Autonomous Data Processing and Behaviors for Adaptive and Collaborative Underwater Sensing

by

Keja S. Rowe

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2012

Author .....
Department of Electrical Engineering and Computer Science
Feb 1, 2012

Certified by .....
Henrik Schmidt
Professor of Mechanical and Ocean Engineering
Thesis Supervisor

Accepted by .....
Dennis M. Freeman
Chairman, Masters of Engineering Thesis Committee

Autonomous Data Processing and Behaviors for Adaptive and Collaborative Underwater Sensing

by

Keja S. Rowe

Submitted to the Department of Electrical Engineering and Computer Science
on Feb 1, 2012, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

In this thesis, I designed, simulated and developed behaviors for active riverine data collection platforms. The current state-of-the-art in riverine data collection is plagued by several issues which I identify and address. I completed a real-time test of my behaviors to insure they worked as designed. Then, in a joint effort between the NATO Undersea Research Center (NURC) and Massachusetts Institute of Technology (MIT) I assisted the Shallow Water Autonomous Mine Sensing Initiative (SWAMSI)'11 experiment and demonstrated the viability of multi-static sonar tracking techniques for seabed and sub-seabed targets. By detecting the backscattered energy at the mono-static and several bi-static angles simultaneously, the probabilities of both target detection and target classification should be improved. However, due to equipment failure, we were not able to show the benefits of this technique.

Thesis Supervisor: Henrik Schmidt
Title: Professor of Mechanical and Ocean Engineering

Acknowledgments

I would like to thank the Office of Naval Research for supporting this research. Without their financial backing, it would not have been possible.

I would also like to thank Professor Henrik Schmidt for his guidance and support during this project as well as Maurice Fallon and the Laboratory for Autonomous Marine Sensing Systems for their help in completing this project.

Lastly, I would like to thank Professors John Leonard and Patrick Winston for their general support during my time at the Institute.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

When scientists and engineers collect data, they are often doing so as a means to accomplish some objective or goal, not for the sake of data collection itself. They may be trying to determine how certain environmental properties effect an environmental phenomenon. Such as the effects of UV radiation on the degradation of a certain type of plastic, or the effect of ambient water temperature on cuddle fish spawning rates. Or, they may be looking for an environmental feature such as the thermocline in a large body of water, or the thalweg of a river. However, regardless of what the scientist or engineer is using the data for, there is some criterion that differentiates bad data (data that does not help you accomplish your objective and does not give you insight as to why it cannot be accomplished) from good data (data that helps you accomplish your goal or helps you show it cannot be accomplished). Collecting data on an autonomous platform allows us to assess the data collected against that criterion in real-time and use the results of that assessment to guide further data collection. By employing this strategy, one can almost always guarantees good data, if it exists.

1.1 Riverine Data Collection

In the aquatic world, scientists and engineers sometimes need to collect data in riverine environments. Currently, **river drifters** are used to accomplish this task. River

Figure 1-1: Example of a 6 inch diameter passive River Drifter. [3]

drifters such as the one shown in Figure 1-1 are *passive* platforms, which means they have no means of propelling themselves in their environment, so they simply float down a river from their insertion point. They are usually equipped with devices such as depth sounders, temperature sensors, water velocity sensors, and other novel sensors which they use to make the measurements of interest. For data collection and storage, they are outfitted with small (usually single-board) computers and data storage devices such as a hard drive or SD card. For communication, they usually have a radio frequency (RF) transceiver, wi-fi radio, and/or satellite radio.

Unfortunately, when deploying these river drifters in the field, scientists and engineers often run into several problems. The two biggest problems are poor vehicle reliability and poor spatial resolution. By using a vehicle with active propulsion, these two problems can be satisfactorily solved. A team consisting of Ocean Sciences Group, Maribotics, and myself completed a feasibility study in which we designed and evaluated a river drifter with active propulsion, the ability to propel itself in its environment, which we called the **Active River Explorer** or ARE. Ocean Sciences Group developed the hardware, Maribotics developed the electronics, and I developed the control behaviors.

To approach the project, I focused my efforts on the two major shortcomings of

passive drifters. First, the passive drifters tend to all follow the same streamline for the majority of the river even when they are spread out along the width of the river upon their entry into the river. This results in a lot of data for one streamline down the river, but not very much data for the rest of the width of the river, thus the current drifters have poor spatial resolution of the river. Second, the passive drifters tend to get caught in the natural debris along the banks of the river. When they get caught in such debris, they are often never recovered. By designing our 'active' drifter or ARE to spend less time along the banks of the river, the survivability of the system will be increased resulting in longer datasets and a higher probability of recovery.

Chapter 2

Simulating Rivers

In order to determine if the developed behaviors improve the utility of the system, example rivers needed to be created in simulation. Then, the performance of the passive drifter and ARE can be compared and the increases in survivability and spatial resolution can be assessed.

2.1 Modeling River Flows

First, a simulation of a riverine environment needed to be created. To do this, I used software called *Mission Oriented Operating Suite* or **MOOS**, a suite of tools developed for maritime robotic development[1].

2.1.1 Mission Oriented Operating Suite

MOOS is an inter-process communication system developed for autonomous vehicle development. The software systems of autonomous vehicles often consist of many processes which run independently but both require information from neighboring processes and provide information that neighboring processes use. MOOS groups processes into communities, then both gives a structure for these processes to run independently and enables this information sharing in a **Publish-Subscribe** manner. Upon initialization, an application will tell MOOS which published variables within

its community it is interested in. Then, this Publish-Subscribe schema provides that application with all information under its variable names of interest. Furthermore, MOOS ensures that all interested applications receive the information that this application publishes.

2.1.2 River Model from Academic Paper

After reviewing papers that described the operation of passive river drifters, I generated a river in simulation that produced similar paths when our ARE was run passively using a MOOS process called *iMarineSim*. iMarineSim is a maritime dynamics simulator which has the ability to take a text file which represents novel water currents as a vector field. To create a computational model of a river, I started by taking an image from Google Maps of a river with an appropriate bends, then creating a piecewise linear approximation of the river banks. I then sampled the space inside the riverbanks and numerically determined the curvature of the river at that point. Finally, I inserted water velocity vectors into iMarineSim at the sampled points whose magnitudes were weighted by that curvature. I adjusted the weights until the path traversed by a passive vehicle in simulation matched that of the drifters tracked in Emery's paper[3].

Figure 2-1 shows the track of two passive drifters in a riverine environment. The vehicles start with different insertion points but converge on the same streamline. While traversing the bends of the river, the drifters are pushed to the outside edge of the river, due to the inertia of the water in the turns.

In Figure 2-2, the river is simulated and the colored dots show the track of the ARE operating passively in simulation.

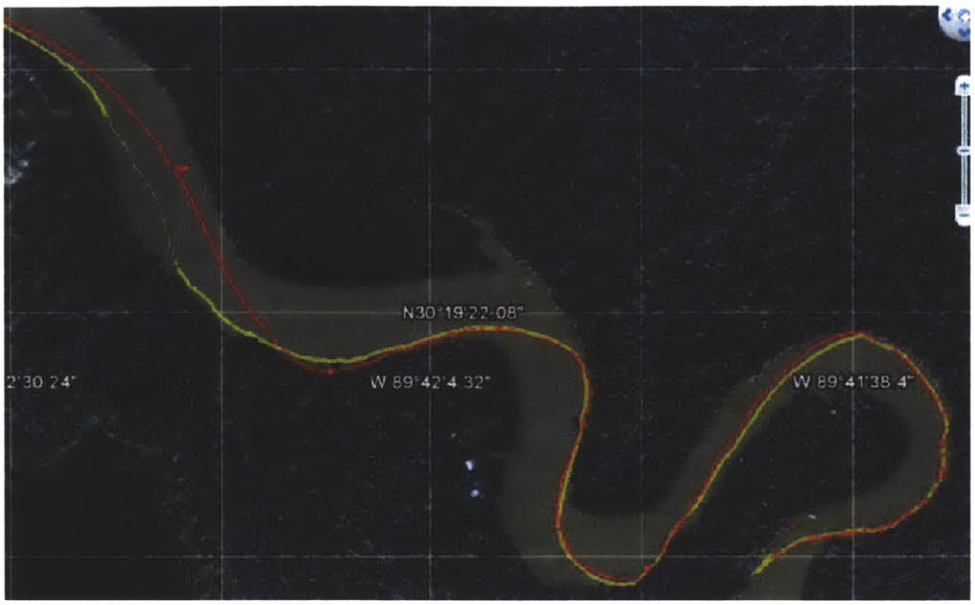

Figure 2-1: Track of two passive drifters in a riverine environment. [3]

Figure 2-2: Track of simulated passive drifter. The color of the dots represents depth. Deep blue represents deeper water, while bright red represents shallower water. Yellow arrows show the direction of local water currents.

