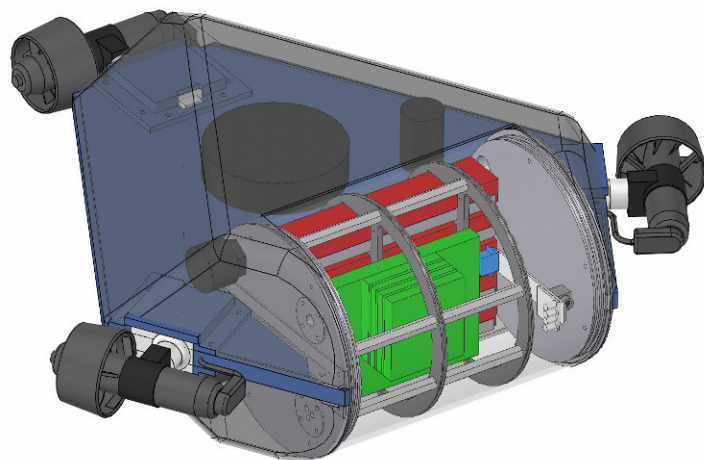


NC STATE UNIVERSITY

# SEAWOLF I

Autonomous Underwater Vehicle Platform



Seawolf I's deltoid hull and vectored thrust design are patent-pending and proprietary to Vortex HC, LLC.

## Authors

William Cox, Mike Faircloth, Sterling Greene, Frankie Myers, Jim Simpson, Ryan Sturmer

## Additional Team Members

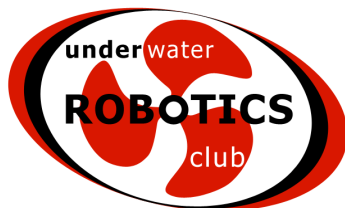
Erin Anderson, Brian Burney, Mike DeCarli, DeJuan Gales, Bill Heisey, Eric Jones, Jesse McClelland, Matt McGinnis, Jim McKinney, Nader Moussa, Eric Phillips, Justin Sherrill, Alex Stewart, Chris Thomas, Derek Thompson, Lee Wakefield, Daniel Zuckley

## Faculty Advisors

Dr. John Sutton, Mr. Bart Greene

## Corporate Partner

Vortex HC, LLC



**Underwater Robotics Club**  
Box 7306, SORC  
North Carolina State University  
Raleigh, NC 27695  
<http://www.ncsurobotics.org>



**Vortex HC, LLC**  
900-E Perimeter Park Dr.  
Morrisville, NC 27560  
(919)462-8828  
<http://www.vortexhc.com>

**Abstract--***This paper describes the design and construction of Seawolf I, an autonomous underwater vehicle (AUV) developed by students from the Electrical and Computer Engineering department at NC State University and staff from Vortex HC, LLC. Seawolf I features three poseable thrusters, located at each vertex on the AUV's deltoid hull. Each thruster can independently pivot for maximum maneuverability in 6 DOF. All electronics are housed in a clear acrylic tube at the front of the vehicle. Seawolf I features a full suite of sensors including a DVL, IMU, Altimeter, 180-degree tiltable color camera, and an acoustic hydrophone array. Mission planning, sensor data aggregation, and image processing are carried out on a 550MHz PC/104 stack. Acoustic navigation is handled by a custom DSP system.*

## Mechanical Design

### Chassis

The vehicle's midplane is cut from UHMW Polyethylene. Around the vehicle is a custom-fitted fiberglass shell, designed to present a uniform drag surface to the water. Closed-cell foam and lead ballast weights are attached at various points inside the shell to set trim and buoyancy.

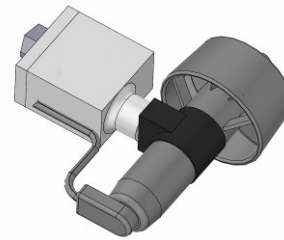


*Fiberglass shells*

### Servo/Thruster Assemblies

The vehicle is propelled by three Seabotix thrusters which mount to custom submersible servo housings, depth rated to more than 200 feet. The housings are fitted with 3-pin Seacon connectors which provide signal and power. The Seabotix thrusters are fitted with

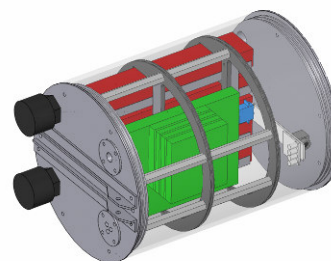
custom electrical connectors machined from aluminum and sealed with Polyurethane.



