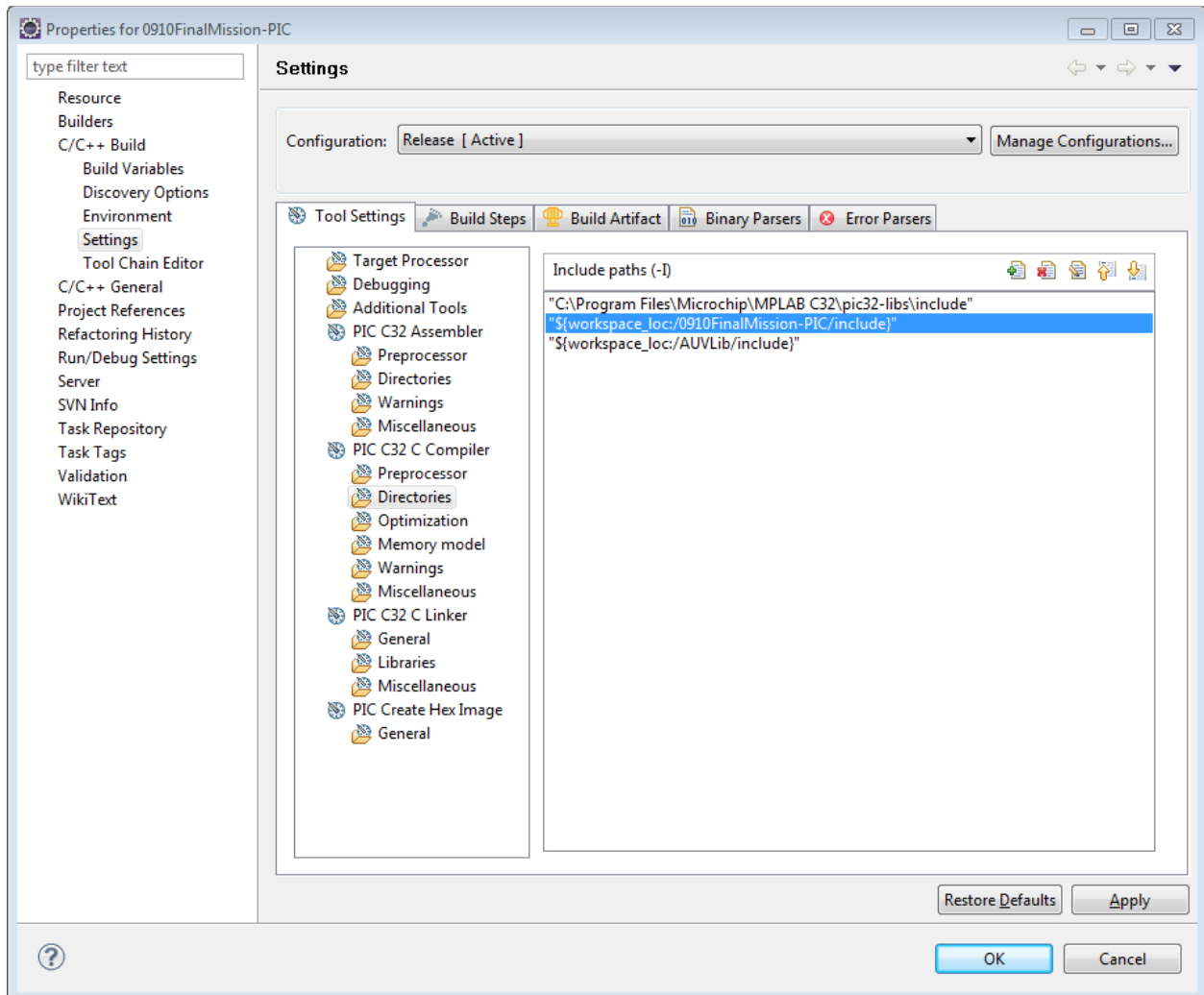
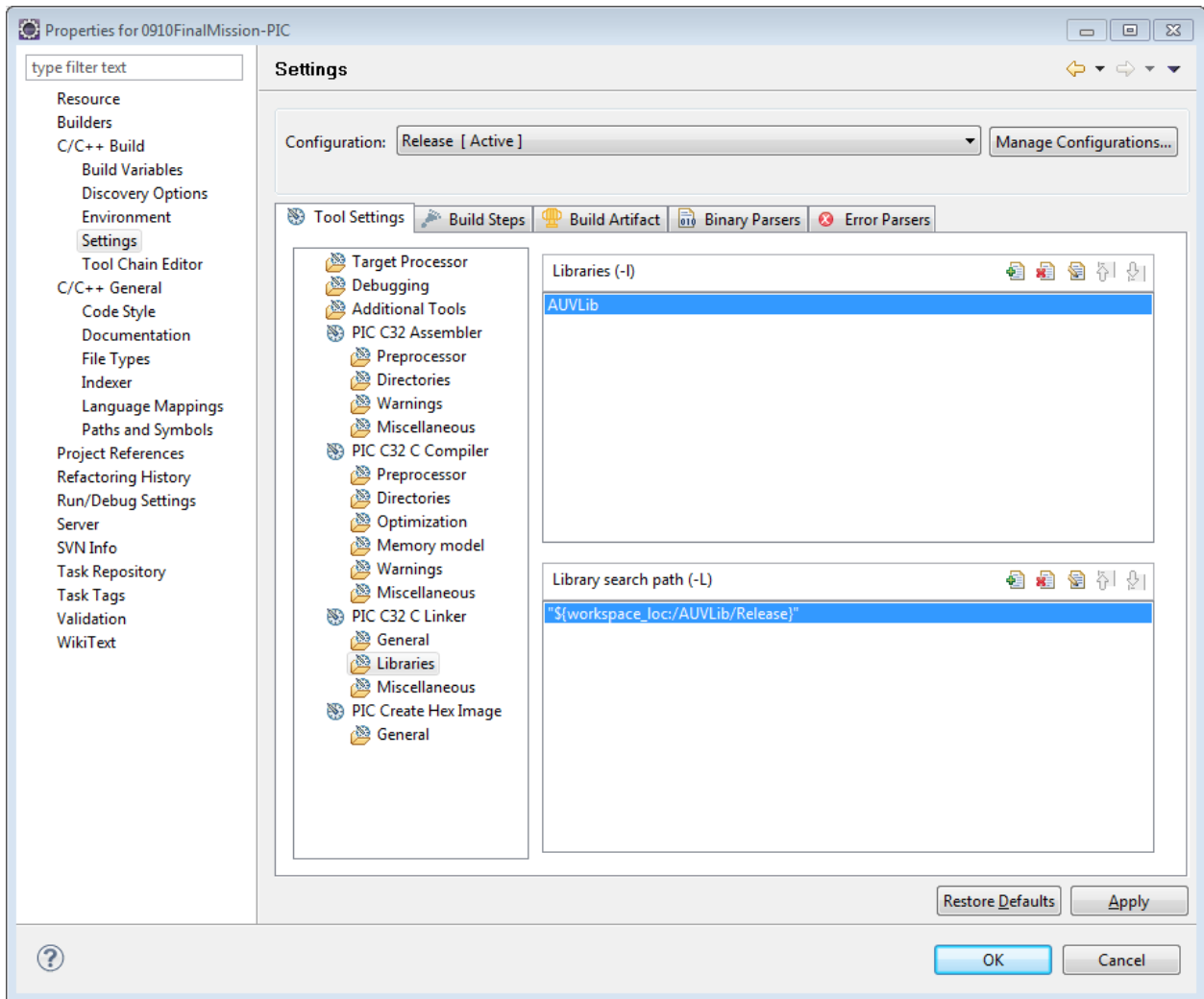


Figure 23: Eclipse Project Include Paths

Still in the Tool Settings tab of C/C++ Build->Settings, change the optimization level to "Optimize (-O1)", as higher levels of optimization are not available with the Lite version of C32. Configure the PIC C32 C Linker libraries as follows.



**Figure 24: Eclipse Project Library Paths**

To initially upload the new project on SourceForge, right click on the project folder, and click “Share Project...” in the Team menu. Once this is done, there are two commands in the Team menu which are most commonly used. The Update command updates the local copy of the files with any new changes since they were checked out/previously updated without overwriting changes made locally. The Commit command submits local changes to the server. It is generally recommended to run an update before committing, as conflicts can occur if others have committed changes of their own since you last updated.

## Downloading Code

### Warning

**DO NOT CONNECT THE USB CABLE TO THE PIC32 STARTER KIT WHILE IT IS CONNECTED TO THE REVISION 2.0 CONTROLLER. DOING SO WILL PERMANENTLY DAMAGE THE PROCESSOR.** The reason for this can be seen in the power regulation schematic from the PIC32 Starter Kit User's Guide below:

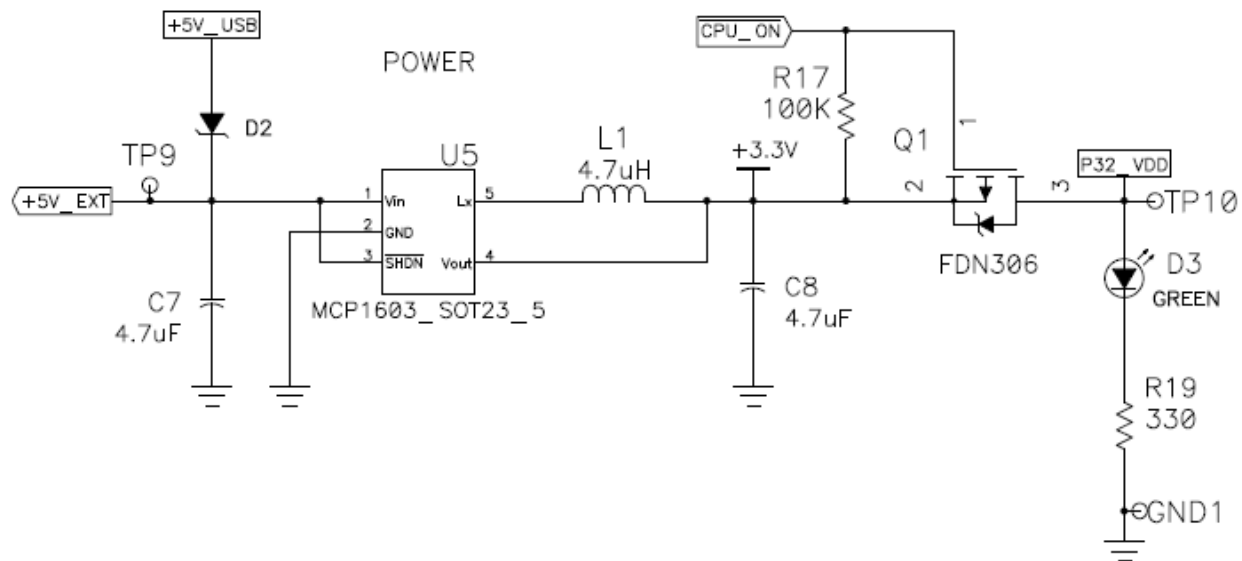


Figure 25: PIC32 Starter Kit power input schematic

Note that there is a reverse voltage protection diode on the +5V\_USB input (D2), but no such diode for the +5V\_EXT input. Connecting the USB input in this configuration will cause the entire daughtercard 5V rail (including the output pin of the 7805 regulator) to be connected to -5V. This mistake was made once (the batteries were not connected to the +12V input). The only observations made before realizing that something was wrong and disconnecting the USB cable were that the 7805 regulator and the PIC32 itself were very hot. After two failed attempts to solder a new PIC32MX360F512L to the Starter Kit board, the Starter Kit had to be replaced.

## Software Considerations

The first problem to deal with is that Microchip has not updated the USB drivers for the basic PIC32 Starter Kit for Windows 7. Instructions are available online for setting them up with Windows Vista, but Windows XP was used for this project.

Unfortunately, at this time there is no way to download code using Eclipse. The code has to be imported into MPLAB or MPLAB X to download. Despite being beta-quality software, MPLAB X was used over MPLAB because MPLAB was not recognizing the PIC32 Starter Kit. Differences in how MPLAB X handles header file directories compared to Eclipse require modification of the include paths in the source code. This situation may be avoidable with proper project configuration in the future. A separate project is available on SVN called AUVLib.X that shows the differences in configuration.

