Title
Calibration and Environment Characterization for Autonomous Aquatic Actuated Sensing

Author
Chen, Victor L

Publication Date
2008-12-01

University of California
Los Angeles

Calibration and Environment Characterization for Autonomous Aquatic Actuated Sensing

A thesis submitted in partial satisfaction
of the requirements for the degree
Master of Science in Electrical Engineering

by

Victor Liu Chen

2008

The thesis of Victor Liu Chen is approved.

Gregory J. Pottie

Mark H. Hansen

William J. Kaiser, Committee Chair

University of California, Los Angeles

2008

Table of Contents

1 Introduction **1**

1.1 Overview 1

1.2 Contribution of Thesis 2

2 Overview of NIMS Systems **4**

2.1 NIMS-RD 4

2.1.1 Remote Sensors Interface 5

2.2 NIMS-AQ 9

2.2.1 Remote Access and Control 11

2.3 Water Quality Sensors 11

3 Horizontal Calibration and Localization For Cabled Systems . **14**

3.1 The Effects of Sag on NIMS RD 14

3.2 The Hall Effect And Its Application to Horizontal Localization . . 18

3.3 Hardware Implementation 19

3.4 Software Implementation 24

4 Sonar Preliminaries **26**

4.1 Overview 26

4.2 Sonar Signal Processing 28

5 Vertical Calibration and Localization for Cabled Systems . . . **36**

5.1 Overview 36

5.2 Previous Methods for Calibration 37
5.3 Range Estimation 39
5.4 System Integration 42

6 Environmental Characterization with Miniature Sonar 44
6.1 Spatial Characterization 44
6.1.1 Aquatic Laboratory Subsurface Reconstruction 45
6.1.2 Lake Subsurface Reconstruction 45
6.1.3 Occupancy Grid Mapping 49
6.2 Semantic Characterization 53
6.2.1 Mesocosm Experiments 56
6.2.2 Field Experiment 60

7 Conclusion and Future Work 65

A Source Code 67
A.1 relay.py 67
A.2 adc.py 68
A.3 hall.py 74
A.4 sonar.py 79
A.5 gridmap.py 87

References 94

List of Figures

2.1 Schematic drawing of the NIMS-RD platform 5

2.2 Schematic of the SensorBox showing the power and data connections among the various components as well as the inputs from the shore and outputs to the sensors 7

2.3 Schematic of the SensorBox showing the power and data connections among the various components as well as the inputs from the shore and outputs to the sensors 8

2.4 Schematic drawing of the NIMS-AQ environmental sensing platform 10

2.5 Aquatic sensors used on NIMS-RD and NIMS-AQ 12

3.1 Horizontal position error due to sag displacement (not drawn to scale) 14

3.2 Percent error in cable position due to cable sag 16

3.3 The Hall effect 18

3.4 Hall effect sensor from Sensor Solutions 19

3.5 Modular housing attached to the shuttle with calibration cable, guide pulley, and hall sensors shown 22

3.6 Magnet arrangement representing 111101 and the corresponding output signal 23

4.1 Miniature, ultrasonic scanning sonar developed by Imagenex Corp 27

4.2 Procedure for estimation of range from example reflection signal after application of each filter. The raw reflection signal is shown in Figure(a). Figure(b) shows the reflections signal superimposed with the cubic spline smoother and the local maxima identified. Figures (c)-(e) show the signal after application of the NN, MR, and ER filters respectively. 33

4.3 Example of how cluster analysis is used to produce a more accurate range estimate using range estimates from multiple sonar readings 35

5.1 Colormaps of subset of (a) range estimates from sonar to water surface, (b) range estimates from sonar to lake bottom, (c) and full depth of water column 40

5.2 Satellite map of the UC Merced campus lake with experimental transect labeled and NIMS AQ operating on the lake with the sonar and other sensors attached 41

5.3 Comparison of depth measurements from sonar, Hydrolab, and tape measure 42

5.4 Depth profile of transect executed at Lake Fulmor using autonomous depth profiling 43

6.1 Obstacle arrangement and surface reconstruction from pool mapping experiment 46

6.2 Example of the spatial map acquired using the NIMS-AQ platform on a part of the executed transect. The depth reported by the sonar as indicated by the surface color is consistent with the position of the z-axis. 47