As the figures show, the path of the simulated passive drifter closely resembles the path of the passive drifter described in Emery's paper. The simulated drifter is pushed to the outside of the turns when the river bends and follows the flow of the river otherwise. With the simulated passive drifter showing the same behavior as observed passive drifters, a comparison of the ARE and the passive drifter both

in simulation is an acceptable preliminary representation of expectable real world performance of the ARE behaviors developed.

2.2 Modeling River Beds

Along with modeling river currents, I also model river beds to test the thalweg-following behavior. However, unlike river currents, depth profiles of river beds are often highly dissimilar because they tend to be cluttered with debris such as rocks and submerged logs. Therefore, to allow many different depth profiles to be simulated, the simulator simply requires the definition of a function:

$$f(0 \leq x \leq 1) \geq 0 \tag{2.1}$$

This function provides the simulated depth for any point given the percentage of the local river width the vehicle is away from the left river bank. By defining the depth function in this manner, it is robust as the river widens in certain sections and narrows in others and ensures a non-negative depth.

Chapter 3

Active Vehicle Behaviors

3.1 Behavior-based Autonomy

The second half of the Autonomy Software that I used to implement this project is the interval programming based, multi-objective optimization application called **pHelmIvP**[1]. Through this software, the autonomous vehicles are guided in completing certain objectives through *behaviors*. Some examples of behaviors are a waypoint-following behavior, which traverses a set of waypoints in a prescribed order, or a collision-avoiding behavior, which steers the vehicle away from known obstacles. Each behavior guides the vehicle in moving through the environment by providing a *desired heading*, *desired speed*, and/or *desired depth*. The vehicle then invokes its control systems to try to achieve those desired values. To allow for multi-behavior optimization, behaviors provide the process pHelmIvP with a continuous function called an *objective function* that describes the utility that it achieves from various values of desired heading, desired speed, and/or desired depth. pHelmIvP then chooses a value which maximizes the sum of the utility from all of the behaviors that are active.

3.2 ARE Behaviors

To meet the project objectives, two autonomous behaviors were developed. One that substantially increases the survivability of the vehicles and one that substantially in-

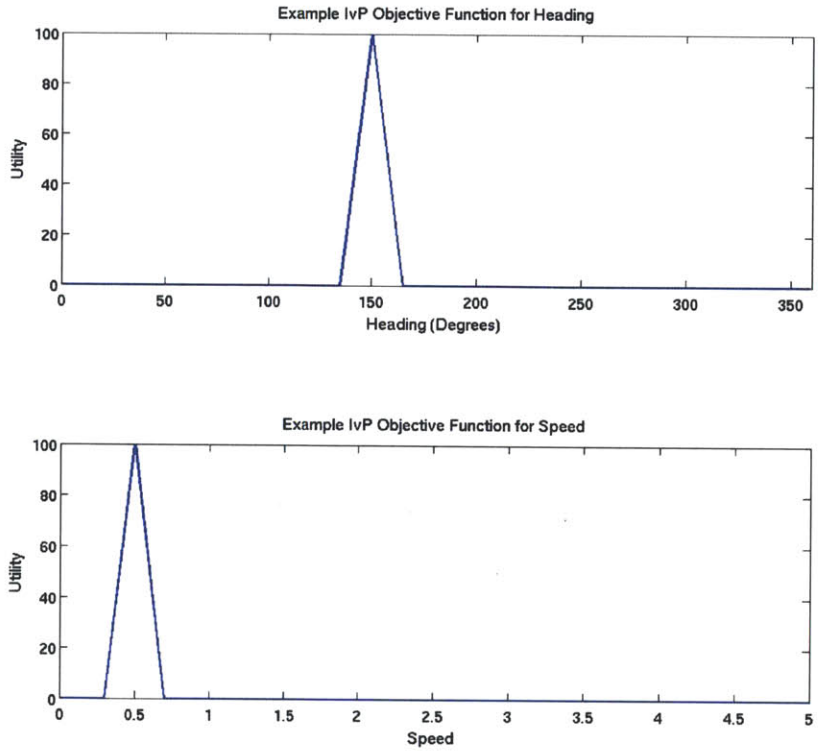

Figure 3-1: Example objective functions used by pHelmIvP for multi-behavior optimization.

creases the spatial resolution of the data.

To achieve the enhanced spatial resolution, a **river exploratory** behavior was designed and developed. This exploratory behavior travels towards either the left or right bank of the river until it comes within a user defined range from that bank, then it turns and heads toward the opposite bank, again until it is within a user defined range of that bank. The behavior repeats this process for the duration of the river or until the behavior is turned off.

To achieve the enhanced survivability, a **thalweg following** behavior was designed and developed. The thalweg following behavior guides the vehicle to navigate the river over the deepest trough for the duration of the river or until the behavior is turned off.

By switching between these two behaviors and tuning the adjustable user defined

parameters, the spatial resolution of the data and the survivability of the system can be tweaked by the user.

3.2.1 River Exploratory Behavior

To improve the spatial resolution of the data obtained by the ARE, the River Exploratory behavior was developed. The River Exploratory behavior guides the vehicle to produce a data set that is a good representation of the depth across the entire width of the river. The behavior starts by aligning the vehicle with the direction the river is flowing. The vehicle is able to determine the direction the river is heading either by having a map of the river and knowing its own position, or by tracking the distances of the right and left river banks from its starboard and port sides. Once the vehicle's heading is within 5 degrees of the river's heading, then it turns to head towards the left bank. The angle at which the vehicle cuts across the river is called the *cutAngle* which the user is able to set. The behavior sets peak utility at an desired heading which is the direction the river is pointing minus the cutAngle. The vehicle traverses across the river at this angle until it gets close enough to the left river bank. When the vehicle is within the distance specified by a user-defined variable called *buffer*, then it turns around and heads toward the right river bank. The desired heading is calculated buy adding the cutAngle to the direction of the local river flow. Once the vehicle gets within the buffer distance to the right river bank, then it turns around to head back to the left river bank and the process continues. Figure 3-2 shows a simulated ARE exploring the riverine environemt using this behavior. The C++ source code for this behavior is included in appendix A.

To implement this behavior, the vehicle requires some solution for shoreline detection. For mapped rivers, the maps can be loaded on to the vehicle before deployment. However, for unmapped rivers, additional sensors whether they are vision based or otherwise, would be needed to implement this behavior on the ARE.

Figure 3-2: Simulated ARE exploring riverine environment.The yellow arrows show local water currents. The colored dots represent places where the vehicle took measurements. The depth of the measurements is on a false color scale. Deep blue indicates deeper water while bright red indicates shallower water.

Tests from Forrest Lake, ME

In November 2010, I tested the behaviors developed for this project on a kayak designed to run MOOS. Although the kayak is not the same form factor as the ARE, it was a readily deployable vehicle capable of maritime autonomy. I performed these tests on Forest Lake in Gray, ME, at one of the facilities where the autonomous kayaks are fabricated. Through testing the behaviors developed in simulation for the ARE on the kayak substitutes, I was able to improve my confidence that the behaviors will perform as expected on the physical ARE as shown in Figure 3-3.