## Electrical Subsystem Connections

This process has been greatly simplified from the previous revision of the controller PCB. The actuator wires now have proper end connectors, so short circuits are not a serious concern. The silkscreen labels on the PCB show which actuator connects to which header without having to refer to this guide. Never connect a maneuvering jet actuator or the pump to a ballast solenoid header, as operation of the vehicle in this configuration will burn out the traces for the ballast solenoid MOSFET. Only two banana clip connections are required to power the controller. These 12V and ground inputs are also marked on the PCB silkscreen.

## Sealing the Vehicle

Sealing the vehicle is vital to its operation. A generous portion of petroleum jelly should be applied to the lower section of the hull where the “I” channel is attached. The jelly should be flush with the top of the “I” channel. When attaching the hull be sure not to damage any wires or shift any ballast weights. The top section will sit in the middle of the “I” channel and eight black rubber clamps around the vehicle should be secured. When the clamps are secured be sure to verify that the top of the hull is fully in the “I” channel and no gaps can be seen. Next it

is best to spread another layer of the petroleum jelly where the top of the hull and the “I” channel meet.

## **Transporting the Vehicle**

Transporting the vehicle seems simple but if not done with caution can lead to severe damage to various subsystems or the vehicle hull. A cart with wheels is used for transport. The vehicle sits in a white platform constructed from PVC piping. The vehicle should be strapped down first to this platform, using two green straps found in the lab, and then the platform should be strapped to the cart with a yellow strap also found in the lab.

When wheeling the vehicle on the cart at least two people should be involved. One person will push and steer the cart while the other helps avoid objects and prevent the vehicle from sliding if a strap is not secured correctly. When transporting the vehicle from the cart to the test site only the yellow strap should be removed. The white platform should remain attached to the vehicle and used to help carry the vehicle. Removing the vehicle from the platform is the last step in transporting the vehicle.

## Appendix C: Modem Operation Guide

### Modem Setup

The process to set up the modems is fairly straightforward. There are 2 hydrophones and one 12V power cable that must be plugged into each of the modems. The barrel jack connector for the 12V cable splits into two cables. The cable with a white strip in it is for +12V while the all black wire is ground.

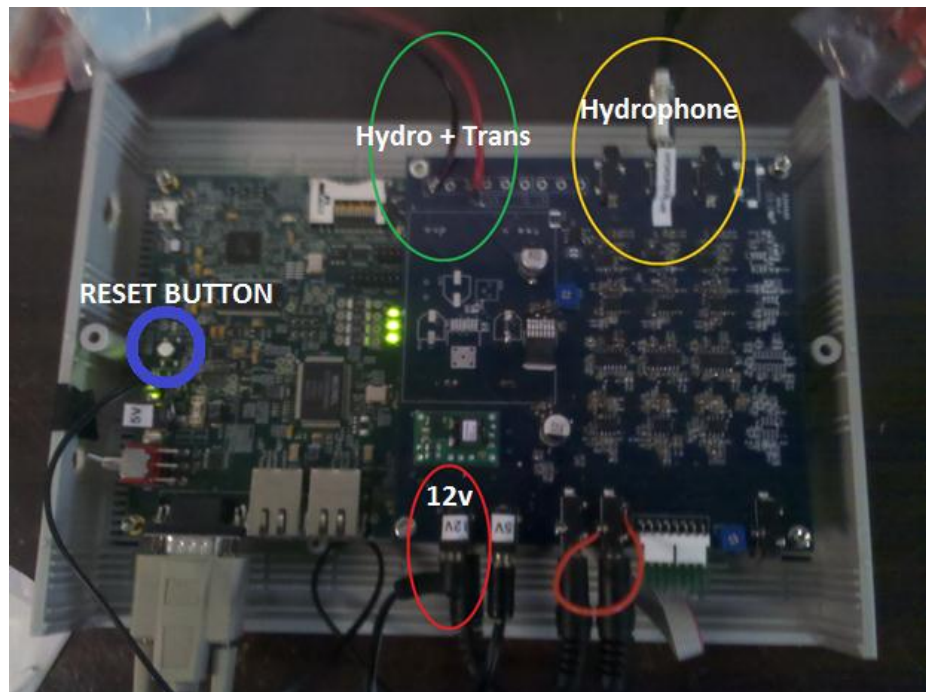


Figure 26: First modem

Currently, there are two hydrophones for each modem. One hydrophone for each modem will get plugged directly into the modem on the slot that reads “Hydrophone HY” (yellow in picture above). According to the UConn group, there is a possibility in the future of a third hydrophone used as a second receiver. Each modem is slightly different in terms of where the hydrophones are connected. The second modem (shown below) has a different hydrophone location (also shown in yellow) but the 12V jack and the second hydrophone are located in the same location.



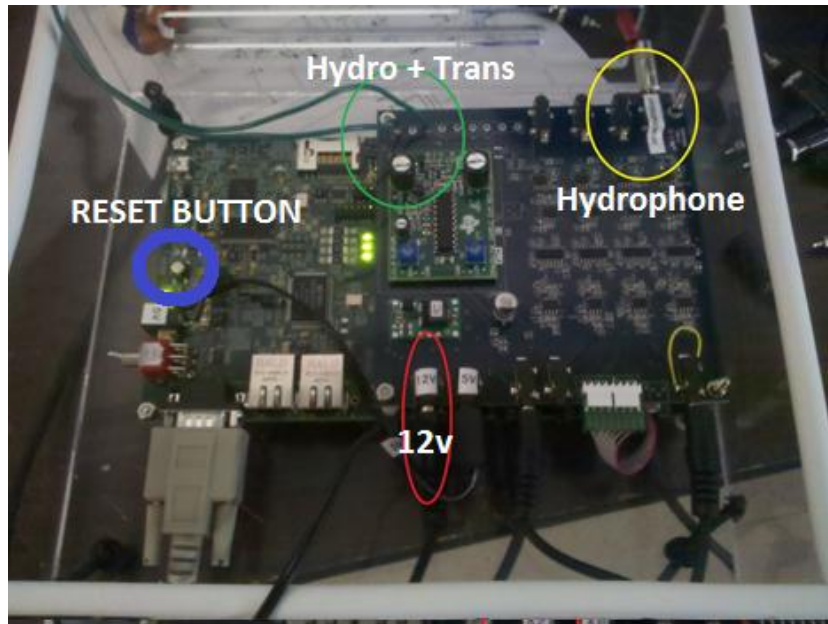


Figure 27: Second modem

The second hydrophone from each modem must be plugged first into a transducer and then into the modem. There are currently 2 transducers in the lab. These are shown below. Only 2 hydrophones have the correct physical connection to plug into the transducers. Once the hydrophones are plugged into the transducers they are then plugged into the modems on the location shown by green circles in the pictures above. Again this connection is different than the first set of hydrophones so there should be little to no confusion.



Figure 28: Modem transducers

The last picture below shows an example of a pair of hydrophones to be plugged into a modem. There are 2 sets of cables so they should be split up to match the pair in the picture below.

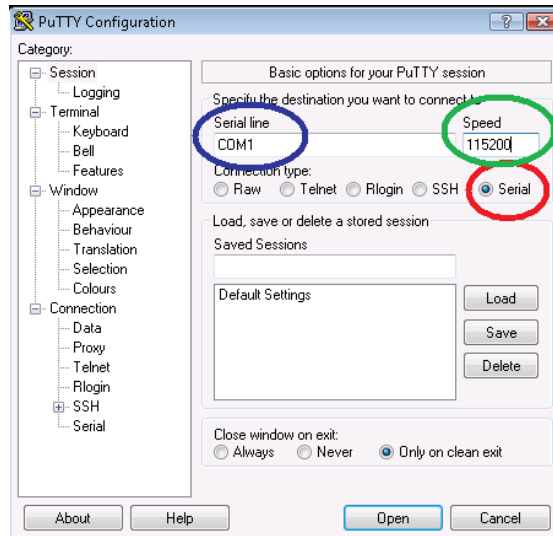


**Figure 29: Hydrophones**

Lastly plug the serial cable from the modem into either the daughter card, for the sub modem, or a computer for a command modem. It is important that the cables used are Null Modem cables and not standard serial cables. The modems will not work otherwise.

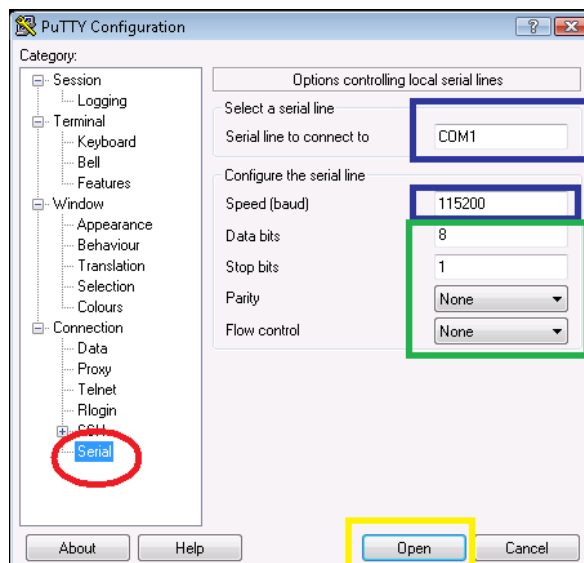
### **PuTTY Setup**

In order to send and receive serial data from the modems one must use a terminal program. Because Putty is available on almost every PC on campus, it was chosen as the terminal program of choice. Firstly you need to select the serial radial button from the first menu (shown in red below). Next make sure correct COM port is selected (Blue in picture below). Lastly change the baud rate, or speed, to 115200 (green in picture below).



**Figure 30: PuTTY Configuration**

Now select the serial sub menu from the list of option along the left side of the window (red in picture below). The COM and speed (blue in picture below) should be what was selected in the first menu. If there is a difference make sure the settings are correct before proceeding. The most important part in this menu is selecting the proper data bits, stop bits, parity and flow control options. The options shown below in green should always be used when working with the modems. Lastly select open to open the serial connection window.



Data = 8  
 Stop = 1  
 Parity = None  
 Flow Control = None

**Figure 31: PuTTY Serial Configuration**

## Modem Modes of Operation

Once the modems are set up there are two ways in which they can be operated. The first way is an auto-ping configuration. In this configuration the modems will ping each other every 16 seconds. While the modem will ping each other every 16 second a user can also send a message in the mode as one normally would. Secondly the modems can be set to only send data when prompted by the user. To change between these configurations a switch must be toggled on the modem. The switch in question is located in a bank of 4 switches shown below. Switches 0-2 should be left alone, while switch 3 changes which mode the modems are currently in. In the picture shown below the modem is currently in “send on user command” mode. The mode will toggled to auto-ping mode by switching the position of switch 3 as explained above.

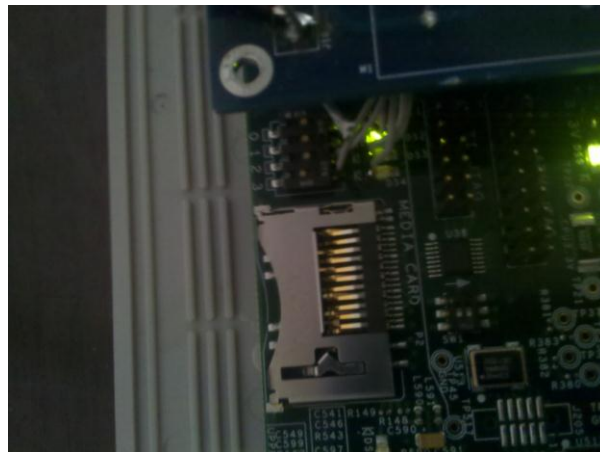
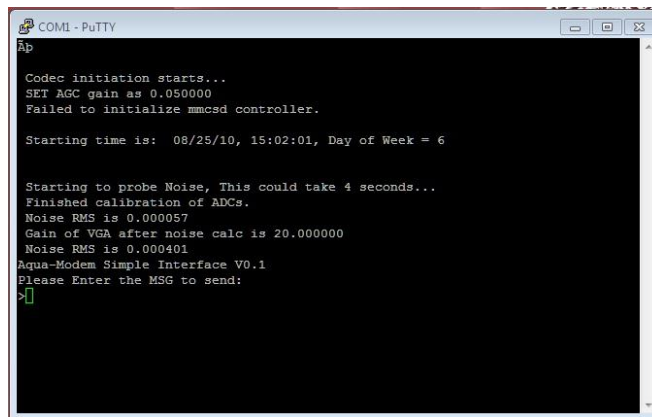


Figure 32: Modem Switches

## Sending and Receiving Commands

Once the modems are setup and the Putty terminal windows have been opened, turn on the power to the modems via a power supply or through the AUVs battery. The picture below shows what the modems should print to the terminal window after being powered. If this message is not shown check all previous steps and make sure there are lit green LEDs on the modems (signaling they are getting power).