6.3 Resulting points clouds (dark points) after applying NN, MR, and ER filters to the unfiltered point cloud (light points) 48
6.4 Obstacle arrangement for gridmapping experiment 51
6.5 Top view surface reconstructions from grid mapping experiment performed in swimming pool (note: x- and y-axes are not proportional) 52
6.6 Groundwater system involving the hyporheic zone 54
6.7 Echograms from buckets filled with sand and clay 56
6.8 Sonar output signals from sediment comparison experiment . . . 59
6.9 Sonar output signals for sand and clay from large mesocosm experiments 60
6.10 Reflection coefficient distribution along the lake transect. 61
6.11 Soil core sampler 62
6.12 Echogram from part of the executed transect at the UC Merced campus lake 63
6.13 Sediment samples extracted from the transect at the UC Merced campus lake 64

List of Tables

2.1 Sensors commonly used for NIMS-RD aquatic deployments 13

6.1 Number of points in the point cloud and percent reduction in size after application of nearest neighbors (NN), multiple reflections (MR), expected range (ER) filters. 49

6.2 Size of occupancy grid for each scan 51

6.3 Qualitative summary of the composition of soil samples acquired from executed transect. 62

Acknowledgments

First I would like to express my utmost gratitude to my advisor, Dr. William J. Kaiser, for all his support and encouragement throughout the years. He has been a great source of inspiration both academically and personally and his dedication to his students is unparalleled.

I would also like to thank the members of my thesis committee, Dr. Mark H. Hansen and Dr. Gregory J. Pottie, for their guidance and support with my thesis and as instructors.

I would like to extend a heartfelt thanks to Dr. Maxim Batalin of CENS for his assistance and expertise as well as members of the ASCENT group, in particular, Henrik Borgstrom, Brett Jordan, Amarjeet Singh, and Michael Stealey. I have been all too fortunate to be surrounded by such a talented and enthusiastic group and both their intellect and companionship are truly appreciated.

I would also like to express my appreciation towards the UCLA Electrical Engineering Department, CENS and the NIMS program for funding my work. My gratitude is due to the many administrators from these organizations, including Fe Asuncion, Deeona Columbia, and Emy Murakawa, who work very hard to enable the research we do.

Abstract of the Thesis

Calibration and Environment Characterization for Autonomous Aquatic Actuated Sensing

by

Victor Liu Chen

Master of Science in Electrical Engineering

University of California, Los Angeles, 2008

Professor William J. Kaiser, Chair

High fidelity data acquisition of dynamic spatiotemporal phenomena for aquatic environmental research suggests the use of actuated sensors. Furthermore, characterization of the floor in aquatic environments, besides having application for robot localization, is beneficial for environmental science. The NIMS RD and NIMS AQ cable robotic platforms are designed to meet these requirements and satisfy the constraints of large scale, in-field deployments. The development of methods for autonomous range detection as well as spatial and semantic mapping in underwater environments enable greater reliability for in-field operations and high fidelity sensing.

These methods are demonstrated to be important for future developments including localization, navigation, and path planning, particularly for 3D mobility. Experiments have been performed in both controlled environments and lake environments.

CHAPTER 1

Introduction

1.1 Overview

In environmental sensing applications, there is an urgent need for large scale water resource monitoring. These aquatic domains can be characterized as highly variable spatial and temporal phenomena. In addition, these phenomena may be transient and thus the available time window with which to collect scientific evidence may be short. This requires rapid deployment both to permit rapid coverage as well as rapid response to events. The problem is that both high precision measurements of 3D volumes must be performed while environmental constraints are uncertain [SBS07].

We propose a solution to this problem by introducing the Networked Infomechanical Systems (NIMS) family of actuated (robotic) sensing systems [JBK07, PBG05, BSB06]. Two more recent iterations of the NIMS platform design are the rapidly deployable NIMS (NIMS-RD) and aquatic NIMS (NIMS-AQ) systems. This thesis focuses on addressing the problems associated with the calibration, localization, and environmental characterization by these mobile robots in aquatic environments.