Multi-Vehicle Simulations

The spatial resolution of the system can be increased by using multiple vehicles. The river exploration behavior can be configured to explore any continuous fraction of the width of the river. Therefore, if three vehicles are available, they can each be configured to explore a different third of the river, resulting in a very dense data set across the entire width of the river. This is accomplished by splitting the river into any number of continuous lanes, then assigning diffrent lanes to each of the vehicles.

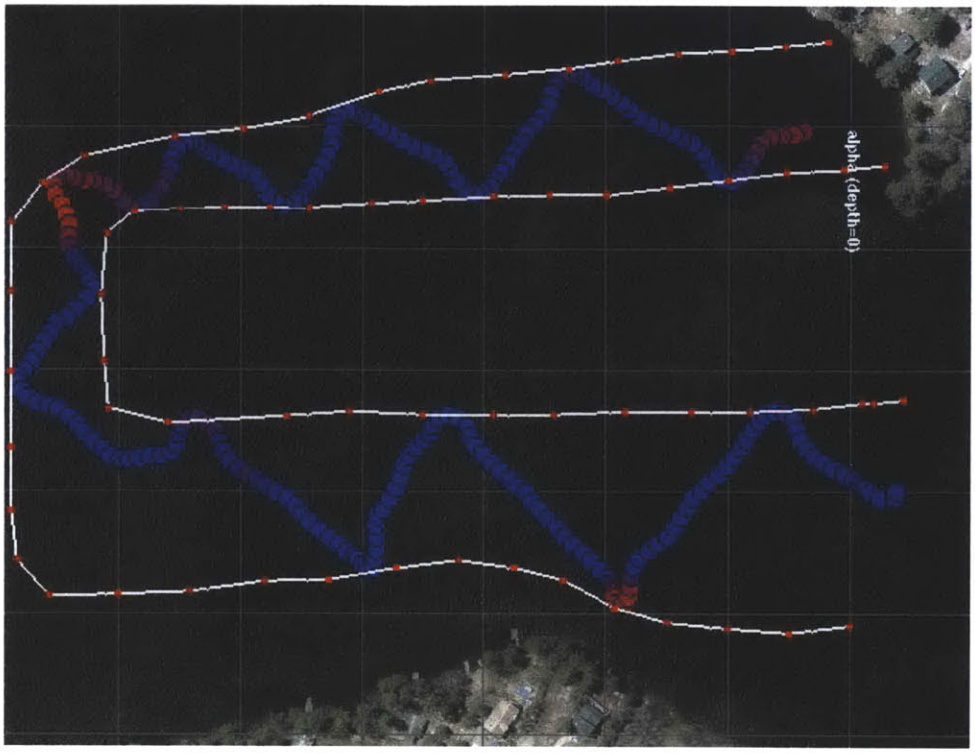

Figure 3-3: Real-World deployment of river exploration behavior. Colored dots represent places where vehicle took measurements. Depth of measurements is on a false color scale. Deep blue indicates deeper water while bright red indicates shallower water. Vehicle title indicates that it was deployed at a depth of 0 or on the lake surface at the onset of the test.

The user indicates how many lanes there are by providing a value for *numLanes*, this effectivly splits the river into a number of equal width adjacent rivers from the vehicle's point of view. Then, the user can assign the vehicle to a specific lane by defining *laneNum*.This tells the vehicle which sub-river it should stay within, counting from the left. Figure 3-4 shows a simulation of four vehicles that are operating simultaneously. The density of colored dots shows the enhanced spatial resolution over the equivalent passive data collection scenario.

3.2.2 Thalweg-Following Behavior

To improve the survivability of the ARE, the thalweg following behavior was developed. The thalweg following behavior finds the deepest trough in the river and follows it for the length of the river or until the behavior is stopped. The passive drifters are

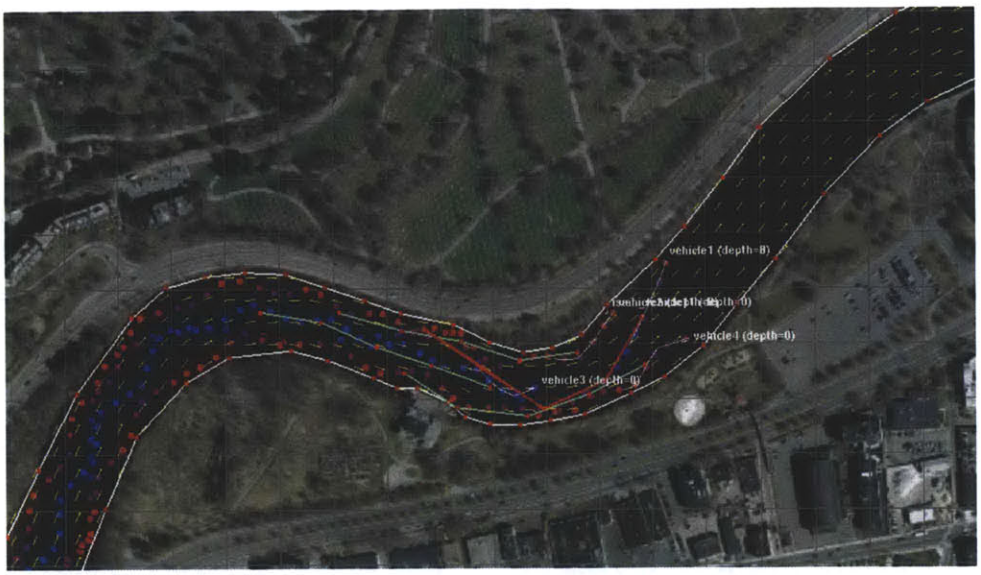

Figure 3-4: Four vehicle river exploration simulation. Vehicle 1 is operating under an exploratory behavior across the entire width of the river. This vehicle displays a red trail to so show its path. Vehicles 2, 3, and 4 are operating under exploratory behaviors as well, however, they are collectively splitting the river into thirds and each exploring a third of the river. Vehicles 2, 3, and 4 all leave light green trails to show their path.

usually lost when they get trapped in dense marine plant growth or natural debris such as fallen branches that reside in the riverine environment. The natural debris and plant growth tend to occur close to the river banks. Therefore, by tracking the deepest trough in the river, we decrease the probability of coming into contact with elements of the environment that can trap the ARE and cause both the hardware loss and prevent the ARE from collecting more useful data. The behavior starts by reading a user defined parameter entitled *approach*, it then generates a desired heading that is the sum of the approach variable and the direction of the local water velocities. Once the actual heading is within five degrees of the desired heading, it monitors the depth of the river to determine whether it is increasing or decreasing. If the depth is increasing, then it continues along a heading that is the sum of the local water velocity and the approach variable. If it is decreasing, then it switches its desired heading to the local water velocity minus the approach variable and again waits for its actual heading to align with its desired heading. This process repeats all the way down the river. The C++ source code for this behavior is included in

Figure 3-5: Simulation of ARE in thalweg-following behavior. Yellow arrows show local water currents. Colored dots represent places where vehicle took measurements. Depth of measurements is on a false color scale. Deep blue indicates deeper water while bright red indicates shallower water.

appendix B. Figure 3-5 shows a simulated ARE following the thalweg of a river. The implementation of this behavior requires knowledge of the depth of the river as well as information on the local water velocities. Using a compass sensor, depth transducer, and water velocity sensor is all that is required to implement this behavior.

Although this behavior works well in simulation, it will likely need some development through situational experience to generate robust performance in the real world. As mentioned earlier, the depth profiles of rivers are highly variable and this algorithm will likely need some tuning to be able to handle complex profiles. Also, a process which reads the raw depth measurements and determines if the actual depth of the river is increasing or decreasing in the presence of both measurement and environmental noise would very much help his behavior achieve robust performance in a real-world deployment. Also, currently, its possible for this process to become fixated on a trough which represents a local minimum while their may be a deeper one representing a global minimum. However, the created behavior is a good first pass at a solution and encapsulates the general strategy.