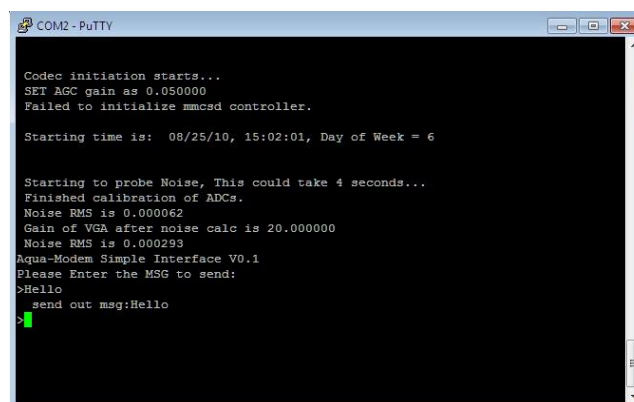
```
COM1 - PuTTY
Ap
Codec initiation starts...
SET AGC gain as 0.050000
Failed to initialize mmcsd controller.

Starting time is: 08/25/10, 15:02:01, Day of Week = 6

Starting to probe Noise, This could take 4 seconds...
Finished calibration of ADCs.
Noise RMS is 0.000057
Gain of VGA after noise calc is 20.000000
Noise RMS is 0.000401
Aqua-Modem Simple Interface V0.1
Please Enter the MSG to send:
>
```

Figure 33: Modem startup

Sending messages is fairly straightforward. Simply type the message that is to be sent in the window corresponding to the transmitting modem and hit enter. The modem will reply echoing what message is being sent by returning “send out msg: message” where message is the message that has been sent (in the figure below the message sent is “Hello”). The follow figure shows what should happen on the sending window when attempting to transmit.



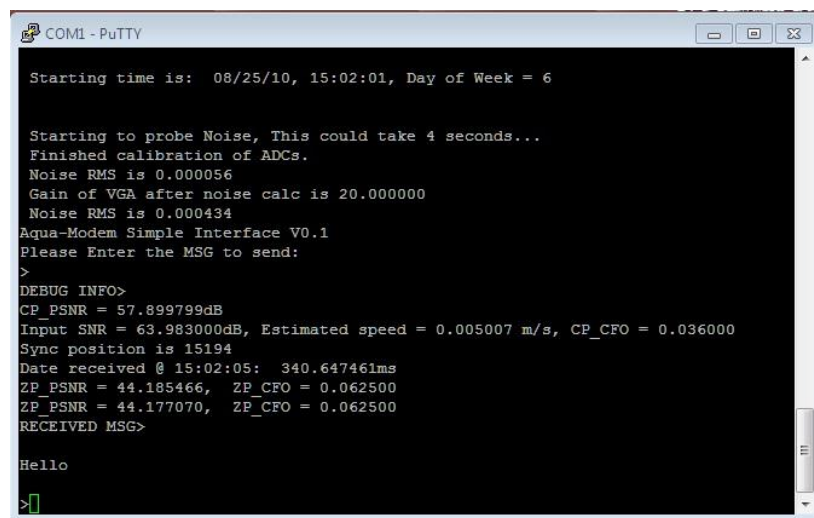
```
COM2 - PuTTY
Codec initiation starts...
SET AGC gain as 0.050000
Failed to initialize mmcsd controller.

Starting time is: 08/25/10, 15:02:01, Day of Week = 6

Starting to probe Noise, This could take 4 seconds...
Finished calibration of ADCs.
Noise RMS is 0.000062
Gain of VGA after noise calc is 20.000000
Noise RMS is 0.000293
Aqua-Modem Simple Interface V0.1
Please Enter the MSG to send:
>Hello
send out msg:Hello
>
```

Figure 34: Modem transmission

After the message has been transmitted an audible squeal can be heard from the acoustic modems. This squeal should be followed by an update to the receiving modems terminal window. The figure below shows the terminal window of the receiving modem after getting the message "Hello" from the previous step. Currently, there is quite a bit of debug information that also gets printed out with the received message. This is normal, and future upgrades to the modems firmware should remove the debug information for less computation time on the AUV.



```
COM1 - PuTTY
Starting time is: 08/25/10, 15:02:01, Day of Week = 6

Starting to probe Noise, This could take 4 seconds...
Finished calibration of ADCs.
Noise RMS is 0.000056
Gain of VGA after noise calc is 20.000000
Noise RMS is 0.000434
Aqua-Modem Simple Interface V0.1
Please Enter the MSG to send:
>
DEBUG INFO>
CP_PSNR = 57.899799dB
Input SNR = 63.983000dB, Estimated speed = 0.005007 m/s, CP_CFO = 0.036000
Sync position is 15194
Date received @ 15:02:05: 340.647461ms
ZP_PSNR = 44.185466, ZP_CFO = 0.062500
ZP_PSNR = 44.177070, ZP_CFO = 0.062500
RECEIVED MSG>

Hello
>
```

Figure 35: Modem Reception

From here repeat the previous steps to send any signal from one modem to another. As long as the modems have 2 hydrophone connected each should be able to send and receive. There is no need to reset the modems for each message; simply type in a new message on either screen and press Enter.

## Troubleshooting

Should there be any issues with the modem make sure to go over each step again, double checking all connections. Another good strategy for troubleshooting is to use a voltmeter to check that the power supply used for 12v is actually outputting 12v. Finally if there are still issues with the modems contact Janny Liao at UConn. Please request Professor Hussein for contact information for Janny Liao.

## Appendix D: MATLAB Code for Motion Simulations

### Simple Motions

```
function simp_motion(mode,delta_x)
% Mode values
%1 - x
%2 - yaw
%3 - z
%4 - pitch

%% Initialize
a1=0;

switch(mode)
    case 1
        X=26;
        m=72;
        a1=26/m;
        lab='X [m]';
    case 2
        X=26;
        r=0.201;
        Iz=4.6;

        a1=X*r/Iz;
        lab='Yaw [radians]';
    case 3
        a1 = 18*.61/72;
        lab='Pitch [radians]';
    case 4
        a1 = 18*.1/.526;
        lab='Roll [radians]';
end

%% Solve
if(delta_x>0)
    t_run=sqrt((delta_x/2)*2/a1);
else
    t_run=sqrt((-delta_x/2)*2/a1);
    a1=-a1;
end

steps=1000;
dt=(t_run)/steps;
t=0:dt:t_run;

xs1=zeros(length(t),2);
xs2=zeros(length(t),2);

for i=1:length(t)
```

```

    %Phase 1
    x=a1*t(i)^2/2;
    xs1(i,:)=[t(i) x];

    %Phase 2
    x=-a1*t(i)^2/2+a1*t_run*t(i)+a1*t_run^2/2;
    xs2(i,:)=[t(i)+t_run x];
end

fprintf('Half runtime: %0.6g\n',t_run)

plot(xs1(:,1),xs1(:,2))
hold on
plot(xs2(:,1),xs2(:,2),'g')
legend('Phase 1','Phase2')
xlabel('Time [s]')
ylabel('lab')

```

## Vertical Maneuver

### Ballast

```

function [tb] = Ballast(h_des,ho)

global mo P2o V2o rho empty A Af g Cd V

%% Define Contant variables
% mo=65;
P2o=172368.932; %25 psi
% V2o=1*10^-003;
V2o=0.007296588;
rho=998;
empty=0;

A=0.03^2*pi/4;
g=9.81;
Cd=0.6;
Af=0.213*1.22;
V=0.408*0.213*1.2;
mb0=V2o*rho;

mo=rho*V;

%% initiate progress bar
Pbar=waitbar(0,'0% complete','Name','Simulating AUV Ballast System...');
%% Initialize State Vector
s0=zeros(3,1);
s0(1)=0;
s0(2)=ho;
s0(3)=0;

```



```

%% Simulation time
t0=0;
t_max=40;
dt=0.001;
tstep=0.05;
tspan=0:dt:tstep;

%% Initizlize Matricies
tb=zeros(4,2);

stage=1;
down=0;

%% run simulation 2
s_count=1;
figure
for count = 0:(t_max/tstep)
    t=t0+count*tstep;

    %solve ODE
    [t_ode,s_ode] = ode45('Ballast_ODE',tspan,s0);

    s_size=size(s_ode);

    %set new initial conditions
    t=t+tstep;
    s0=s_ode(s_size(1),1:3)';

    %add points to matrix
    tc(s_count:s_count+s_size(1)-1)=t+t_ode;
    sc(s_count:s_count+s_size(1)-1,:)=s_ode(1:s_size(1),:);
    s_count=s_count+s_size(1);

    %controller
    mb=s0(1);
    h=s0(2);
    vel=s0(3);
    delta=h-h_des;

%     empty=-2;

    switch(stage)
        case 1
            empty=-1;
            stage=2;
            tb(1,:)=[t empty];
        case 2
            if(mb>=mb0*0.57)
                empty=0;
                tb(2,:)=[t empty];
            end
    end
end

```

```

    if(delta>=0)
        stage=3

        subplot(3,1,1)
        hold on
        plot(tc,sc(:,1))

        subplot(3,1,2)
        hold on
        plot(tc,sc(:,2))

        subplot(3,1,3)
        hold on
        plot(tc,sc(:,3))
        clear tc sc
        s_count=1;
    end

case 3
    empty=1;
    if(mb<=0.02*mb0)
        empty=0;
        tb(3,:)=[t empty];
        stage=4

        subplot(3,1,1)
        plot(tc,sc(:,1),'g')

        subplot(3,1,2)
        plot(tc,sc(:,2),'g')

        subplot(3,1,3)
        plot(tc,sc(:,3),'g')
        clear tc sc
        s_count=1;
    end

case 4
    empty=-2;
    tb(4,:)=[t empty];
    stage=5

case 5
    if(vel<0)
        empty=0;
        stage=6
        subplot(3,1,1)
        plot(tc,sc(:,1),'r')

        subplot(3,1,2)
        plot(tc,sc(:,2),'r')

        subplot(3,1,3)
        plot(tc,sc(:,3),'r')
        clear tc sc
    end

```

```

        s_count=1;
    end

    case 6
        break
    end

    %update progres bar
    ratio=count/(t_max/tstep);
    waitbar(ratio,Pbar,sprintf('%6.2f%% Complete',100*ratio));

end

% figure
subplot(3,1,1)
% plot(tc,sc(:,1))
title('Balast Mass')
xlabel('Time [s]')
ylabel('Mass [kg]')
legend('Phase 1','Phase 2','Phase 3')

subplot(3,1,2)
% plot(tc,sc(:,2))
title('Sub Position')
xlabel('Time [s]')
ylabel('Depth [m]')
legend('Phase 1','Phase 2','Phase 3')

subplot(3,1,3)
% plot(tc,sc(:,3))
title('Sub Velocity')
xlabel('Time [s]')
ylabel('Velocity [m/s]')
legend('Phase 1','Phase 2','Phase 3')

%% Close Progress bar
close(Pbar);

```

## Ballast ODE

```

function sdot = Ballast_ODE(t,s)

global mo P2o V2o rho empty A Af g Cd V

mb = s(1);
h = s(2);
v = s(3);