*Servo/thruster assembly*

### Electronics Tube

The electronics are contained in an acrylic tube with aluminum end caps. The tube is designed with easy assembly and servicing in mind. Each end cap has two rubber o-rings to provide a watertight seal. One end cap is fixed to the electronics chassis, and it is onto this end cap that the bulkhead connectors, kill switch, and power switch are mounted. The other end cap is easily removable, providing quick access to the batteries. Two ribs machined from Delrin provide additional reinforcement for the tube, dividing it into thirds. These ribs are mounted to the fixed end cap via aluminum spacers. They provide channels for the batteries and electronics to slide into. The PC/104 stack, DSP board, and camera/servo assembly are mounted to G-10 fiberglass, which slides into the ribs. The electronics chassis is thermally conductive to the end caps, aiding in heat dissipation.



*Electronics tube*

### ***Dropper Mechanism***

This year's mission states that the robot must deposit two "markers" into a black bin. To accomplish this we constructed a water-tight dropper mechanism that is magnetically coupled to the markers. When the dropper is released the magnetic force between the marker and dropper is broken and the markers are free to drop.

The dropper mechanism consists of two pull-type solenoids with high-strength magnets attached. While the solenoid is deactivated the magnet is coupled to the markers through the wall of the dropper box. To drop a marker, a solenoid is activated and its magnet pulls away from the wall, releasing the marker and allowing them to drop.

## **Electrical Design**

### ***Batteries***

Seawolf I is powered by six ThunderPower 14.8 volt, 8000mAh batteries. These batteries are used to provide two 29.6 volt busses. The first bus, a series combination of two batteries, powers the PC-104, the interconnection board and all sensors. The second bus, a series/parallel combination of four batteries, provides power to the thrusters.

### ***Switching***

Electronics power is delivered to the interconnection board via a Crydom 100 volt/40 amp solid-state relay. Thruster power is delivered directly to the bulkhead connectors via a 100 volt/100 amp Crydom solid-state relay. Each relay is controlled by a magnetic reed switch which is mounted inside the fixed end cap. Two external sliding rods with magnets attached actuate the reed switches from outside the electronics tube to interrupt power to the thrusters and electronics separately. These rods protrude slightly from the top of the vehicle allowing easy access.

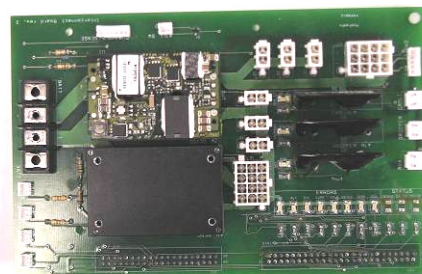
The thruster switch circuit is also fed through a MOSFET switch on the interconnect board, which is toggled with a digital output on the PC/104, allowing software to "kill" the thrusters directly.

### ***Power Filtering***

The output of each relay is filtered using capacitors. The thruster relay uses two 3300 uF capacitors in parallel and the electronics relay uses one 2700 uF capacitor. These capacitors provide adequate filtration (spikes, high frequency noise, etc) for each application.

### ***Power/Signal Distribution PCB***

A custom designed PCB mounts on top of the PC/104 stack and contains two Datal DC/DC converters (12V and 6V) which supply power to the sensors, PC/104, DSP board, and servos. The board also acts as a central hub for digital and analog I/O to and from the PC/104. In addition, a microcontroller for thruster control is mounted on this board, as well a relay to control dropper operation.



*Power/signal distribution PCB*

### ***Cabling and Connectors***

Connections to the outside of the electronics tube utilize Seacon bulkhead connectors and cables. Two bulkhead "pie" connectors are used—the first connecting the DVL, altimeter, and servos, and the second connecting the thrusters.