1.2 Contribution of Thesis

NIMS-RD and NIMS-AQ have proven to be successful field robotic systems, having been used in many aquatic field deployments for environmental sensing. Nevertheless, there are a number of aspects of the system which may be improved. This is due to several factors including limitations of the cabled robot design, the requirements of rapid response environmental sensing, and the time and energy costs of large scale deployments. This thesis addresses the problem of horizontal and vertical calibration of the actuation system and how these calibration subsystems can extend to system localization. In addition, we discuss an application of the sonar sounding capability for semantic characterization of underwater environments. Contributions of this thesis include:

1. Design and implementation of remote power distribution and data routing platform for the sensor node
   (a) Weather resistant packaging and connectors
   (b) Modular and scalable data and power interface for remote sensors
   (c) Relay control interface for physical sampling device
2. Calibration of horizontal and vertical actuation
   (a) Environmentally resistant horizontal calibration system using magnets and Hall effect sensors
   (b) Accurate and expedient depth profiling using sonar device and robust signal processing
   (c) Integration with NIMS control software
3. Environmental characterization of aquatic environments:

(a) Spatial characterization of aquatic subsurface structure using constrained mobility and 360deg sonar scanning capability

(b) Semantic characterization of aquatic subsurface composition

CHAPTER 2

Overview of NIMS Systems

2.1 NIMS-RD

The Networked Infomechnical Systems (NIMS) was designed to address many of the issues of actuated environmental sensing. It allows remote, high fidelity sensing using a minimal physical footprint. The rapidly deployable NIMS [JBK07], henceforth to be referred to as NIMS-RD, is a recent evolution in the NIMS architecture capable of operating in both terrestrial and aquatic environments. The equipment can be packed up and transported using any mid-size SUV or van and once on site, it can be set up in a matter of hours and operated by an experienced 2- or 3-person team. After environmental sampling has been performed, it can be quickly broken down and moved to a new location. On the other hand, the installation may be left in place to be used again days, weeks, or even months later with only minimal maintenance, satisfying a requirements of semi-permanent environmental sensing installations.

The NIMS-RD system, shown in Figure 2.1 consists of three main cables, the actuation module, mounting hardware, and shuttle node. The three cables used are a main support cable, horizontal actuation cable, and vertical actuation cable. The main support cable is 7 x 19 class strand core, nylon coated 3/16 inches wire-rope rated at 1760 pounds break strength. The support cable may utilize infrastructure present in the environment (e.g. trees, boulders, poles) or it

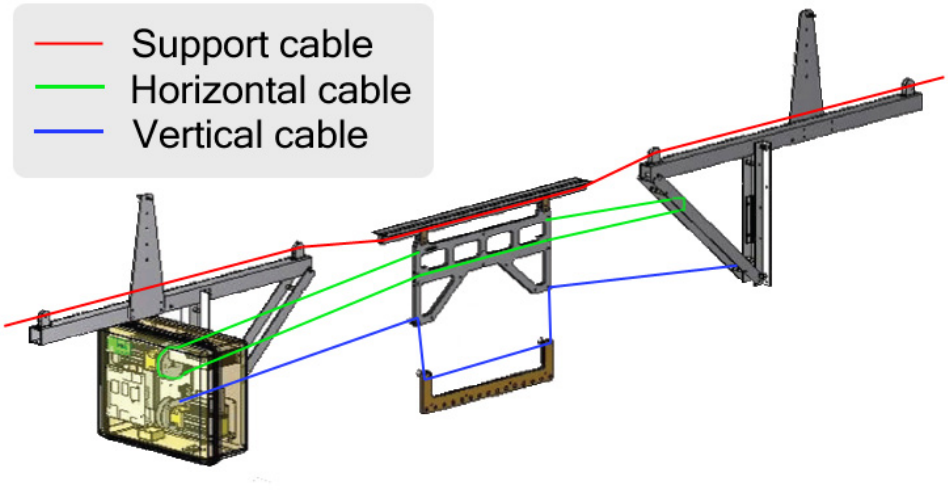


Figure 2.1: Schematic drawing of the NIMS-RD platform

may be secured to portable infrastructure (e.g. anchored ladders, vehicles). The mounting hardware is attached on both ends of the transect with the actuation module secured on one end and the counter mass system attached to the other end.

2.1.1 Remote Sensors Interface

It is often necessary in actuated sensing to enable real-time access to sensor data in order to validate accurate sampling of the phenomena and to determine if and when a system error has occurred. In addition, adaptive sampling algorithms rely on access to real-time data for analysis. This requires a method of routing the data from the payload back to the control system where it can be analyzed. The variety of aquatic sensors, each of which communicates using RS-232 serial interfaces have varying power requirements. Thus modularity and scalability were two important considerations in the design of a platform that could accomodate various combinations of these sensors as demanded by the application.