```

```

m = mb+mo;
P2 = P2o*V2o/(V2o-mb/rho);
hddot=0;

mdot=0;
if(empty==1)
    P1=101325+rho*g*h;
    mdot = -A*sqrt(2*rho*(-P1+P2));
    if(P1>P2)
        mdot = A*sqrt(2*rho*(P1-P2))
    end

    if(mb<=0)
        mdot=0;
    end
elseif(empty==-1)
    P1=448159.224;
    mdot = A*sqrt(2*rho*(P1-P2));

    if(P1<P2)
        mdot = -A*sqrt(2*rho*(-P1+P2));
    end

elseif(empty==-2)
    hddot=-36/m;
end

mdot_max=1.13562354*rho/3600;

if(mdot>mdot_max)
    mdot=mdot_max;
end
% P1=101325+rho*g*h;
% mdot = -A*sqrt(2*rho*(-P1+P2));

hddot = hddot + g-1/2*rho*v^2*Cd*Af*sign(v)/m-rho*V*g/m;
hdot = v;

sdot = zeros(3,1);
sdot(1) = mdot;
sdot(2) = hdot;
sdot(3) = hddot;

```

## Dive Maneuver Lookup Table Generator

This script uses a slightly modified version of the Vertical Maneuver code which disables the progress bars, console output, and plot generation. If this is reused, be sure to remove the commas directly before each closing bracket and after the final closing bracket.

```

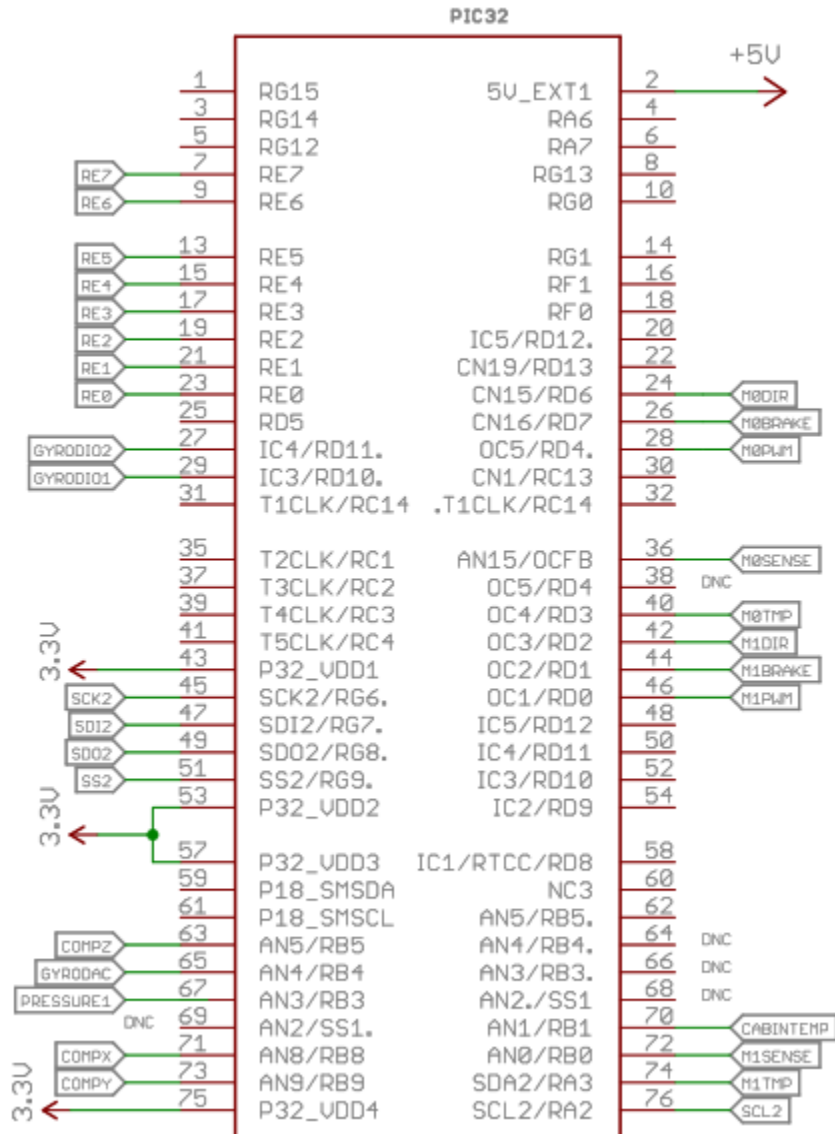
fid = fopen('ballast_table.txt', 'w');
progbar=waitbar(0, '0% complete', 'Name', 'Simulating AUV Ballast System...');
for row = 0:0.1:4.0
    progratio = row/4.0;
    waitbar(progratio,progbar,sprintf('%6.2f%% Complete',100*progratio));
    fprintf(fid, '//h_0 = %u dm \n{', row*10);
    for column = 0:0.1:4.0
        fprintf(fid, 'Row: %u, Col: %u\n', row*10, column*10);
        bal = Ballast(column,row);
        out = round(bal(3,1)*1000);
        fprintf(fid, '%5.0d,', out);
        fprintf(fid, '%6.3f\n', out);
    end
    fprintf(fid, '},\n');
end
fclose(fid);

close(progbar);

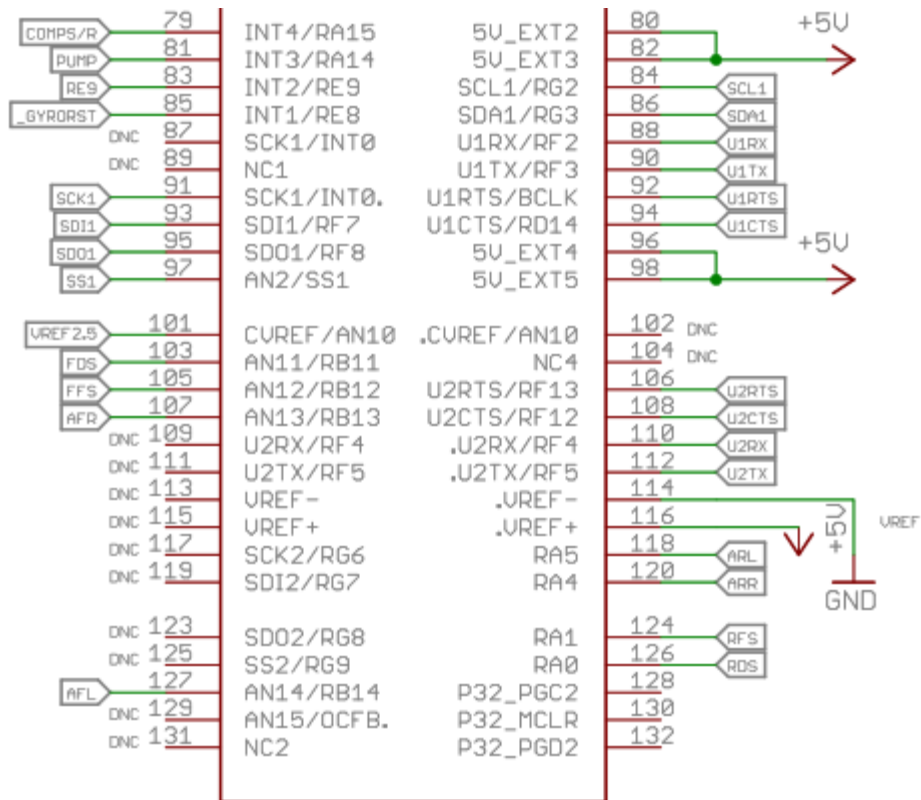
```

## Appendix E: Controller Revision 2.0 Schematics

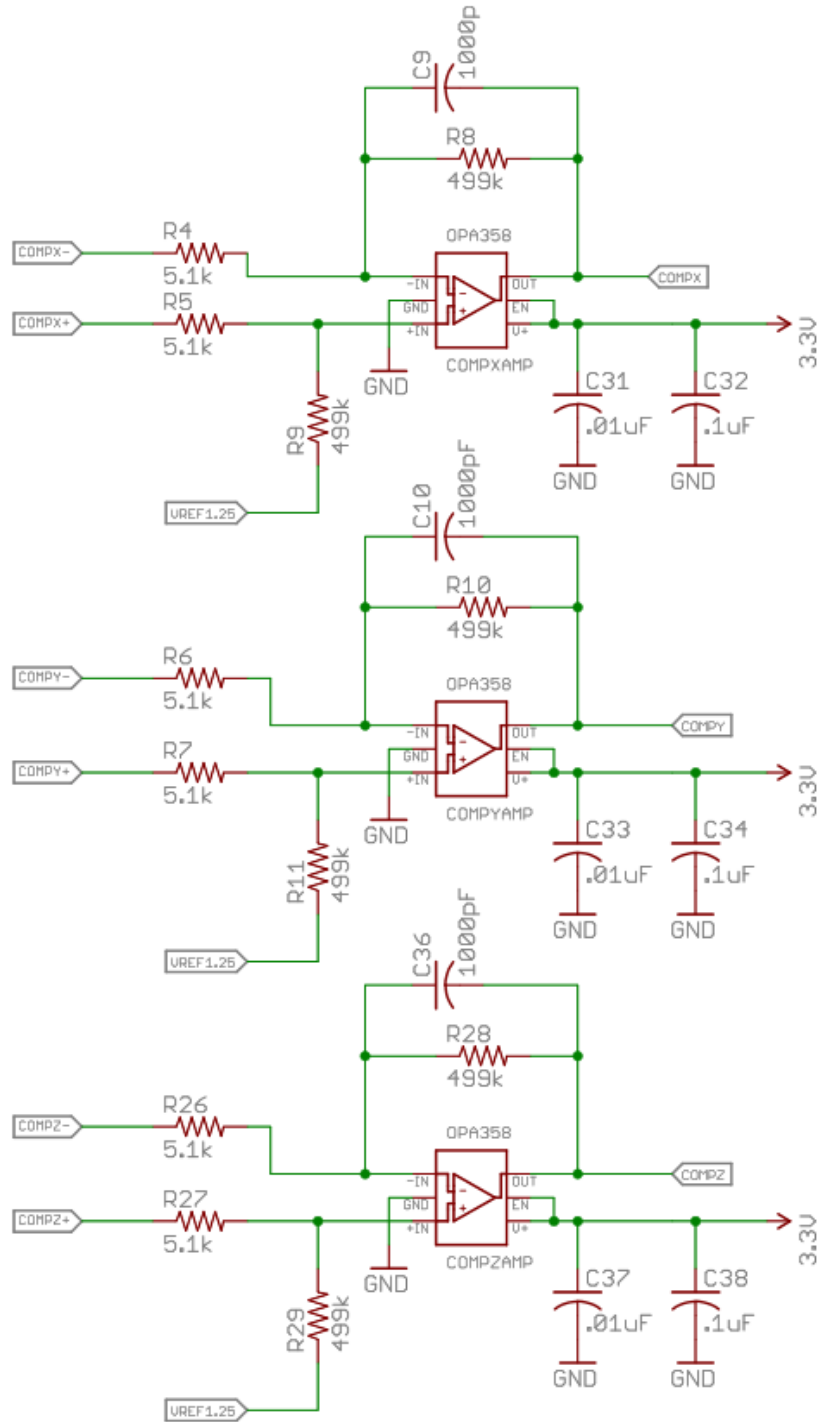
### PIC32 Connector (upper half)