Chapter 4

Active vs. Passive Performance Comparison

Since drifters tend to get stuck in debris along the banks of the river, then I used the percent of runtime spent close to the river banks as a proxy for the probability of getting stuck. Then, by comparing the percent of time spent within 5m of the banks for the passive drifter, an ARE operating in an exploration behavior, and a ARE operating in a thalweg-following behavior, I compared the expected survivability of the various systems.

Table 4-1 shows a clear improvement in survivability for both the ARE in Explorer Mode and the ARE in Thalweg-following mode. However, by tuning the various user defined parameters and modifing the objective functions, many diffrent levels of performance can be achieved. It is also possible to run both behaviors at the same time and allow the multi-behavior optimization to generate a solution that is a hybrid of

Table 4.1: Vehicle survivability analysis

Type of River Drifter	% of time spent within 5m of banks
Passive	7.84%
ARE Explorer	3.38%
ARE Thalweg Following	0.00%

the two behaviors depending on your needs. Furthermore, even the spatial resolution of the data obtained by the exploratory behavior is greater than that obtained by the passive drifter, and the multi-vehicle dynamics can be tuned to improve the survivability of the AREs in Explorer Mode as well.

Chapter 5

Tracking objects on the Sea floor

The second part of my thesis is aimed at assisting the Laboratory for Autonomous Marine Sensing Systems in developing a multi-static synthetic aperture SONAR approach.

Our globes oceans are a vast wilderness covering well over two-thirds of this planet's surface. Within this seascape lie countless items of interest from shipwrecks to lost cargo to icebergs to mineral deposits. Furthermore, there are dynamic items of interest as well. Often, man-made vessels underway or schools of fish are the targets of a search. As a civilization, we have developed numerous methods for searching for things on land. However, most of these methods, such as looking with one's naked eye, using binoculars or a telescope, or perhaps even utilizing satellite imagery, are based on visual means which are not as effective below the surface of the world's waters. Visibility within the oceans waters is highly variable, but, generally very poor when compared to visibility on land.

SOund Navigation And Ranging or SONAR was developed as a solution to this issue. By utilizing pressure waves rather than high-frequency electromagnetic waves as the spatial sense signal, far greater volumes could be sensed underwater. Rather than the few hundred meters that it is theoretically possible to see underwater[4], SONAR can be used to sense targets miles away. We aim to further increase this performance

by a utilizing a multi-static configuration rather than the nominal mono-static configuration, where the source and receiver are on the same platform. By separating the source and receiver(s) we introduced the possibility for infinitely variable bi-static angles, the angle separating the source and receiver from the target, and enhance the potential spatial resolution and probability of detection of our target.

Chapter 6

Autonomous Underwater Vehicles (AUVs)

In the last few years, Autonomous Underwater Vehicles (AUVs) have developed to the point where their utility has extended from academia to both public and private industry. Previously, it was not possible to send AUVs on complex missions with a high degree of reliability. An entire team was required to support the vehicle as inevitable errors arouse. Currently however, AUVs in the field can reliably complete said complex missions with the support of only one human operator. Furthermore, the vast majority of the operator's time is spent passively observing the AUV after they send the commands.

6.1 Systems on AUVs

AUVs require a number of systems to implement their functionality. The following is a brief overview:

Hull Contains the rest of the systems and protects them from the environment

Propulsion Provides the ability for the AUV to move in a controlled manner

Navigation Allows the AUV to determine where it is

Communication Enables the AUV to communicate with its operators and other vehicles

Computation Provides generic computational resources for arbitrary tasks

Environmental Sensors Allows the AUV to monitor various aspects of its environment

Tools Enables the AUV to manipulate various aspects of its environment(eg. SONAR source)

Together, these systems give the AUV the ability to sense, manipulate and transpose itself through its maritime environment.

6.2 Underwater Navigation

On the surface of the water, the AUVs can utilize the Global Positioning System (GPS) to determine where they are. However, as soon as they go below the surface, GPS no longer functions.

To determine where they are, the AUVs use various navigational sensors. The AUVs that LAMSS operates utilizes Inertial Measurement Units (IMUs), Doppler Velocity Logs (DVLs), and Long Baseline (LBL) transceivers. As with any measurement, each of these devices is prone to measurement errors. Therefore, we have a process called *pNav* which runs in MOOS and fuses all of these measurements with a model for vehicle motion in an Extended Kalman Filter (EKF). However, pNav was developed to operate with a dedicated LBL transceiver where in our experiment we utilized a single acoustic transceiver both for communication with other vehicles and for integrating our vehicle into the LBL network. Since the acoustic transceiver was managed by a process called *pAcommsHandler* which manages all the functionality of the transceiver, then I created a process called *pSplitLBL* which receives information from pAcommsHandler containing all of the LBL ranging data and repackages that

information in a format suitable for pNav to parse and make use of in its calculations. Along with this process, I created a process called *pSimLBL* which generates simulated ranging data for use in vehicle simulations. pSimLBL operates by initializing with knowledge of the locations of all of the LBL nodes, then periodically receiving information about the vehicle's current position, then calculating the acoustic two-way ranging time to each of the nodes, then packaging the result in a format analogous to what pNav would receive during a real mission.

Chapter 7

Multistatic, Synthetic Aperture SONAR

The fundamental idea behind this concept is to separate the source and receiver from a single platform and put them on two or more distinct platforms as shown in Figure 7-1. In the Figure, red beams represent the path of travel of the direct blasts, while green beams represent the path of travel of indirect blasts. From the source platform, the source generates a sound wave whose characteristics are known to all platforms. Typical source waves are an up-chirp or down-chirp where the source emits a tone that is monotonically increasing in frequency or decreasing in frequency. The receivers platforms contain hydrophone arrays. The arrays usually consist of sixteen elements and allow for beam-forming processes to be used on the data which they collect. The beam-forming process enables the platform to determine which direction the sound wave came from relative to the heading of the platform. The platforms themselves are fully capable AUVs. These autonomous underwater vehicles are equipped with sophisticated navigation, communication, and propulsion systems as well as numerous environmental sensors. With these attributes, they can be commanded to do things like traverse pre-defined geometries, trail a moving target, and/or avoid collision scenarios. Furthermore, these vehicles can asses a situation with their sensor suite and modify their behavior based on that situation.

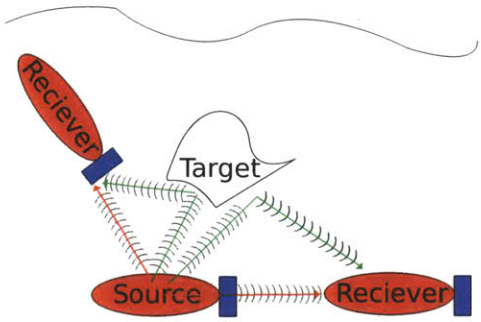

Figure 7-1: Sketch of three AUVs implementing Multi-static Synthetic Aperture SONAR.

7.1 SONAR Processing Tool-chain Overview

Our sonar processing tool-chain consists of several discrete stages. First, the acoustic signal is digitized by the hydrophone array and stored electronically. Then, those electronic files are read and a beam-forming[5] process is completed. After the signals are split-up into discrete beams, then a matched filter[8] is run against a local copy of the acoustic source file. The results from the matched filter are normalized[7] to reduce environmental noise, then those results are put through an automatic thresholding[7] process. Finally, a process is used to select the direct(wave arriving directly from the source) and indirect(wave reflected off of the target) blasts from the data if they are discernible. We compile the results into a **contact report** which details the angle of incidence, time of arrival, and intensity of the direct and indirect blasts. The tool-chain is diagrammed in Figure 7-2, where wiring is shown between the various modules of the chain, starting with the individual hydrophone data and terminating with the contact report. (The normalization process is bundled within the thresholding module.)