Another consideration in the design of the data routing method was variability in the sensor payload. Introducing additional sensors would have required festooning more cables from the control system to the payload. This was undesirable as the additional cables would add considerable weight to the load bearing cable, increasing the sag. In addition, preparation of the cables (i.e. bundling into a single cable) for each sensor combination depending on the application requirements would have been a significant burden. Moreover, using many cables would increase the resistance exerted on the accumulation of festooning shuttles against the main node moving back to the origin. The combination of this resistance from the festooning, the upward slope as a result by the cable sag, and the lack of friction on the drive cable can result in highly undesirable cable slippage.

To resolve these issues, the use of a serial-to-ethernet multiplexer in line with a copperlink ethernet extender allows the control system to communicate with all of the remote serial devices at a theoretical limit of up to 5 miles using only a single festooned ethernet cable. The serial-to-ethernet multiplexer allows up to 4 serial devices to be multiplexed into a single channel. If more serial devices are required for an application, a second or third serial-to-multiplexer can be easily added using a hub. This allows any combination of our aquatic sensors to be operated using the same module and requiring only minimal reconfiguration.

The task of routing power for each device was burdened by similar drawbacks. Significant voltage drops and the need to potentially festoon many additional cables made routing DC power inefficient. Routing 120 VAC from the shore and converting to the various DC voltages on the shuttle enabled us to reliably power the remote devices using only a single festooned power cable. A schematic diagram illustrating the data and power connections between the various devices on the platform is shown in Figure 2.2.

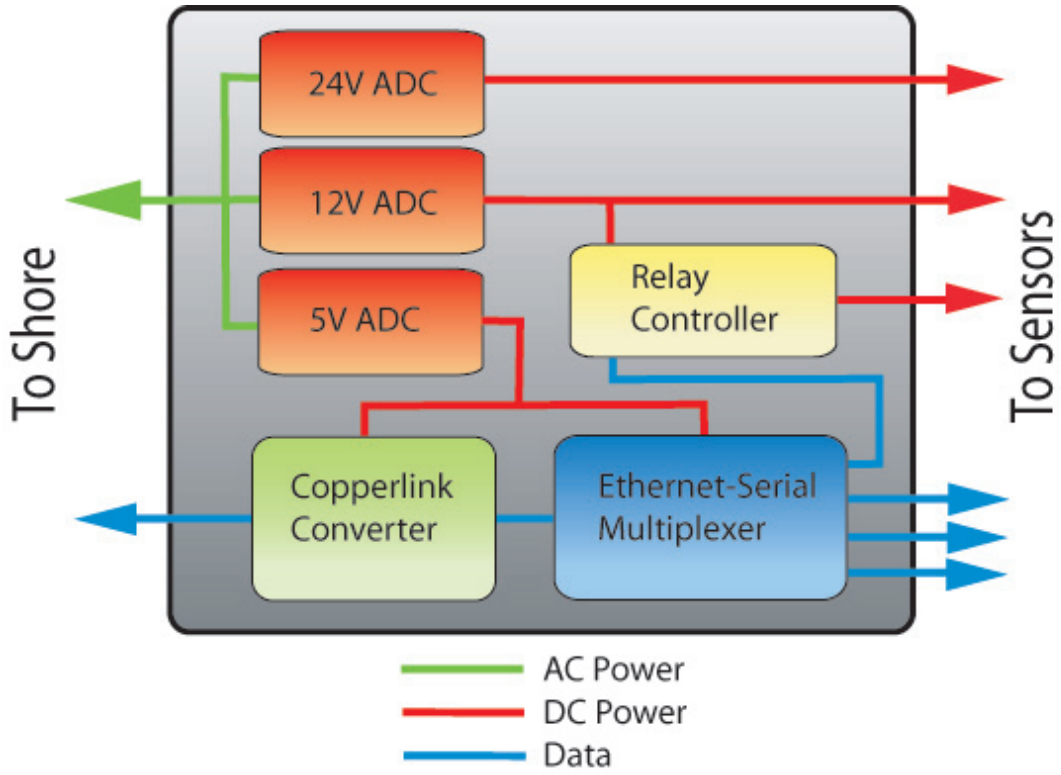


Figure 2.2: Schematic of the SensorBox showing the power and data connections among the various components as well as the inputs from the shore and outputs to the sensors