## PIC32 Connector (lower half)

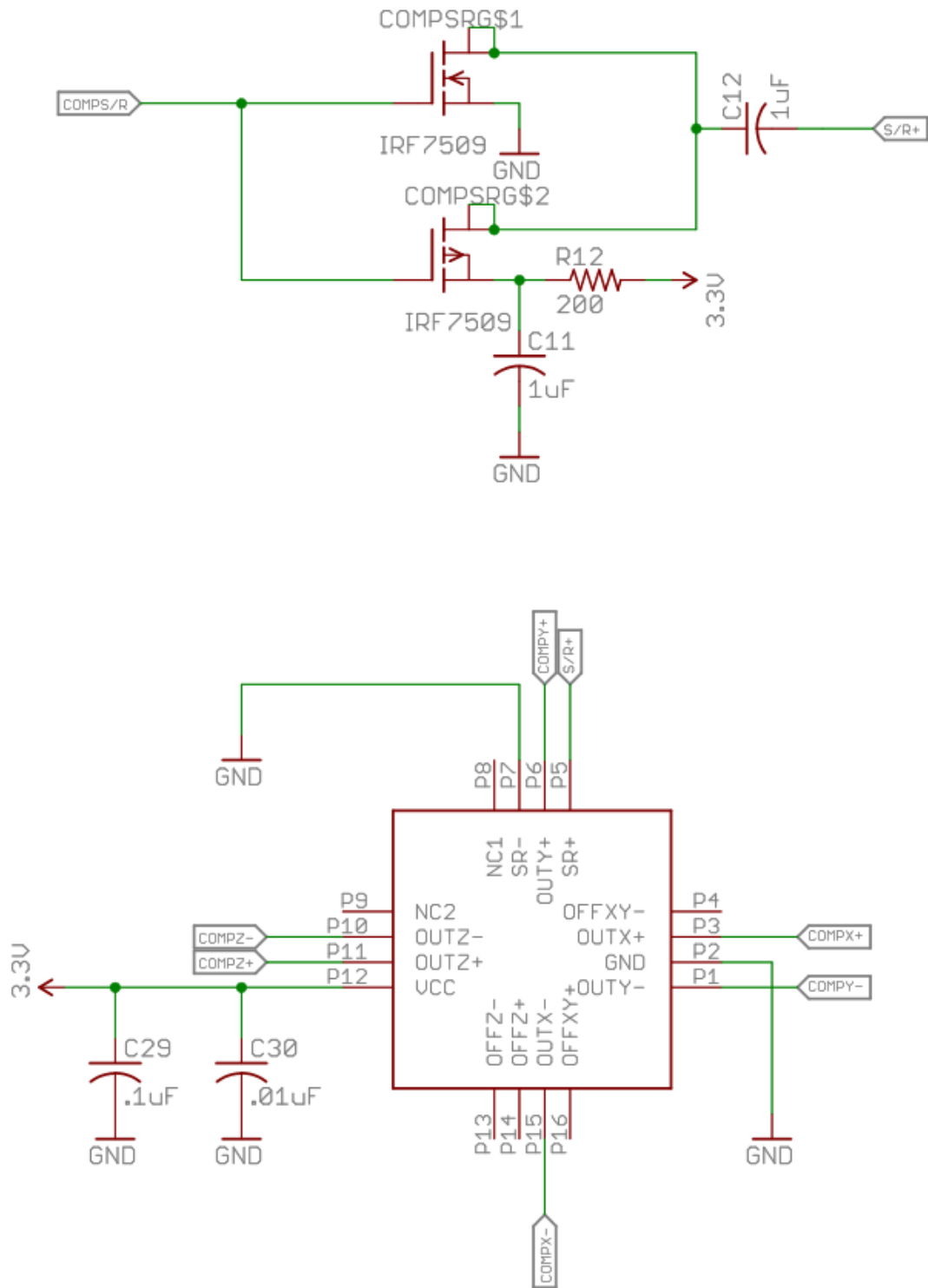


# Compass Amplifiers

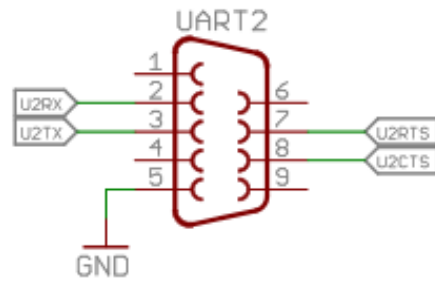
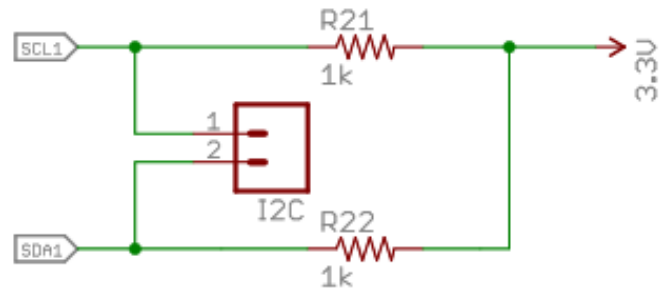
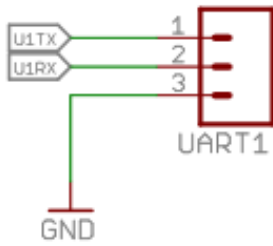
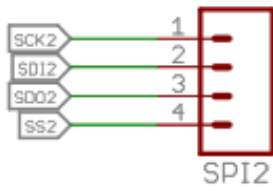
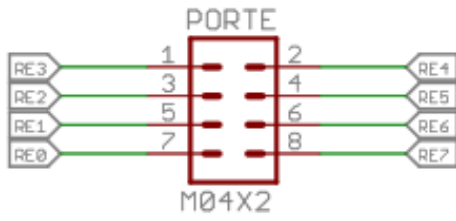




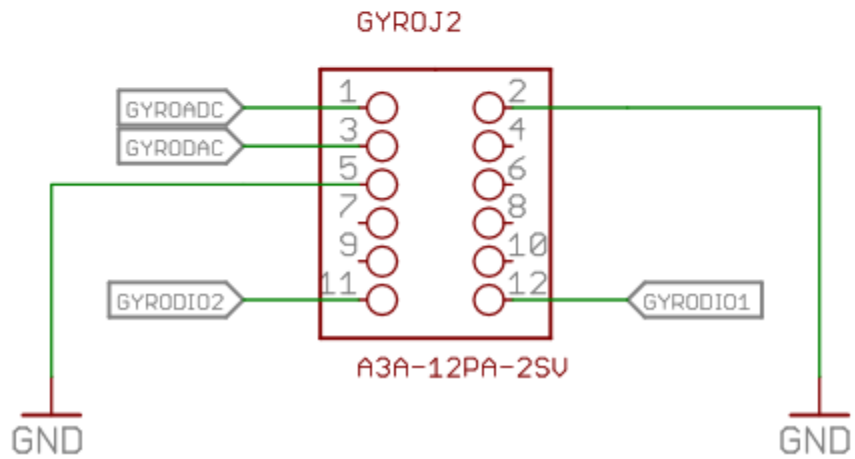
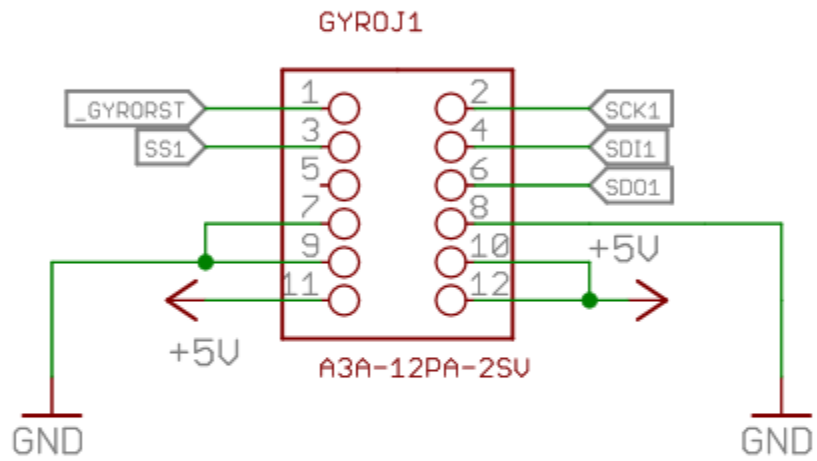
## Compass and Compass Set/Reset Circuit



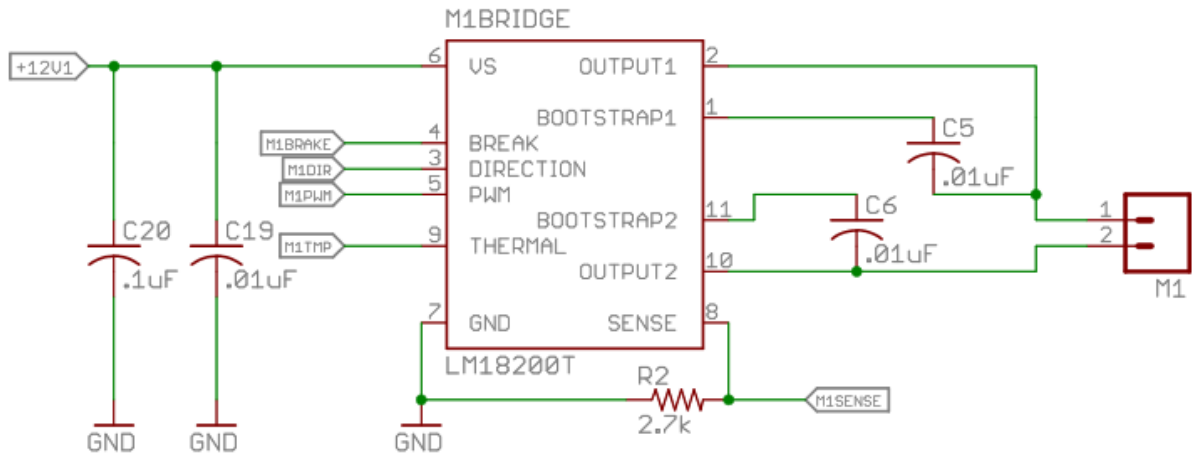
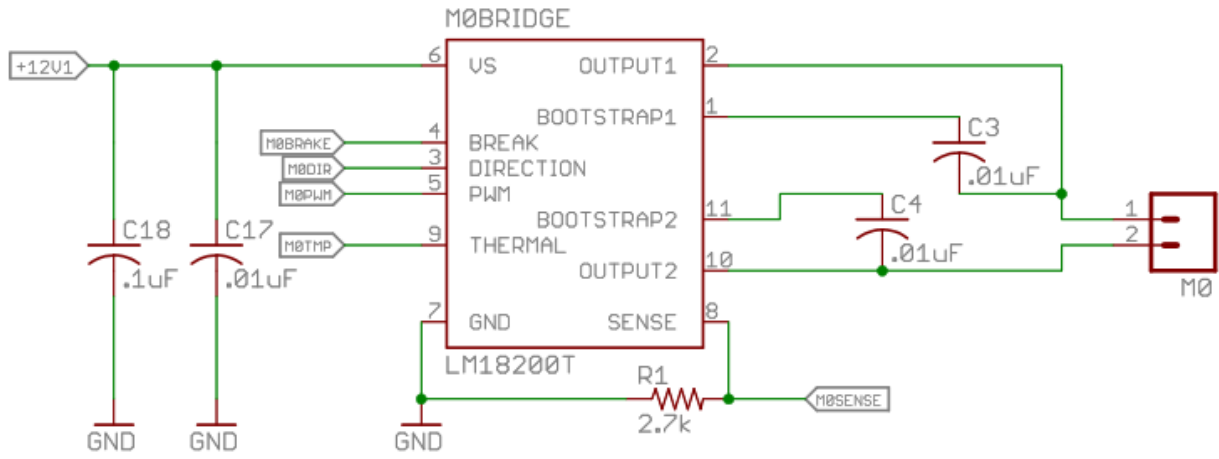
## External Communications