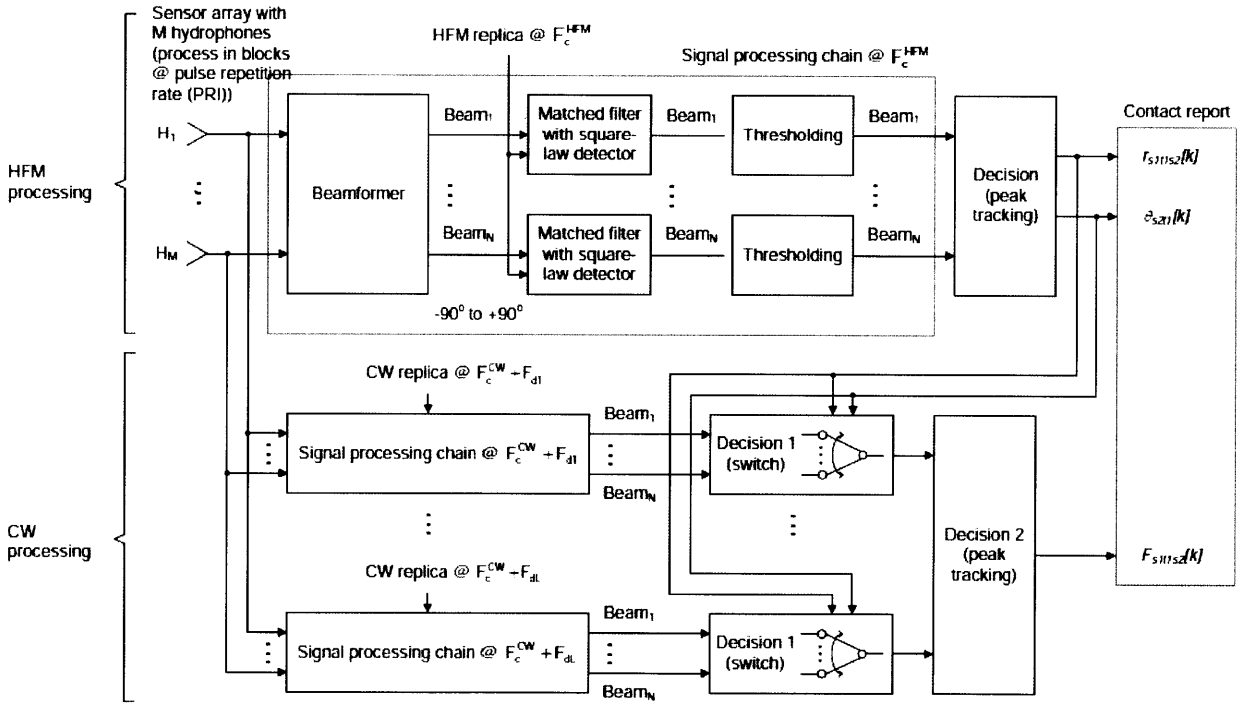

Figure 7-2: Diagram of SONAR Processing tool-chain[6].

Chapter 8

Shallow Water Autonomous Mine Sensing Initiative (SWAMSI '11) Experiment

The Shallow Water Autonomous Mine Sensing Initiative 2011 (SWAMSI '11) Experiment occurred January 2011 on the Naval Undersea Research Centre (NURC) facility in La Spezia Harbor, Italy. The purpose of SWAMSI is to detect and classify seabed and sub-seabed targets.

8.1 Experimental Setup and Assets

8.1.1 Hardware

The major assets in this experiments included two Bluefin21 AUVs with forward-mounted hydrophone arrays, an OEX Harpo AUV with an acoustic source and towed array, an Long Baseline (LBL) net, and a shore side command center with an acoustic modem and transceiver.

The first Bluefin21 AUV, Unicorn, has a 'single' array with 16 hydrophone elements mounted in a co-linear fashion as shown in Figure 8-1. Our second Bluefin21 AUV, Macrura, has a 'dual' array which consists of two parallel 8 co-linear element

Figure 8-1: Bluefin21 Unicorn AUV with 'single' array

sections, shown in Figure 8-2. Both of the hydrophone arrays contain elements and electronics optimized for processing 7-8 kHz sound waves. The last AUV, shown in Figure 8-3, Harpo (owned and supported by NURC), was equipped with the TOSSA acoustic source and SLITA towed array. The TOSSA source was not mounted in the Figure, but was used as the acoustic source for this experiment due to the failure of our primary source. The TOSSA was designed to produce 3-4 kHz waves where our original higher frequency source was designed for 7-15 kHz operation.

To aid with underwater navigation, we deployed a Long BaseLine or LBL net. By placing several LBL Beacons at known locations and measuring the round-trip ranging times of sound waves between the vehicles and the beacons, we are able to enhance our position estimates.

We use an acoustic communication network to send our vehicles mission commands and specifications as well as enable our vehicles to transmit information about their position and status to the control center and other vehicles. Our link to that communication network is an acoustic modem on the shore and transceiver in the water beside the shore.

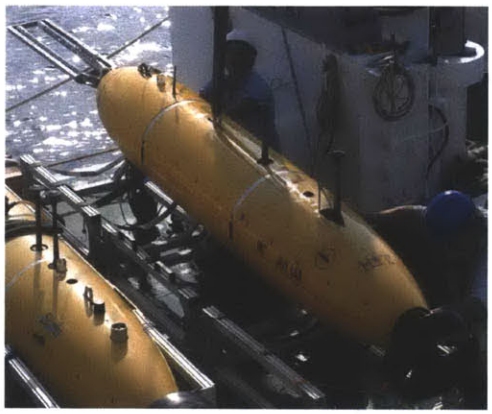

Figure 8-2: Bluefin21 Macrura AUV with 'dual' array

Figure 8-3: OEX Harpo AUV with SLITA array

8.1.2 Software

All of the AUVs use the MOOS-IvP autonomy software discussed earlier. As mentioned, this software provides both the publish-subscribe communication architecture for the various autonomy processes that run within a vehicle and the multi-behavior optimization capabilities that effectivly provide autonomous decision making.

8.1.3 Communication

The vehicles are capable of using wifi communications while on the surface of the water, however, underneath the surface, wifi is ineffective. To enable communications underwater, all of the vehicles are equipped with an acoustic communication system which utilizes the WHOI Micromodem.

8.2 Data Processing

During the expriment we collected all of the raw data from each of the hydrophone arrays. The data consisted of a timeseries of values representing the hydrostatic pressure levels which each of the hydrophone elements measured. To determine the location of the targets from this data required significant processing. I acheived this by first invoking a process called *pActveSonar* which performed the necessary matched filtering and beamforming to generate the contact report. Using a Perl script, I parsed the contact reports and the navigation logs for the vehicles and assembled all of the data into a Comma-Sperated-Variable (CSV) list which I could import into MATLAB. Then in MATLAB, I solved for the target solutions using the bistatic methods outlined by Henery Cox[2]. The full text of the MATLAB script is provided in appendix C.

8.3 Experimental Results

During the experiment, we are able to visually track the locations of all the vehicles from the command center. Figure 8-4 shows a screenshot from the monitor dedicated to the observation of the vehicle positions on Jan 20 2011. The yellow trail represents the path of the Unicorn AUV, while green and orange display the paths of Macrura and OEX Harpo respectively. The times beside the vehicle labels indicate the amount of time since their last position update over the acoustic network. The positions displayed for Macrura, Unicorn and OEX Harpo are 2 minutes 6 seconds old, 30 seconds old, and 53 seconds old respectively. The sphere, rock, and rockan are underwater targets which we are trying to localize. The LBL Beacons which form the LBL net are also labeled.

After post-processing the SONAR returns to determine the location of the target, images such as Figure 8-5 and Figure 8-6 were produced. Unfortunately, due to the mismatch between array parameters and source frequency after the failure of our primary source, the beam-forming process returned standard deviations around 145

Figure 8-4: Screenshot of monitor displaying vehicle tracks

degrees. Effectively, the angles that it returned were ambiguous. That coupled with the diminished signal-to-noise ratio due to the prominent acoustic reflections from the seawall, made this a very hard dataset. In Figures 8-5 and 8-6, the black asterisks are the targets, the cyan dot is the receiving platform, the green dot is the source platform, the white dots form the equi-time-of-arrival ellipse, and the red dots are the beam-forming solution and its shadow (due to left-right beam-forming ambiguity). Figure 8-7 shows the compilation of many returns. The blue dots represent the beam-former located returns, while the targets are again black asterisks. Due to the inaccuracy of the beam-former, it is highly likely that all of the returns are originating from the sea wall.