### **Health Monitoring**

The voltage levels of both busses are monitored using simple resistor voltage dividers. Battery current is monitored by the PC/104 using two Amploc AMP50 hall-effect linear current sensors. Two thermistor probes from Oven Industries—one mounted in the battery stack and the other mounted near the electronics—monitor temperature in the electronics tube. The outputs of the voltage dividers, current sensors, and thermistors are connected to the interconnect PCB, which in turn connects them to the PC/104's onboard DAQ. They are all monitored in software to ensure proper operation.

### **Navigation Sensors**

#### ***DVL***

Seawolf I uses an RDI Workhorse Navigator doppler velocity logger for navigation. The DVL uses a sonar array to accurately measure ship velocity relative to the bottom and an internal pressure sensor to measure depth. The DVL also provides roll, pitch and heading. The PC/104 receives data from the DVL over a RS232 serial port.

#### ***IMU***

In addition to the DVL, we have included a MT9 inertial measurement unit from Xsens Motion Technologies which provides more accurate roll, pitch, and yaw data. This device is housed in the electronics tube and, like the DVL, communicates with the PC/104 via RS232.

#### ***Altimeter***

Seawolf I also features a PA200 high-precision altimeter from Tritech International, which is connected to the PC/104 via an analog input.

### **Camera**

A Sony 38CSHR NTSC board camera is mounted on a boom attached to a servo which pivots the camera 180 degrees at the front of the electronics tube.

### **Hydrophone Array**

For acoustic navigation, we installed an array of 3 Reson TC4013 miniature hydrophones equidistant around the bottom of the vehicle. The coaxial cables from these hydrophones enter the electronics tube through a watertight removable aluminum puck in the fixed end cap.



*Reson TC4013 hydrophone*

### **Computer Systems**

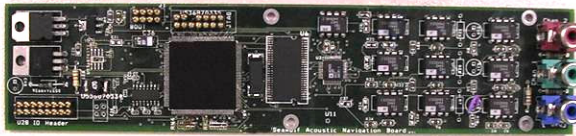
#### ***PC/104***

For our main computer, we chose the Diamond Systems Hercules PC/104 with a 550MHz Via Eden processor, 256MB of RAM, and a 20GB 2.5 inch hard disk. This particular model has onboard data acquisition (DAQ) circuitry that provides 32 analog I/O and 40 digital I/O lines (4 of which are PWM channels), thus eliminating the need for an additional card. A Parvus FG104 framegrabber board is used to capture images from our camera.

#### ***Acoustic Navigation DSP Board***

It was quickly realized that a dedicated processor would be required to process acoustic data from our hydrophone array. For this, we chose to build our own system based on the Texas Instruments TMS320VC5502 DSP. We use a 64Mbit SDRAM chip for storing sample buffers and

for algorithm scratch space. For sampling data from the hydrophones, we tried several ADCs before settling on the AD7655 from Analog Devices. This particular ADC has 4 input channels and samples two channels simultaneously. It is capable of a per-channel throughput of 250kSPS. The signal chain from the hydrophones begins with a 3-stage instrumentation amplifier circuit composed of AD743 op-amps. The first two stages each provide a gain of 20, and the last stage is a summing amplifier which uses the AD780 precision 2.5V reference source to center the signals at 2.5V before they are sent to the ADC.