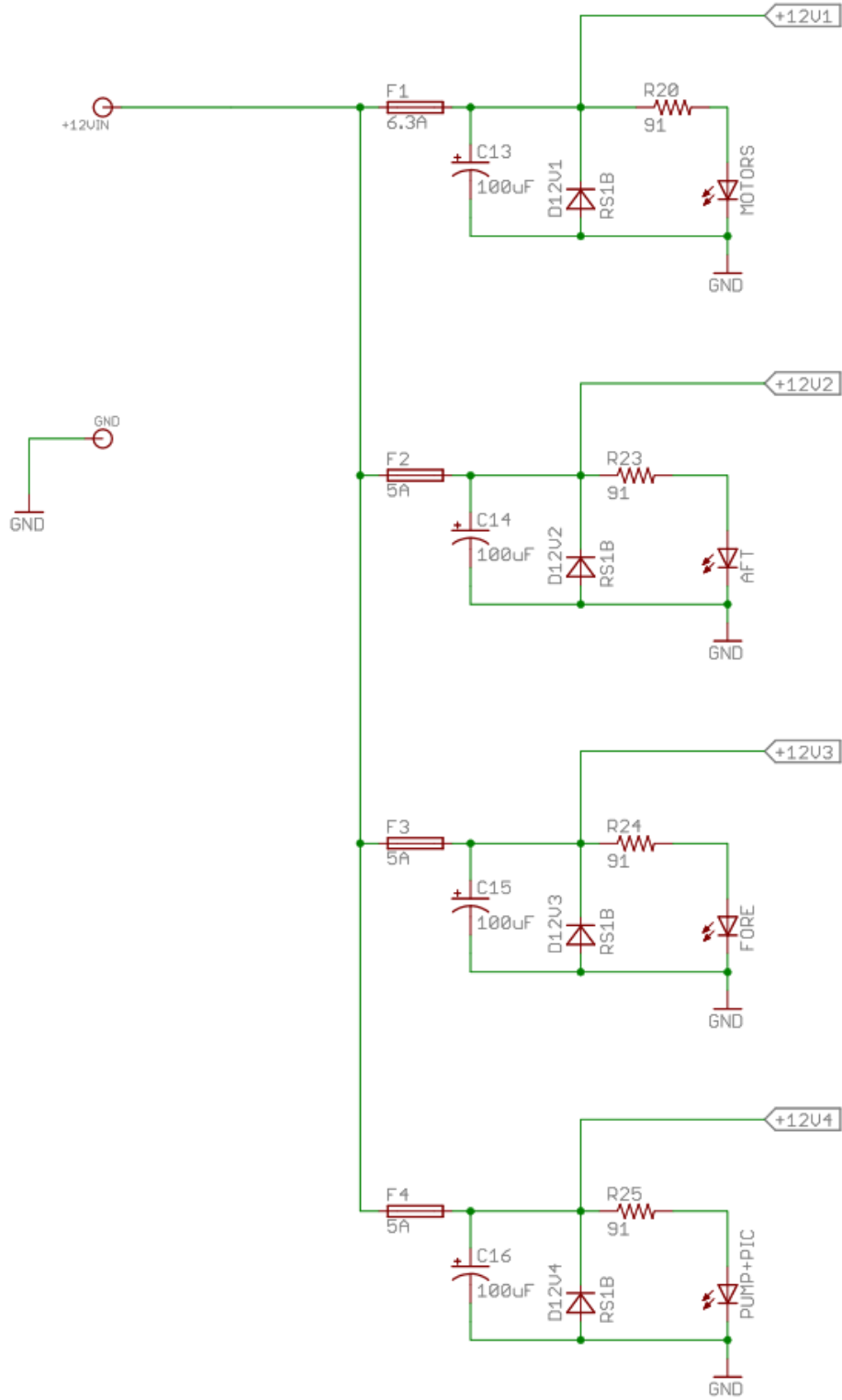
# Gyro



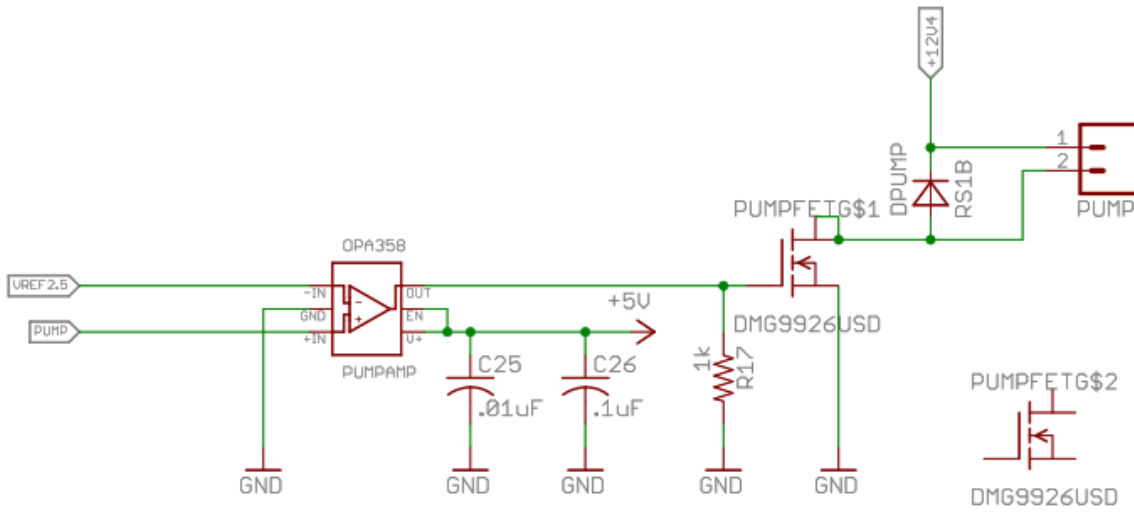
## Motors



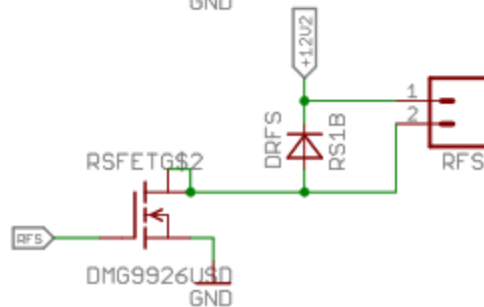
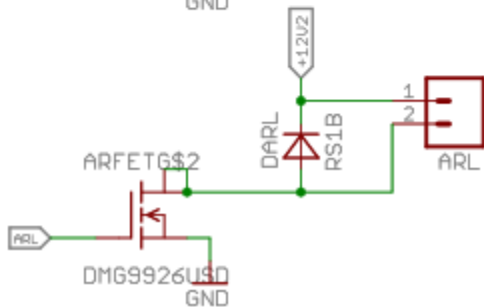
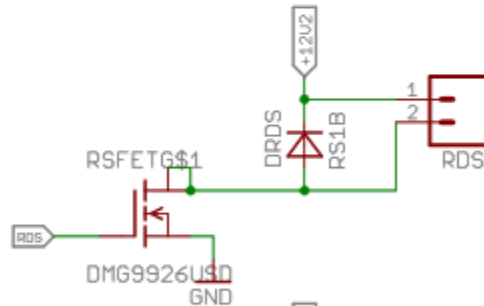
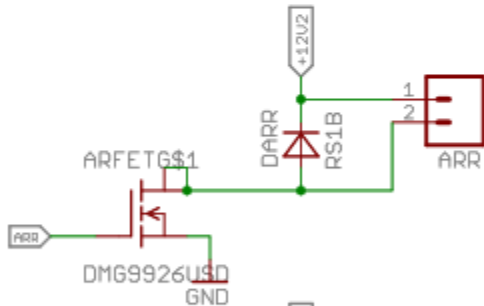
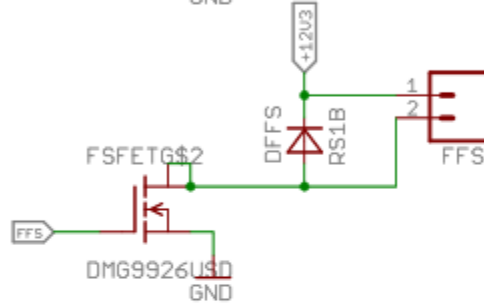
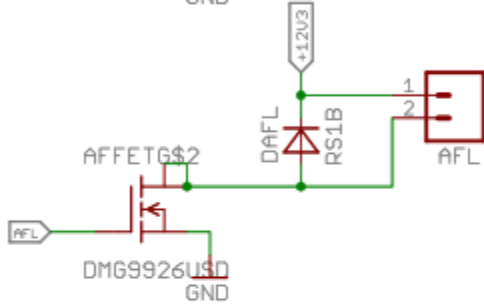
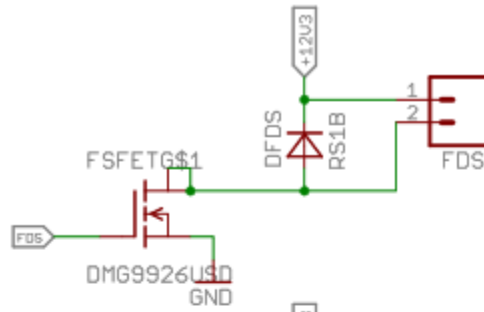
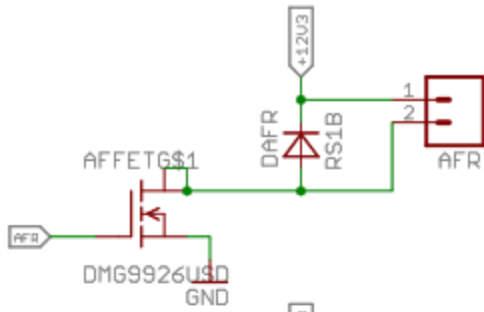
# Power Connections



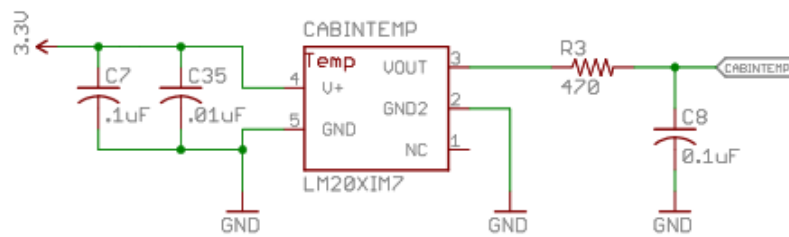
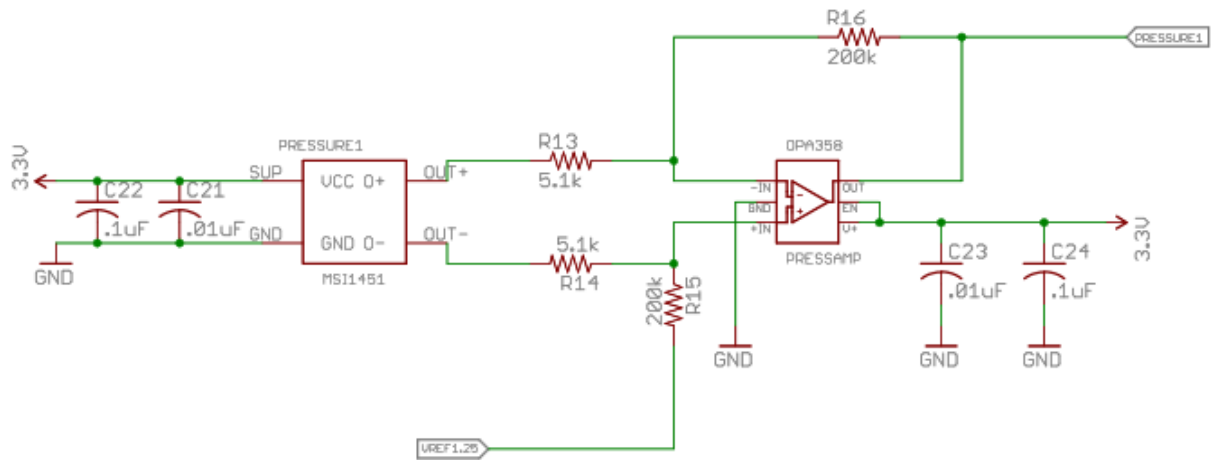
# Pump