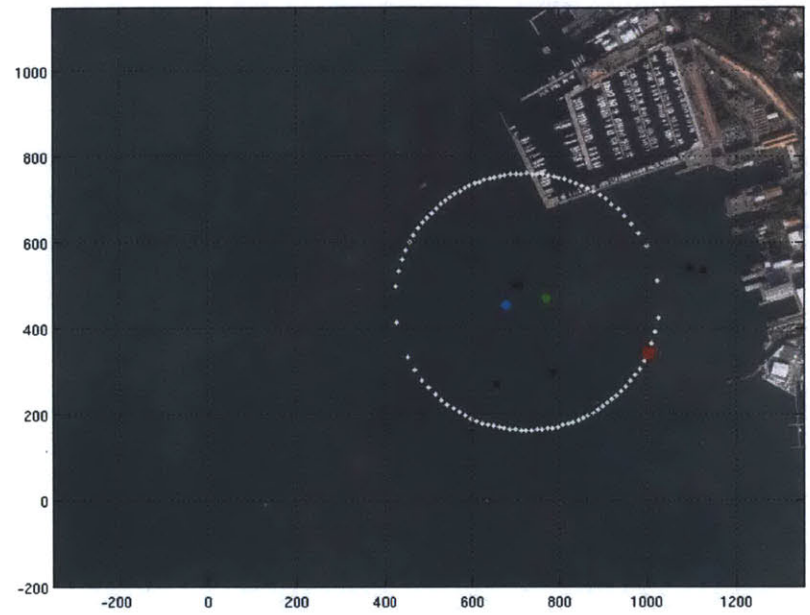

Figure 8-5: Result from single SONAR return

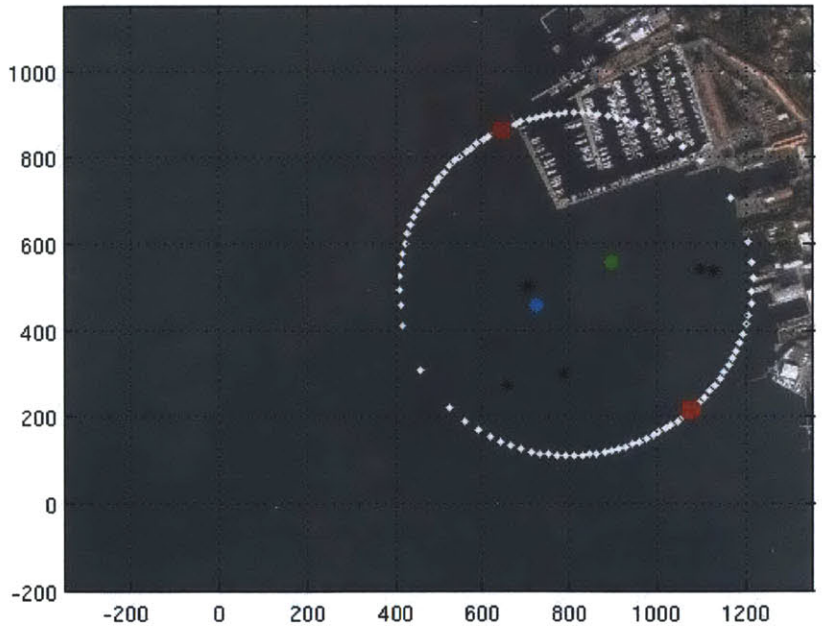

Figure 8-6: Result with shadow from single SONAR return

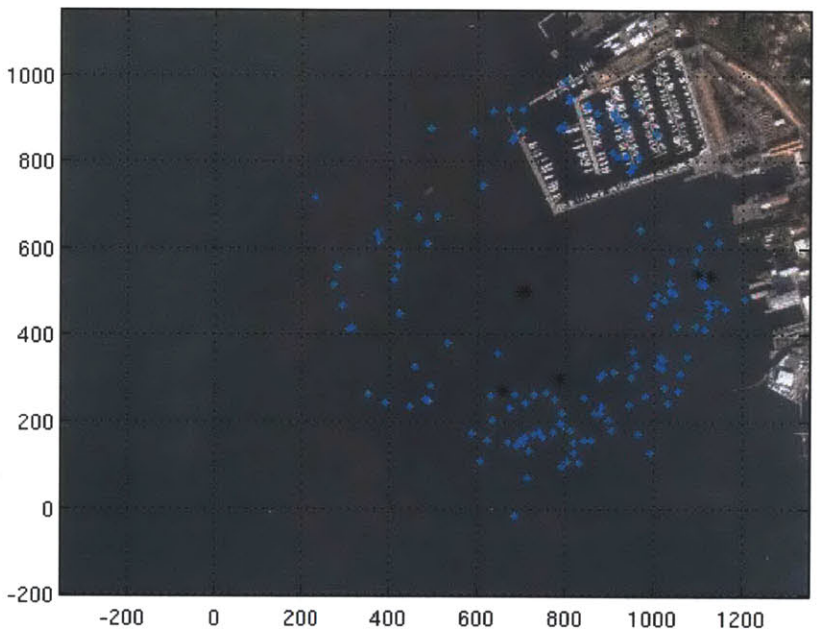

Figure 8-7: Results from multiple SONAR returns

Chapter 9

Conclusions

9.1 Autonomous Riverine Data Collection Behaviors

The behaviors I developed greatly improve over the current state of the art. By intelligently navigating the rivers using the thalweg-following and river exploratory behaviors which I devised and developed, a very rich data set can be produced with greatly lowered probabilities of vehicle entrapment. The enhanced survivability of the AREs means that longer sections of river can be explored and logged in a single run. Furthermore, I was able to test and verify the River Exploratory behavior in Forrest Lake, ME.

9.2 SWAMSI '11

During this experiment I was able to configure pNav, and I created a process which allowed pNav to receive LBL data through the hardware configuration that we used in the experiment. I created a process to simulate LBL data for vehicle testing and I post processed the SONAR data using pActiveSonar, the Perl programming language, and MATLAB.

Although it was unfortunate that our primary acoustic source failed, we are grateful that we could collect some data using the TOSSA source. Hopefully we will be able to have our source repaired and demonstrate the functionality of this Multi-static, Synthetic Aperture SONAR method. After successfully tracking the targets, we would like to develop behaviors which autonomously guide the vehicles to optimize their bi-static angle such that target localization and identification is idealized.