*Acoustic Navigation PCB*

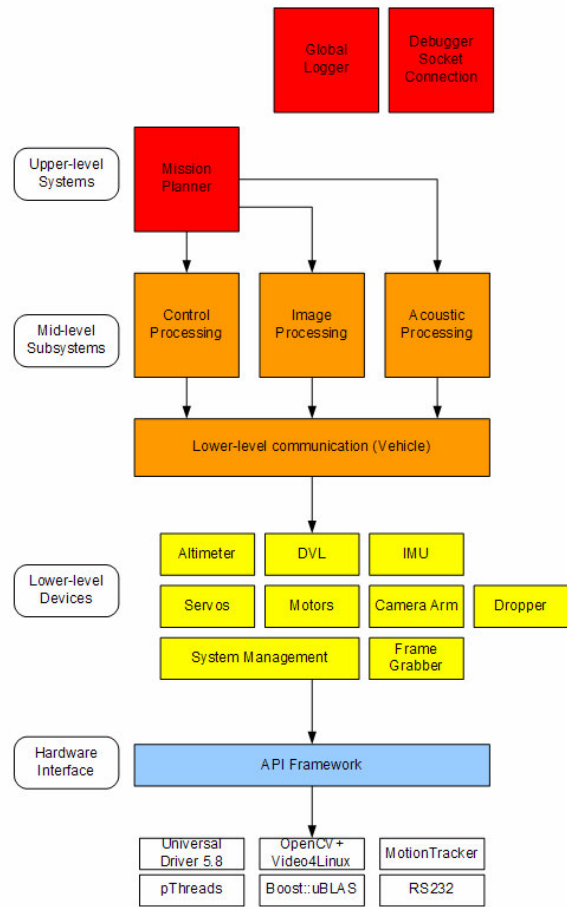
We used Spectrum Digital’s C5510DSK development kit to prototype our algorithms and analog front end hardware and then migrated our design to a custom 4-layer PCB. This PCB contains the DSP, SDRAM, ADC, op-amps, power supply electronics, an RS232 level shifter. In addition, we’ve included a header for a JTAG emulator, which allows realtime, in-circuit debugging of the system

**Thruster Controller Processor**

A BasicATOM microcontroller is used to arbitrate communication between the PC/104 and the thrusters’ I2C bus. Each message is formatted with a start-byte, three bytes representing the speed of each thruster and a checksum. The start-byte guarantees that the microcontroller will not misinterpret messages and the checksum guarantees that it will not execute corrupted data. If the microcontroller does not receive a signal from the PC/104 for a certain length of time, it automatically shuts down the thrusters.

**Software Architecture**

The main computer runs Slackware 10 Linux with a 2.6 kernel. Diamond Systems Universal Driver 5.8, Boost uBLAS, XSens Motiontracker, and OpenCV are external libraries we use for collecting sensor data, calculating control solutions and image processing. Seawolf I’s software is written in C++ and is organized into three tiers along with a common hardware API. Each tier adds a layer of abstraction from the hardware.



*Software organization*

**Hardware API**

Seawolf I uses several interfaces to communicate to all of its devices. For instance, the DVL and IMU use RS232, but the altimeter uses an analog input to the onboard DAQ. The thruster servos rely on PWM signals from the DAQ. A central API

was developed that encompasses all of these requirements, providing a streamlined interface to the computer's hardware and a unified error handling scheme. This API allows for future updates to underlying hardware drivers without modification of upper level components.

The API wraps functionality provided by Diamond System's Universal Driver, which allows access to I/O on the DAQ. The API also provides functions for RS232 serial communication. Several functions using the Video4Linux library were added that allow the image processing routines to capture video frames or change properties of the framegrabber.

### Multi-threaded Software Framework

We utilize the pthreads library for creating multiple execution threads in Seawolf I's main program. One of the challenges with using this C threading library with our C++ application was creating a C++ function of a class as a thread. The pthreads library only allows C-linked functions to be threads, but we needed C++ member functions to run in separate threads. The only workaround for this problem was to create a separate C-linked function which calls our C++ member function. This fix allowed us to write almost every class as a separate execution thread (or sometimes multiple threads in a single class).