## Solenoids and Actuators

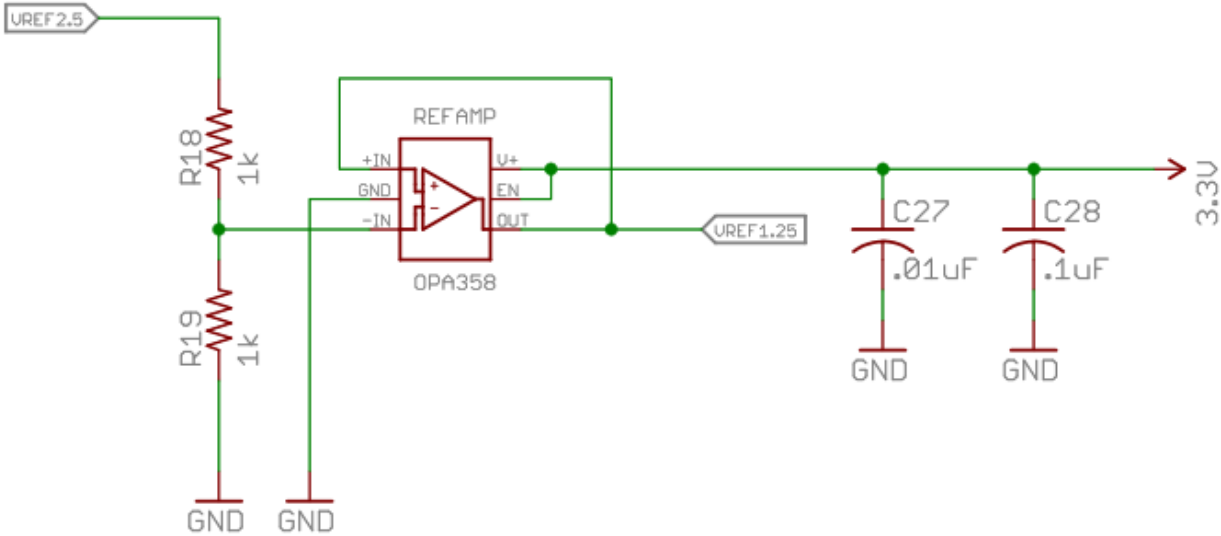
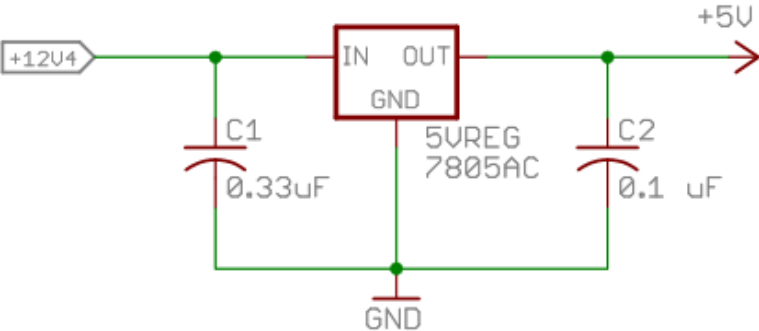


## Temperature and Pressure Sensors





# Voltage Reference Amplifier and 5V Regulator



## Appendix F: AUVLib Documentation

### File List

Here is a list of all documented files with brief descriptions:

<a href="#">auv.h</a> .....	91
<b>system.h</b> .....	92
communication/ <a href="#">acoustic_modem.h</a> .....	95
communication/ <a href="#">uart.h</a> .....	96
control/ <a href="#">base_control.h</a> .....	97
control/open_loop_manuevers.h .....	105
sensors/ <a href="#">analog_sensors.h</a> .....	108
sensors/gyro.h .....	109

## auv.h File Reference

```
#include "communication/acoustic_modem.h"
#include "communication/uart.h"
#include "control/base_control.h"
#include "control/open_loop_maneuvers.h"
#include "control/closed_loop_maneuvers.h"
#include "sensors/analog_sensors.h"
#include "sensors/gyro.h"
#include "system.h"
#include <math.h>
#include <p32xxxx.h>
#include <plib.h>
#include <stdlib.h>
#include <string.h>
```

---

## Detailed Description

### Author:

Joe Baker

### Date:

November 9, 2010

This is a meta-header file which includes all of the other AUV headers.

Definition in file [auv.h](#).

## system.h File Reference

### Defines

- #define **SYS\_FREQ** (8000000L)
- #define **GetSystemClock()** (8000000ul)
- #define **GetPeripheralClock()** (GetSystemClock()/(1 << OSCCONbits.PBDIV))
- #define **GetInstructionClock()** (GetSystemClock())
- #define [INTERNAL\\_TEMP\\_THRESHOLD](#) 75
- #define [DEPTH\\_PRESSURE\\_THRESHOLD](#) 9000
- #define [FILL](#) 1
- #define [DRAIN](#) 2

### Functions

- void [auv\\_init](#) (void)  
*Calls all other initialization functions. Call this at the beginning of mission code.*
- void [Vref2\\_5](#) (void)  
*Voltage reference generatr init. Initializes the PIC32's comparator reference output voltage to 2.5V based on the 5V input to the Vref+ pin.*
- int [wait](#) (int msec)  
*Wait a specified time in milliseconds based on the T3 timestamp.*

### Variables

- unsigned long int **timestamp**
  - short int **ballast\_status**
  - short int **lmotor\_temp\_flag**
  - short int **lmotor\_temp\_previous**
  - float **lmotor\_current**
  - short int **rmotor\_temp\_flag**
  - short int **rmotor\_temp\_previous**
  - float **rmotor\_current**
  - float **cabin\_temp**
  - float **pressure**
  - float **compass\_heading**
  - float **gyro\_x**
  - float **gyro\_y**
  - float **gyro\_z**
  - float **gyro\_accel\_x**
  - float **gyro\_accel\_y**
  - float **gyro\_accel\_z**
  - float **supply\_voltage**
  - int **current\_depth**
-

## Detailed Description

[system.h](#)

### Author:

Joe Baker

### Date:

January 31, 2011

Miscellaneous functions required for basic operation of the AUV systems as well as system health checks based on the internal sensor systems.

Definition in file [system.h](#).

---

## Define Documentation

**`#define DEPTH_PRESSURE_THRESHOLD 9000`**

Safe operating threshold for external pressure sensor: (testing required) psi

Definition at line 29 of file system.h.

**`#define DRAIN 2`**

Macro for use with the ballast\_status flag variable.

Definition at line 38 of file system.h.

**`#define FILL 1`**

Macro for use with the ballast\_status flag variable.

Definition at line 37 of file system.h.

**`#define INTERNAL_TEMP_THRESHOLD 75`**

Safe operating threshold for internal temperature sensor: 75 degrees Celsius

Definition at line 23 of file system.h.

---

## Function Documentation

**`void auv_init (void)`**

Calls all other initialization functions. Call this at the beginning of mission code.

**Returns:** void

Definition at line 61 of file system.c.

`int wait (int msecs)`

Wait a specified time in milliseconds based on the T3 timestamp.

Parameters:

<i>msecs</i>	Number of milliseconds to wait
--------------	--------------------------------

Returns: void

Definition at line 95 of file system.c.

## communication/acoustic\_modem.h File Reference

### Defines

- #define [BAUD\\_115200](#) 21

### Functions

- void [modem\\_init](#) (void)  
*Configure UART2 for the UConn acoustic modem (8 data bits, 1 parity, no RTS/CTS).*
- void [handle\\_opcode](#) (void)  
*Perform actions based on the given opcode and set/clear status flags as needed.*

### Variables

- unsigned char **opcode\_start\_found**
  - unsigned char **opcode\_end\_found**
  - unsigned char **modem\_enabled**
  - char **opcode\_buffer** [11]
- 

### Detailed Description

#### Date:

February 8, 2011

#### Author:

Joe Baker

Definition in file [acoustic\\_modem.h](#).

---

### Define Documentation

[#define BAUD\\_115200](#) 21

U2BRG register value for 115.2k baud

Definition at line 14 of file acoustic\_modem.h.

---

### Function Documentation

[void handle\\_opcode](#) (void)

Perform actions based on the given opcode and set/clear status flags as needed.

Returns: void

Definition at line 74 of file acoustic\_modem.c.

### void modem\_init (void)

Configure UART2 for the UConn acoustic modem (8 data bits, 1 parity, no RTS/CTS).

Returns: void

Definition at line 25 of file acoustic\_modem.c.

## communication/uart.h File Reference

### Functions

- void **PutCharacterU1** (const char character)
  - void **WriteStringU1** (const char \*string)
  - void **PutCharacterU2** (const char character)
  - void **WriteStringU2** (const char \*string)
- 

### Detailed Description

#### Date:

March 10, 2011

#### Author:

Joe Baker

UART helper functions from Microchip's UART Interrupt example code.

Definition in file [uart.h](#).



## control/base\_control.h File Reference

### Defines

- #define [FORWARD](#) 0
- #define [REVERSE](#) 1

### Functions

- void [control\\_init](#) (void)  
*Initializes I/O ports, timers, etc. for the AUV motion control systems.*
- void [Pump\\_On](#) (void)  
*Turn pump on.*
- void [Pump\\_Off](#) (void)  
*Turn pump off.*
- void [Dive](#) (void)  
*Turn pump, RFS, and FFS on.*
- void [Dive\\_Stop](#) (void)  
*Turn pump, RFS, and FFS off.*
- void [Surface](#) (void)  
*Turn pump, RDS, and FDS on.*
- void [Surface\\_Stop](#) (void)  
*Turn pump, RDS, and FDS off.*
- void [All\\_Jets\\_On](#) (void)  
*Turn pump, AFL, AFR, ARL, and ARR on.*
- void [AFL\\_On](#) (void)  
*Turn AFL on.*
- void [ARL\\_On](#) (void)  
*Turn ARL on.*
- void [ARR\\_On](#) (void)  
*Turn ARR on.*
- void [AFR\\_On](#) (void)  
*Turn AFR on.*
- void [All\\_Jets\\_Off](#) (void)  
*Turn pump, AFL, AFR, ARL, and ARR off.*
- void [AFL\\_Off](#) (void)  
*Turn AFL off.*
- void [ARL\\_Off](#) (void)  
*Turn ARL off.*
- void [ARR\\_Off](#) (void)  
*Turn ARR off.*
- void [AFR\\_Off](#) (void)  
*Turn AFR off.*
- void [Fill\\_Solenoids\\_On](#) (void)  
*Turn FFS and RFS on.*
- void [Drain\\_Solenoids\\_On](#) (void)

*Turn FDS and RDS on.*

- void [RDS\\_On](#) (void)  
*Turn RDS on.*
- void [FDS\\_On](#) (void)  
*Turn FDS on.*
- void [FFS\\_On](#) (void)  
*Turn FFS on.*
- void [RFS\\_On](#) (void)  
*Turn RFS on.*
- void [Fill\\_Solenoids\\_Off](#) (void)  
*Turn FFS and RFS off.*
- void [Drain\\_Solenoids\\_Off](#) (void)  
*Turn FDS and RDS off.*
- void [RDS\\_Off](#) (void)  
*Turn RDS off.*
- void [FDS\\_Off](#) (void)  
*Turn FDS off.*
- void [FFS\\_Off](#) (void)  
*Turn FFS off.*
- void [RFS\\_Off](#) (void)  
*Turn RFS off.*
- void [Set\\_M0](#) (unsigned char pct\_duty, unsigned char dir)  
*Set left motor to specified percent throttle and direction.*
- void [M0\\_Brake\\_On](#) (void)  
*Turn M0 brake on.*
- void [M0\\_Brake\\_Off](#) (void)  
*Turn M0 brake off.*
- void [Set\\_M1](#) (unsigned char pct\_duty, unsigned char dir)  
*Set right motor to specified percent throttle and direction.*
- void [M1\\_Brake\\_On](#) (void)  
*Turn M1 brake on.*
- void [M1\\_Brake\\_Off](#) (void)  
*Turn M1 brake off.*

---

## Detailed Description

### Author:

Joe Baker

### Date:

November 9, 2010

This header file contains all prototypes and definitions pertaining to the control systems of the AUV.

Definition in file [base\\_control.h](#).

---

## Define Documentation

### `#define FORWARD 0`

Motor direction macro for the H-Bridge direction pins.

Definition at line 19 of file `base_control.h`.

### `#define REVERSE 1`

Motor direction macro for the H-Bridge direction pins.

Definition at line 20 of file `base_control.h`.

---

## Function Documentation

### `void AFL_Off (void)`

Turn AFL off.

Returns: void

Definition at line 67 of file `base_control.c`.

### `void AFL_On (void)`

Turn AFL on.

Returns: void

Definition at line 44 of file `base_control.c`.