Appendix A

Source Code for River Exploratory Behavior

```
#ifdef _WIN32
#pragma warning(disable : 4786)
#pragma warning(disable : 4503)
#endif
#include <math.h>
#include <stdlib.h>
#include "BHV_Explore.h"
#include "MBUtils.h"
#include "BuildUtils.h"
#include "ZAIC_PEAK.h"
#include "OF_Coupler.h"
#include "compassMath.h"

using namespace std;

//-----------------------------------------------------------
// Procedure: Constructor
```

```
BHV_Explore::BHV_Explore(IvPDomain gdomain) :
  IvPBehavior(gdomain)
{
  this->setParam("descriptor", "bhv_explore");
  m_domain = subDomain(m_domain, "course,speed");

  // Behavior Parameter Default Values:
  currentState = ALIGN;
  approach = 15; //keep track of whether we heading toward or away from the thalweg
  river = 180;
  compass = 0;
  oldDepth = 0;
  cutAngle = 45;
  laneNum = 1;
  numLanes = 1;
  riverEnd = 0;
  heading = 0;
  thrust = 20;
  realRight = 0;
  realLeft = 0;
  buffer = 5;


  // Declare the variables we will need from the info_buffer
  addInfoVars("DIST_TO_LEFT");
  addInfoVars("DIST_TO_RIGHT");
  addInfoVars("RIVER_HEADING");
  addInfoVars("NAV_HEADING");
  addInfoVars("DEPTH");
```

```
  addInfoVars("CUT_ANGLE");
  addInfoVars("RIVER_END");

}

//-----------------------------------------------------------
// Procedure: onIdleState
//      Note: This function overrides the onIdleState() virtual
//            function defined for the IvPBehavior superclass
//            This function will be executed by the helm each
//            time the behavior FAILS to meet its run conditions.

void BHV_Explore::onIdleState(){}


//-----------------------------------------------------------
// Procedure: setParam

bool BHV_Explore::setParam(string g_param, string g_val)
{
  if(IvPBehavior::setParamCommon(g_param, g_val))
    return(true);

  g_val = stripBlankEnds(g_val);

  if(g_param == "cutAngle")
    {
      double dval = atof(g_val.c_str());
      if(!isNumber(g_val))
cutAngle=45;
```

```
      else
cutAngle = dval;
      return(true);
    }
  if(g_param == "laneNum")
    {
      double dval = atof(g_val.c_str());
      if((dval <= 0) || (!isNumber(g_val)))
laneNum=1;
      else
        laneNum = dval;
      return(true);
    }
   else if(g_param == "numLanes")
     {
       double dval = atof(g_val.c_str());
       if((dval < 0) || (!isNumber(g_val)))
 numLanes=1;
       else
 numLanes = dval;
       return(true);
     }
   else if(g_param == "buffer")
     {
       double dval = atof(g_val.c_str());
       if((dval < 0) || (!isNumber(g_val)))
 buffer=5;
       else
 buffer = dval;
       return(true);
```

```
    }

  return(true);
}

//-----------------------------------------------------------
// Procedure: onRunState

IvPFunction *BHV_Explore::onRunState()
{
  double gain = 1.5;
  //double buffer = 5;
  double pLeft = (laneNum-1)/numLanes;
  double pRight = laneNum/numLanes;
  // CompassMath rose;

  //UPDATE VARS
  bool ok;
  //map raw heading to a compass rose
  double raw = getBufferDoubleVal("RIVER_HEADING", ok);
  if (ok){
    if (raw > 0) {river=raw;}
    else {river=360+raw;}
  }

  raw=getBufferDoubleVal("NAV_HEADING", ok);
  //if the value is negative, then map it to a compass rose
  if (ok){
  compass=rose.Add(raw,0);
  }
```

```
raw=getBufferDoubleVal("CUT_ANGLE", ok);
if (ok){cutAngle=raw;}

if (getBufferDoubleVal("RIVER_END", ok) == 1){riverEnd = 1;}
else {riverEnd = 0;}


raw = getBufferDoubleVal("DIST_TO_LEFT", ok);
if(ok){realLeft=raw;}

raw = getBufferDoubleVal("DIST_TO_RIGHT", ok);
if(ok){realRight=raw;}


  //-> ALIGN WITH RIVER
if (currentState == ALIGN){
  if (rose.Add(compass, (-1*river)) < 5 || rose.Add(compass, (-1*river)) > 345){cur
  else {
    heading=river;
    thrust=5;
    //cout << "aligning, diff is: " << rose.Add(heading, (-1*compass)) << endl;
  }
}

//-> GO TOWARDS THE LEFT RIVER BANK
if (currentState == HEADING_LEFT){
  double dist = realLeft;
  double width = realLeft + realRight;
  dist = dist - (pLeft*width);
```

```
    cout << "width is: " << width << ", bounds are: " << pLeft << " and " << pRight <

    if (dist < buffer){currentState=HEADING_RIGHT;} //-> were close enough to the lef
    if (dist < 2*buffer) {thrust = 10; heading = rose.Add(river ,(-1*cutAngle));} //-
    else {
      heading = rose.Add(river ,(-1*cutAngle));
      thrust = 20;
      //cout << "heading left, dist is: "<< dist << endl;
      }
  }

    //-> GO TOWARDS THE RIGHT RIVER BANK
    else if (currentState == HEADING_RIGHT){
      double dist = realRight;
      double width = realLeft+realRight;
      dist = pRight*width - (width-dist);
      cout << "width is: " << width << ", bounds are: " << pLeft << " and " << pRight
      if (dist<buffer){currentState=HEADING_LEFT;}
      if (dist < 2*buffer) {
thrust = 10;
heading = rose.Add(river,cutAngle);
      }
      else {
         heading = rose.Add(river,cutAngle);
         thrust = 20;
         //cout << "heading right, dist is: " << dist << endl;
      }
    }

    //generate rudder command
```

```
double headingError = rose.Add(heading, (-1*compass));

//translate from rose angles to polar angles
headingError = rose.Polar(headingError);


//generate heading function
ZAIC_PEAK head_ziac(m_domain, "course");
head_ziac.setSummit(heading);
head_ziac.setPeakWidth(15);
head_ziac.setBaseWidth(10);
head_ziac.setSummitDelta(100);
head_ziac.setValueWrap(true);

IvPFunction *head_ipf = head_ziac.extractIvPFunction();


if (headingError*headingError > 100) {thrust = 20;} //were off by > 10 deg, slow
if (riverEnd == 1) {thrust=0; cout << "river ended, stopping" <<endl;}
//m_Comms.Notify("DESIRED_THRUST",thrust);

//generate speed function
ZAIC_PEAK spd_ziac(m_domain, "speed");
spd_ziac.setSummit(thrust/20);
spd_ziac.setPeakWidth(0.2);
spd_ziac.setBaseWidth(0.05);
spd_ziac.setSummitDelta(100);

IvPFunction *spd_ipf = spd_ziac.extractIvPFunction();
```

```
    //couple functions
    OF_Coupler coupler;
    IvPFunction *ivp_function = coupler.couple(head_ipf, spd_ipf,50,50);

    return(ivp_function);
}
```