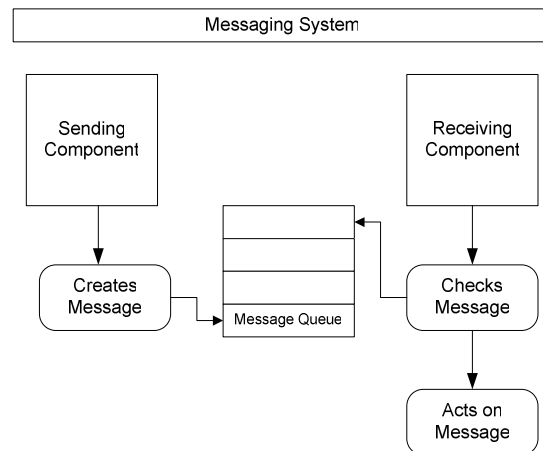
### Subsystem Configuration

Rapid development required a quick and easy way to finely tune variables for our classes. We also wanted to have a way for enabling or disabling parts of the software without recompiling everything. A configuration file format was created to address these issues. The configuration file is globally accessed and components can specify a section in the configuration file that their options should be placed. A typical configuration file has a format as follows:

```
[SECTION] = SOCKET
  activate = 1
  max_retries = 10
  keepalive = 1
  cycletime = 25
```

### Interprocess Communication

We have created a messaging system that allows a lower-level component to send data to a higher level component without having direct access to that class. A message is created by a lower-level component and passed into a message router, which places it into a priority queue. The upper-level component checks its message box (the queue) periodically for new messages and acts upon them. (Figure 1) depicts this relationship. Data can be passed with the message, or the message can identify the data that needs to be accessed in the class that created it. Messages can also originate from the debugger via TCP sockets.



*Messaging system relationship*

### Tier I: Low-level Devices

The lowest level of the software on Seawolf I, directly above the hardware API, is the device level. This level provides interfaces for all of the sensors and peripherals of the vehicle.

All of the device classes were designed with the desire to modularize the entire system.

This design choice greatly aided our migration of our original analog-controlled thrusters to our current I2C controlled thrusters. Because of our loosely-coupled architecture, none of the upper level control systems required new code to use the different thrusters

Since the DVL and IMU are constantly streaming serial data, we use separate execution threads to retrieve data from them. We can poll the voltage on the altimeter's analog channel to determine our altitude with a simple equation.

A health monitoring class uses the analog inputs on the PC/104's onboard DAQ to monitor temperature, voltage and current sensors in the electronics tube. If the system management class measures a sensor that is out of its tolerances, it would report the problem to the mission controller (Tier III) which would decide what to do.

### ***Tier II: Mid-level Subsystems***

The majority of work is performed in the second tier. We divided the vehicle's functions into four subsystems. Each subsystem runs in a separate thread and the software framework provides data/event logging and messaging to each subsystem.

A vehicle subsystem performs device initialization and shutdown, as well as some miscellaneous monitoring information over the low level devices. An acoustics subsystem collects heading information from the DSP. An image processing subsystem interfaces with the framegrabber and performs machine vision algorithms. Finally, a control subsystem uses fuzzy logic to calculate all of our movement solutions given different waypoints and desired conditions.

### ***Tier III: High-level Control***

The highest level of control over the vehicle is a component called mission control. Mission

control executes a pre-defined mission and delegates commands to the lower subsystems. Mission control does not need to know details of how goals are accomplished, only that they are accomplished in a particular order.

The mission controller can setup each subsystem with different tasks and react to messages sent from the lower level upon goal completion. For instance, mission control may want to find a particular shape and move towards it. For each subsystem, mission control sets the conditions. Controls might be asked to perform a sweep search pattern. Imaging would be asked to find the desired shape and alert mission control when it detects a positive match via the event messaging system.

A global logger allows any part of Seawolf I to log data and events in a central location, instead of its individual log file. The socket connection uses two threads for sending and receiving data.

## **Fuzzy-logic Control Software**

### ***Algorithms***

The control subsystem combines fuzzy logic controllers with kinematics equations based on our thruster geometry to produce closed-loop thrust-angle calculations. The control subsystem uses separate controllers for translating, rotating (yaw), stabilizing (roll and pitch), z-stability (maintaining a depth or altitude) and stopping (maintaining a zero speed in the x-y plane). The fuzzy logic controllers read position/velocity data from the DVL, IMU, and altimeter, and output force and torque vectors which are then fed into the kinematics equations which determine optimal thrust/angle solutions for each thruster.