### `void AFR_Off (void)`

Turn AFR off.

Returns: void

Definition at line 72 of file base\_control.c.

void AFR\_On (void)

Turn AFR on.

Returns: void

Definition at line 48 of file base\_control.c.

void All\_Jets\_Off (void)

Turn pump, AFL, AFR, ARL, and ARR off.

Returns: void

Definition at line 61 of file base\_control.c.

void All\_Jets\_On (void)

Turn pump, AFL, AFR, ARL, and ARR on.

Returns: void

Definition at line 39 of file base\_control.c.

void ARL\_Off (void)

Turn ARL off.

Returns: void

Definition at line 77 of file base\_control.c.

void ARL\_On (void)

Turn ARL on.

Returns: void

Definition at line 52 of file base\_control.c.

void ARR\_Off (void)

Turn ARR off.

Returns: void

Definition at line 82 of file base\_control.c.

void ARR\_On (void)

Turn ARR on.

Returns: void

Definition at line 56 of file base\_control.c.

**void control\_init (void)**

Initializes I/O ports, timers, etc. for the AUV motion control systems.

Returns: void

Definition at line 11 of file base\_control.c.

**void Dive (void)**

Turn pump, RFS, and FFS on.

Returns: void

Definition at line 163 of file base\_control.c.

**void Dive\_Stop (void)**

Turn pump, RFS, and FFS off.

Returns: void

Definition at line 172 of file base\_control.c.

**void Drain\_Solenoids\_Off (void)**

Turn FDS and RDS off.

Returns: void

Definition at line 127 of file base\_control.c.

**void Drain\_Solenoids\_On (void)**

Turn FDS and RDS on.

Returns: void

Definition at line 95 of file base\_control.c.

**void FDS\_Off (void)**

Turn FDS off.

Returns: void

Definition at line 138 of file base\_control.c.

**void FDS\_On (void)**

Turn FDS on.

Returns: void

Definition at line 106 of file base\_control.c.

**void FFS\_Off (void)**

Turn FFS off.

Returns: void

Definition at line 133 of file base\_control.c.

**void FFS\_On (void)**

Turn FFS on.

Returns: void

Definition at line 101 of file base\_control.c.

**void Fill\_Solenoids\_Off (void)**

Turn FFS and RFS off.

Returns: void

Definition at line 121 of file base\_control.c.

**void Fill\_Solenoids\_On (void)**

Turn FFS and RFS on.

Returns: void

Definition at line 89 of file base\_control.c.

**void M0\_Brake\_Off (void)**

Turn M0 brake off.

Returns: void

Definition at line 213 of file base\_control.c.

**void M0\_Brake\_On (void)**

Turn M0 brake on.

Returns: void

Definition at line 208 of file base\_control.c.

**void M1\_Brake\_Off (void)**

Turn M1 brake off.

Returns: void

Definition at line 237 of file base\_control.c.

**void M1\_Brake\_On (void)**

Turn M1 brake on.

Returns: void

Definition at line 232 of file base\_control.c.

**void Pump\_Off (void)**

Turn pump off.

Returns: void

Definition at line 158 of file base\_control.c.

**void Pump\_On (void)**

Turn pump on.

Returns: void

Definition at line 153 of file base\_control.c.

**void RDS\_Off (void)**

Turn RDS off.

Returns: void

Definition at line 148 of file base\_control.c.

**void RDS\_On (void)**

Turn RDS on.

Returns: void

Definition at line 116 of file base\_control.c.

**void RFS\_Off (void)**

Turn RFS off.

Returns: void

Definition at line 143 of file base\_control.c.

**void RFS\_On (void)**

Turn RFS on.

Returns: void

Definition at line 111 of file base\_control.c.

**void Set\_M0 (unsigned char *pct\_duty*, unsigned char *dir*)**

Set left motor to specified percent throttle and direction.

Parameters:

<i>pct_duty</i>	0-100 value for a throttle percentage
<i>dir</i>	Motor direction (use FORWARD/REVERSE macros)

Returns: void

Definition at line 193 of file base\_control.c.

**void Set\_M1 (unsigned char *pct\_duty*, unsigned char *dir*)**

Set right motor to specified percent throttle and direction.

Parameters:

<i>pct_duty</i>	0-100 value for a throttle percentage
<i>dir</i>	Motor direction (use FORWARD/REVERSE macros)

Returns: void

Definition at line 218 of file base\_control.c.

**void Surface (void)**

Turn pump, RDS, and FDS on.

Returns: void

Definition at line 180 of file base\_control.c.

**void Surface\_Stop (void)**

Turn pump, RDS, and FDS off.

Returns: void

Definition at line 187 of file base\_control.c.



## control/open\_loop\_manuevers.h File Reference

### Defines

- #define **PI** 3.14159265

### Functions

- void [open\\_horizontal\\_manuever](#) (int decimeters)  
*Translate along the X axis using the motors with open-loop control.*
- void [open\\_full\\_surface\\_manuever](#) (void)  
*Fully surface with open-loop control.*
- void [open\\_surface\\_manuever](#) (int decimeters)  
*Rise a specified amount with open-loop control.*
- void [open\\_dive\\_manuever](#) (int decimeters)  
*Fill the ballast tanks in order to dive to a certain depth with open-loop control.*
- void [open\\_yaw\\_manuever](#) (int degrees)  
*Rotate about the Z axis using the motors with open-loop control.*
- void [open\\_pitch\\_manuever](#) (int degrees)  
*Rotate about the Y axis using the maneuvering jets (TODO: ballast?) with open-loop control.*
- void [open\\_roll\\_manuever](#) (int degrees)  
*Rotate about the X axis using the maneuvering jets with open-loop control.*

---

### Detailed Description

Open-loop motion control functionality for the AUV.

#### Date:

January 29, 2011

#### Author:

Joe Baker

Definition in file [open\\_loop\\_manuevers.h](#).

---

### Function Documentation

#### void [open\\_dive\\_manuever](#) (intdecimeters)

Fill the ballast tanks in order to dive to a certain depth with open-loop control.

#### Parameters:

<i>decimeters</i>	Distance to dive in decimeters
-------------------	--------------------------------

Returns: void

Definition at line 167 of file open\_loop\_maneuvers.c.

**void open\_full\_surface\_maneuver (void)**

Fully surface with open-loop control.

Returns: void

Definition at line 144 of file open\_loop\_maneuvers.c.

**void open\_horizontal\_maneuver (intdecimeters)**

Translate along the X axis using the motors with open-loop control.

Parameters:

<i>decimeters</i>	Distance to travel in decimeters.
-------------------	-----------------------------------

Returns: void

Definition at line 98 of file open\_loop\_maneuvers.c.

**void open\_pitch\_maneuver (intdegrees)**

Rotate about the Y axis using the maneuvering jets (TODO: ballast?) with open-loop control.

Parameters:

<i>degrees</i>	Direction to turn relative to LCS. Positive is in the CCW direction.
----------------	--

Returns: void

Definition at line 243 of file open\_loop\_maneuvers.c.

**void open\_roll\_maneuver (intdegrees)**

Rotate about the X axis using the maneuvering jets with open-loop control.

Parameters:

<i>degrees</i>	Direction to turn relative to LCS. Positive is in the CCW direction.
----------------	--

Returns: void

Definition at line 302 of file open\_loop\_maneuvers.c.

**void open\_surface\_maneuver (intdecimeters)**

Rise a specified amount with open-loop control.

Parameters:

<i>decimeters</i>	Distance to rise in decimeters
-------------------	--------------------------------

Returns: void

Definition at line 154 of file open\_loop\_maneuvers.c.

`void open_yaw_maneuver (intdegrees)`

Rotate about the Z axis using the motors with open-loop control.

Parameters:

<i>degrees</i>	Direction to turn relative to LCS. Positive is in the CCW direction.
----------------	--

Returns: void

Definition at line 195 of file open\_loop\_maneuvers.c.

## sensors/analog\_sensors.h File Reference

### Defines

- #define [ADC\\_RESOLUTION](#) 0.0032226563

### Functions

- void [analog\\_sensor\\_init](#) (void)  
*Initialize the ADC for the AUV's analog sensor systems.*
- void [get\\_cabin\\_temp](#) (void)  
*Sets the cabin\_temp global variable based on the ADC value and LM20 transfer function.*
- void [get\\_compass\\_heading](#) (void)  
*Sets compass\_heading global variable using equations from Honeywell AN-203.*

### Variables

- int **m1\_sense\_result**
  - int **cabin\_temp\_result**
  - int **pressure1\_result**
  - int **gyro\_dac\_result**
  - int **comp\_z\_result**
  - int **comp\_x\_result**
  - int **comp\_y\_result**
  - int **m0\_sense\_result**
- 

## Detailed Description

### Author:

Joe Baker

### Date:

November 10, 2010

This header file contains all prototypes and definitions pertaining to the analog sensor systems of the AUV.

Definition in file [analog\\_sensors.h](#).

---

## Define Documentation

**#define** [ADC\\_RESOLUTION](#) 0.0032226563

Resolution = Supply voltage/ADC bits = 3.3/(2<sup>10</sup>)

Definition at line 18 of file analog\_sensors.h.

## sensors/gyro.h File Reference

### Functions

- void [gyro\\_init](#) (void)  
*Initialize SPI1 for the ADIS16354.*
- 

### Detailed Description

#### Date:

January 31, 2011

#### Author:

Joe Baker

Definition in file [gyro.h](#).

---

### Function Documentation

#### void gyro\_init (void)

Initialize SPI1 for the ADIS16354.

#### Returns:

void

Definition at line 11 of file gyro.c.

## References

- Advanced Circuits. "Advanced Circuits Capabilities." *Advanced Circuits Capabilities*. Aurora: Advanced Circuits, September 1, 2010.
- Barkhordarian, Vrej. "Power MOSFET Basics." *International Rectifier Web site*. October 25, 2005. [www.irf.com/technical-info/appnotes/mosfet.pdf](http://www.irf.com/technical-info/appnotes/mosfet.pdf) (accessed January 21, 2011).
- CadSoft. *Eagle Tutorial Version 5*. Pembroke Pines: CadSoft, 2008.
- CollabNet Inc. *SourceForge Enterprise Edition 4.4 SP1 User Guide*. Brisbane: CollabNet Inc., 2008.
- David, Radu A, Maxwell E French, Brandon M Habin, and Akil Kerjiwal. *Design of Autonomous Underwater Vehicle and Optimization of Hydrodynamic Properties and Control*. MQP, Worcester: Worcester Polytechnic Institute, 2009.
- Di Jasio, Lucio. *Programming 32-bit Microcontrollers in C: Exploring the PIC32*. Burlington: Newnes, 2008.
- Etkin, Bernard. *Dynamics of Atmospheric Flight*. New York: Wiley, 1972.
- Honeywell. "AN-203: Compass Heading Using Magnetometers." *Honeywell Web site*. July 1, 1995.  
<http://www.honeywell.com/sites/servlet/com.merx.npoint.servlets.DocumentServlet?docid=D47F07978-4A99-3FC1-6F40-7CB8271A5B30> (accessed March 10, 2011).
- Microchip. *32-Bit Language Tools Libraries*. Chandler: Microchip, 2009.
- . *PIC32 Family Reference Manual*. Chandler: Microchip, 2008.
- . *PIC32MX3XX/4XX Family Data Sheet*. Chandler: Microchip, 2008.
- Moussette, Daniel, Ashish Palooparambil, and Jarred Raymond. *Optimization and Control Design of an Autonomous Underwater Vehicle*. MQP, Worcester: Worcester Polytechnic Institute, 2010.
- National Semiconductor. "LM20 micro SMD Temperature Sensor Datasheet." *National Semiconductor Web site*. September 21, 2010.  
<http://www.national.com/mpf/LM/LM20.html> (accessed December 9, 2010).
- Schmidt, Louis V. *Introduction to Aircraft Flight Dynamics*. AIAA, 1998.

Suppanz, Brad. *Trace Width Calculator*. January 1, 2007.

[http://www.4pcb.com/index.php?load=content&page\\_id=95](http://www.4pcb.com/index.php?load=content&page_id=95) (accessed December 30, 2010).