Appendix B

Source Code for Thalweg Following Behavior

```
#ifdef _WIN32
#pragma warning(disable : 4786)
#pragma warning(disable : 4503)
#endif
#include <math.h>
#include <stdlib.h>
#include "BHV_Thalweg.h"
#include "MBUtils.h"
#include "BuildUtils.h"
#include "ZAIC_PEAK.h"
#include "OF_Coupler.h"

using namespace std;

//-----------------------------------------------------------
// Procedure: Constructor

BHV_Thalweg::BHV_Thalweg(IvPDomain gdomain) :
```

```
  IvPBehavior(gdomain)
{
  this->setParam("descriptor", "bhv_thalweg");
  m_domain = subDomain(m_domain, "course,speed");

  // Behavior Parameter Default Values:
  currentState = FOLLOWING_THALWEG;
  approach = 15;
  river = 0;
  compass = 0;
  oldDepth = -1000;
  laneNum = 1;
  heading = 0;
  thrust = 20;
  oldError = 100;
  depthCount = 0;
  depthPeak = 0;
  CompassMath rose;


  // Declare the variables we will need from the info_buffer
  addInfoVars("RIVER_HEADING");
  addInfoVars("NAV_HEADING");
  addInfoVars("DEPTH");
  addInfoVars("RIVER_END");

}

//-----------------------------------------------------------
// Procedure: onIdleState
```

```
//      Note: This function overrides the onIdleState() virtual
//            function defined for the IvPBehavior superclass
//            This function will be executed by the helm each
//            time the behavior FAILS to meet its run conditions.

void BHV_Thalweg::onIdleState(){}


//-----------------------------------------------------------
// Procedure: setParam

bool BHV_Thalweg::setParam(string g_param, string g_val)
{
  if(IvPBehavior::setParamCommon(g_param, g_val))
    return(true);

  g_val = stripBlankEnds(g_val);

  if(g_param == "approach")
    {
      double dval = atof(g_val.c_str());
      if(!isNumber(g_val))
approach=15;
      else
approach = dval;
      return(true);
    }

  return(true);
}
```

```
//-----------------------------------------------------------
// Procedure: onRunState

IvPFunction *BHV_Thalweg::onRunState()
{
  double gain = 1.5;
  CompassMath rose;
  thrust = 20;

  //UPDATE VARS
  bool ok;
  //map raw heading to a compass rose
  double raw = getBufferDoubleVal("RIVER_HEADING", ok);
  if (ok){
    if (raw > 0) {river=raw;}
    else {river=360+raw;}
  }


  raw=getBufferDoubleVal("NAV_HEADING", ok);
  if (ok){compass=raw;}
  //if the value is negative, then map it to a compass rose
  compass=rose.Add(compass,0);

  if ((getBufferDoubleVal("RIVER_END", ok) == 1) && ok){riverEnd = 1;}
  else {riverEnd = 0;}

  double depth = oldDepth;
  raw = getBufferDoubleVal("DEPTH", ok);
```

```
if (ok) {depth=raw;}
else {cout << "bad depth value" << endl;}

//if (depth >= oldDepth){depthPeak=depth;}
//else {depthCount = depthPeak - depth;}

double headingError = rose.Polar(rose.Add(heading, (-1*compass)));

if (depth-oldDepth >=0 ){
  //Depth is increasing so continue on heading -> not robust
  heading = rose.Add(river,approach);
}
else {
  //cout << "depth is decreasing, heading error: " <<  headingError << endl;
  //Depth is decreasing so change heading realtive to the river
  if(abs(rose.Polar(rose.Add(heading,-1*compass))) < 5){
    //make sure we are along our desired heading, else wait until we are
    //then change desired heading if still causing decreasing depth
    approach = -1*approach;
    heading = rose.Add(river,approach);
    //cout << "switched heading" << endl;
  }
  else {
    heading = rose.Add(river, approach);
  }
  //depthPeak = depth;
  //cout << "turning" << endl;
}

oldDepth = depth;
```

```
oldError = headingError;
//cout << "depthPeak is: " << depthPeak << " , depthCount is: " << depthCount << en

//translate from rose angles to polar angles
headingError = rose.Polar(headingError);

//generate heading function
ZAIC_PEAK head_ziac(m_domain, "course");
head_ziac.setSummit(heading);
head_ziac.setPeakWidth(15);
head_ziac.setBaseWidth(10);
head_ziac.setSummitDelta(100);
head_ziac.setValueWrap(true);

IvPFunction *head_ipf = head_ziac.extractIvPFunction();


// if (headingError*headingError > 100) {thrust = 10;}
if (riverEnd == 1) {thrust=0; cout << "river ended, stopping" <<endl;}

//generate speed function
ZAIC_PEAK spd_ziac(m_domain, "speed");
spd_ziac.setSummit(thrust/20);
spd_ziac.setPeakWidth(0.2);
spd_ziac.setBaseWidth(0.05);
spd_ziac.setSummitDelta(100);

IvPFunction *spd_ipf = spd_ziac.extractIvPFunction();

//couple functions
```

```
  OF_Coupler coupler;
  IvPFunction *ivp_function = coupler.couple(head_ipf, spd_ipf,50,50);

  return(ivp_function);
}
```

Appendix C

MATLAB code for implementing Bi-static SONAR Target Localization

```
function [solns] = target_loc(data, varargin)

%read data
%check to see if two consecutive pings correspond
%if they do, do math for target localization

solns=[];
sol_index = 1;
sound_speed = 1507;

bounds = size(data);

rocks = [700 500;710 500;1126 537;1096 540;658 271;787 298];
min_x = -350; max_x = 1350; min_y = -200 ; max_y = 1150;

%read the background image file
```

```
img = imread('lotti_edit.png');

for i=2:bounds(1,1)
    if (data(i,9) ~= -1 & data(i-1,9) ~= -1 & data(i-1,5)>data(i,5)+9)%data(i-1,5) -
        diff_vector = data(i,6:7) - data(i,10:11);
        R3 = sqrt(diff_vector*diff_vector');
        Rm = (((data(i,1) - data(i-1,1))*sound_speed)-R3)/2;
        target_bearing_A = data(i,13) + data(i,3);
        target_bearing_B = data(i,13) - data(i,3);

        angRcvSrc = atan2(data(i,7)-data(i,11), data(i,6)-data(i,10));
        gamma_A = compassToRad(target_bearing_A) - angRcvSrc;
        gamma_B = compassToRad(target_bearing_B) - angRcvSrc;

        R2_A = Rm * ((1+(R3/Rm))/(1+ (R3/Rm)*(1-cos(gamma_A))*0.5));
        R2_B = Rm * ((1+(R3/Rm))/(1+ (R3/Rm)*(1-cos(gamma_B))*0.5));

        %output data format
        %time, time_sigma, target_x, target_y, ang_sigma

        solns(sol_index, 1) = data(i,1);
        solns(sol_index, 2) = data(i,2);
        solns(sol_index, 3) = data(i,10)+R2_A*cos(angRcvSrc+gamma_A);
        solns(sol_index, 4) = data(i,11)+R2_A*sin(angRcvSrc+gamma_A);
        solns(sol_index, 5) = data(i,4);

        solns(sol_index+1, 1) = data(i,1);
        solns(sol_index+1, 2) = data(i,2);
        solns(sol_index+1, 3) = data(i,10)+R2_B*cos(angRcvSrc+gamma_B);
        solns(sol_index+1, 4) = data(i,11)+R2_B*sin(angRcvSrc+gamma_B);
```

```
        solns(sol_index+1, 5) = data(i,4);

        sol_index = sol_index+2;
        if (length(varargin)>0 & sol_index>2)

            %calculate ellipse of solutions (assume no anglular info)
            major = (Rm*2)+R3;
            minor = sqrt((major^2)-R3^2);
            x_c = (data(i,6)+data(i,10))/2;
            y_c = (data(i,7)+data(i,11))/2;

            locus_ellipse = make_ellipse(major/2, minor/2, angRcvSrc, x_c, y_c);

            hold off;
            imagesc([min_x max_x],[min_y max_y],flipdim(img,1));
            hold on;
            plot(data(i,6),data(i,7),'.g','MarkerSize',20); %src position
            axis([min_x max_x min_y max_y]);
            plot(rocks(:,1),rocks(:,2),'*k');
            plot(data(i,10),data(i,11),'.c','MarkerSize',20); %reciever position
            plot(solns(sol_index-2:sol_index-1,3),solns(sol_index-2:sol_index-1,4),'.
            plot(locus_ellipse(:,1),locus_ellipse(:,2),'.w'); %sol for ambiguous angl
            set(gca,'ydir','normal');
            grid on;
            pause(1.0);
        end

    end

end
```

```
%From Fundamentals of Bistatic Sonar, Henry Cox
%Rm = (R1+R2-R3)/2;
%R2 = Rm * ((1+(R3/Rm))/(1+ (R3/Rm)*(1-cos(gamma))*0.5));
end


function[rad] = compassToRad(compass)
    rad = (pi/2) - (pi/180)*compass;
end
```

Bibliography

[1] M Benjamin, H Schmidt, P Newman, and J Leonard. *An Overview of MOOS-IvP and a Users Guide to the IvP Helm - Release 4.2.1.* MIT-CSAIL, 2011. http://hdl.handle.net/1721.1/65073.

[2] Henry Cox. Fundamentals of bistatic active sonar. In Y T Chan, editor, *Underwater Acoustic Data Processing*, pages 3–24. Kluwer Academic Publishers, 1989.

[3] L Emery. Drifting buoy for autonomous measurements of river environment. *OCEANS 2009, MTS/IEEE Biloxi*, 2009.

[4] J Jaffe. Computer modeling and the design of optimal underwater imaging systems. *IEEE Journal of Oceanic Engineering*, 15(2), 1990.

[5] D Johnson and D Dudgeon. *Array Signal Processing - Concepts and Techniques.* Prentice Hall, 1993.

[6] R Lum. Multistatic processing and tracking of underwater target using autonomous underwater vehicles. *Underwater Acoustic Measurements: Technologies and Results*, 2009.

[7] R Nielsen. *Sonar Signal Processing.* Artech House, 1991.

[8] H Van Trees. *Detection, Estimation, and Modulation Theory - Part III - Radar-Sonar Processing and Gaussian Signals in Noise.* John Wiley and Sons, 2001.