### ***Fuzzy-logic Macro Language***

We decided to create a simple fuzzy logic evaluation engine using C++ macros for our controls system. A set of macros enable us to mimic FCL (Fuzzy Control Language) and English rules. The following is a sample rule written with our fuzzy logic syntax:

```
rule[0] = If(Temperature) is(Cold)
then
output[0] = High;
end
```

Rules can contain fuzzy logical ANDs and ORs. We can also describe fuzzy membership sets with simple macros. We have the basic set shapes—triangle, sigmoids, trapezoids and singletons. We were able to describe fuzzy controllers with upwards of 64 rules and 8 sets using our fuzzy macro language.

### ***Debugger Automation***

From the debugger, the controls subsystem can be put into different modes which alter the level of control we have over the vehicle. We can choose to directly control each thruster's angle and thrust independently, request open-loop movements by directly manipulating the inputs of the kinematics equations, or provide destination waypoints or desired velocities to test closed-loop control. The run mode is divided into speed-major movements (movements where we're concerned with our current speed only), or translation-major movements (movements where we're concerned with how far we've moved.)

### ***Image Analysis Software***

Seawolf I uses a vision library written with Intel's OpenCV platform to acquire and process images taken from a color CCD camera. The Video4Linux API is used to acquire images, and OpenCV, in conjunction with some specially written processing libraries is used to analyze the images, producing useful mission data. The system

frame rate is adaptive, as images are taken only on an as-needed basis, but can run as high as 20 fps.

### ***Pipeline***

Pipeline detection and tracking is done with an adaptive color thresholding technique. While the vehicle performs a box search pattern in the x-y plane, a 3-channel image is captured from the camera, and is transformed into a single channel grayscale image that is thresholded and assessed to determine the presence, location and direction of the pipe.

The transformation process uses a "target" 3-channel color to determine which pixels are most likely to belong to the pipe, and stores that likelihood as a value in the grayscale image. As a simple example, for a target color of <128,64,0> (taken as an RGB triple) each pixel in the output image is determined by the euclidian distance from that point to the corresponding point in the input image. In this transformed image, output pixels are "darker" (closer to 0) when the input pixels are closer to the target color, and "lighter" (closer to 255) when input pixels are farther away. An adaptive equalization/thresholding technique is then applied to the output image to determine which pixels actually belong to the pipe, and which belong to the background. A regression is applied to a subsample of these pixels and the resulting line information is relayed to mission control for further processing.

### ***Drop Box***

During navigation of the pipeline, the vehicle maintains a sufficient altitude to capture the drop box completely in the field of view (about 4 feet). If the end of the pipe is detected, the image is further scanned for a large white region (the white "cloud" surrounding the bin. If this region is not

found, the vehicle begins a sweep pattern to locate the pipe and/or the white cloud. When the cloud *is* found, the algorithm begins to search for the square drop box. To locate this box, the input image is processed with a fast Hough transform designed to locate squares, and the maximum value of the resulting data is extracted. The location of this value in Hough space is then assessed to determine if the square that is detected is “sensible” given other information about the robot's environment. As an example, since the drop box is of a known size, a square extracted from the de-houghing process will have an apparent size that is comparable, given the altitude of the robot. Other parameters that are checked include orientation, color, and edge definition. The algorithm communicates size and orientation to mission control for repositioning. When the box is located in the center of the frame, the algorithm alerts mission control, which then performs the drop sequence.

### ***Beacon Detection/Analysis***

Both the random sequence indicator beacon and the “docking station” beacon rely on the same algorithm for processing. In addition to adaptive color thresholding, the beacons can be detected through knowledge of the fact that they are blinking. Much like the pipe detector, the goal is to determine, through filtering, which pixels belong to the beacon and which belong to the background. To isolate potentially blinking pixels, an accumulator is used that accumulates the difference between identical pixels over several images in time. This accumulator will reflect a higher value (corresponding to a higher image intensity) at pixels that show the most change over time. Shape recognition is then applied to the accumulator to determine which pixels belong to the beacon, and which belong to background. Once this is determined, these pixels are used to generate a region of interest (ROI) which is used to

analyze all the images in the buffer to determine mission relevant parameters such as blink rate and color. If color cannot be adequately assessed as either green or red (due to color absorption of the water), the algorithm requests mission control to move towards the light.

## **Acoustic Navigation Software**

### ***DSP Bootstrapping***

One reason we chose this particular DSP was because it allows us to bootstrap firmware into RAM over a UART, eliminating the need for non-volatile memory and making firmware upgrades much easier. The firmware for the DSP is stored in a file on the PC/104. During software startup, the PC/104 transmits the file over RS232 to the DSP's UART.

### ***DSP-to-PC/104 Communication***

After bootstrapping is complete, the DSP automatically begins execution of the downloaded firmware. The firmware uses the same UART that was used for bootstrapping to communicate with the PC/104. The PC/104 sends a one-byte command indicating that the DSP should begin analyzing for a ping, which frequency to look for, and whether to search for the practice or competition ping. The DSP sends the resulting ping estimates as a two-byte return value.

### ***Ping Detection Algorithm***

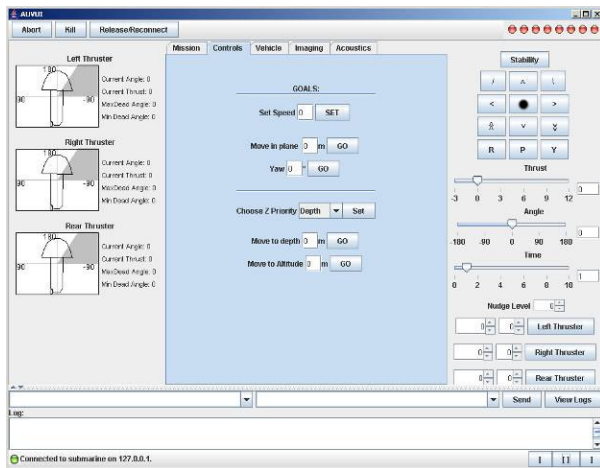
All three hydrophones are sampled within a timer interrupt routine that runs at 200kHz. Data is sampled for 2 seconds, after which the timer is shut off and the ping detection algorithm runs. We first bandpass the captured data on one channel using a 4<sup>th</sup> order Butterworth filter centered around the expected ping frequency with a bandwidth of 1kHz and perform adaptive thresholding to determine a rough location of the edge of

the ping. We then establish a window around this location and bandpass the remaining channels within this window. If we're running in "practice" mode, then we should see two pings, so this analysis is done again to differentiate the pings and select the correct one, based on whether they are .9 seconds apart or 1.1 seconds apart.

Pair-wise time delays are calculated by evaluating the cross correlation series for each pair of channels. Given the geometry of the hydrophone array and assuming a plane wave propagation of the ping, we can triangulate the location of the pinger using simple trigonometry. Using 3 hydrophones not only eliminates the 180-degree ambiguity of 2 hydrophones, but it provides 3 pair-wise delays that can be averaged for greater accuracy.

### External Debugger GUI

We developed a Java-based GUI to communicate with and debug Seawolf I during pool testing. This GUI allows the user to view status information from each subsystem, control the vehicle's motions, and set mission objectives. It uses java TCP sockets to communicate with the vehicle.



*Java debugger*

Data is sent to and from the vehicle in data frames formatted like this:

```
<SET>/<QUE>/<ANS>/<CMD> <TARGET> <FIELD> <DATA>
```

The first field specifies the message type, the second field specifies the target mid-layer subsystem, and the third field identifies the message, and the final field contains the actual data. This data is received by the debugger socket connection thread on the vehicle and is placed in the appropriate mid-level subsystem's messaging queue.

### Acknowledgements

We gratefully acknowledge the following people and organizations for their contributions to this project:

Dr. Steven Goodridge from SignalScape, for guidance and technical support on use of Fuzzy Logic algorithms for mobile robot control. Dr. Kent Scarbrough from ARL/UT, for guidance and technical support on the acoustic navigation algorithms. DARPA ATO and ONR for sponsoring the ARTEMIS program, which has enabled the development of the Seawolf I AUV. Dr. Theo Kooij, ARTEMIS program manager for DARPA ATO and ONR. Mr. Ed Mihalak, SETA support and overall facilitator of the ARTEMIS program.