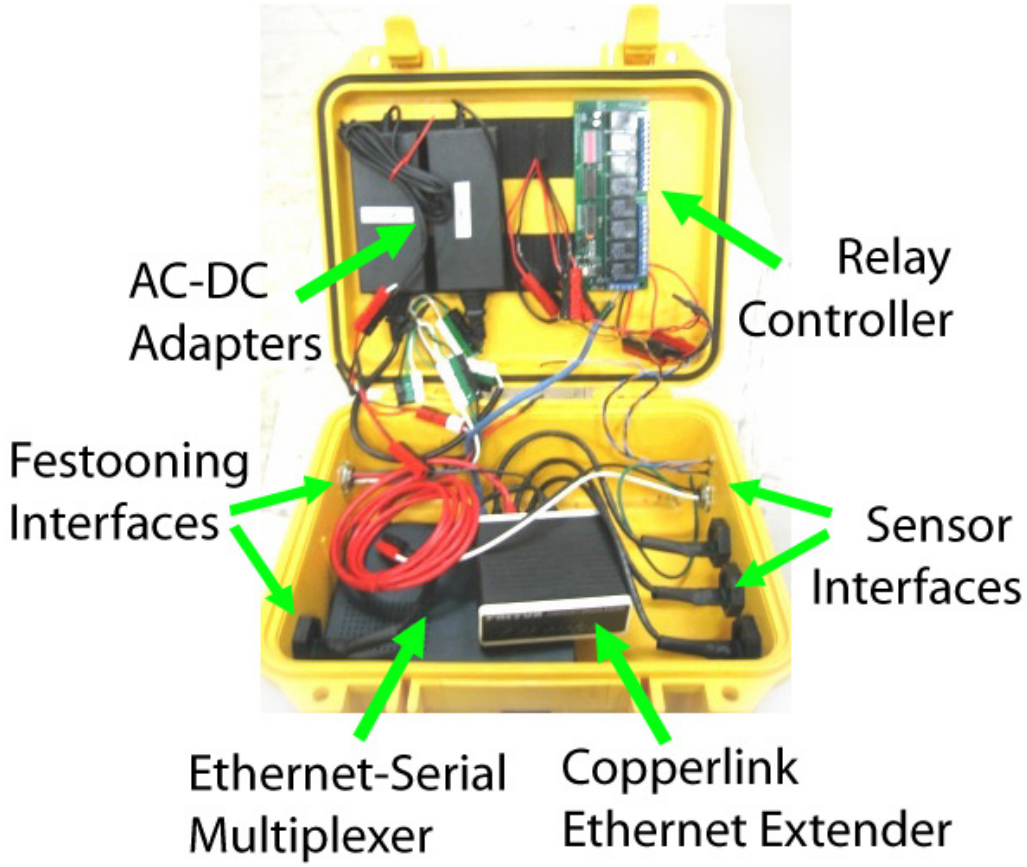


Figure 2.3: Schematic of the SensorBox showing the power and data connections among the various components as well as the inputs from the shore and outputs to the sensors

The SensorBox platform, shown in Figure 2.3 has been successfully used with NIMS-RD deployments at the San Joaquin and Merced Rivers, the UC Merced campus lake, and Lake Fulmor in the San Jacinto Mountains [SBC07, HAG07, SBS07]. Future work for the remote platform will aim to use wireless communications including 802.11 Wi-fi and 802.15.1 Bluetooth. The remote platform will use a battery pack that can be recharged or swapped in the field. This will eliminate the need to festoon data and power cables reducing the load on the main support cable.

2.2 NIMS-AQ

The Aquatic Networked InfoMechanical Systems platform (NIMS-AQ) [SBK08] is the latest in the family of NIMS systems, developed specifically for aquatic applications. Figure 2.2 displays a schematic representation of NIMS-AQ. The system is comprised of a rigid sensing tower supported by two Hobie FloatCat pontoons (developed by Hobie Cat Company) in a catamaran configuration. An actuation module resides on top of the sensing tower that drives the horizontal cable and vertical payload cable for the horizontal and vertical motion, respectively, across a cross-section of the aquatic environment. The vertical cable enables actuation of the sensing payload along the vertical column of water from the water surface to the floor of the aquatic environment. Power for the system is provided by two deep cycle marine batteries housed on top of the pontoons. The horizontal drive cable is kept center-aligned to the craft by using guide pulleys that can be repositioned based on the aquatic environment in which NIMS-AQ is sampling (flowing or still water conditions).

NIMS-AQ was conceived after the tremendous success of NIMS-RD [JBK07] to satisfy the increasing demands of aquatic applications. These include sampling

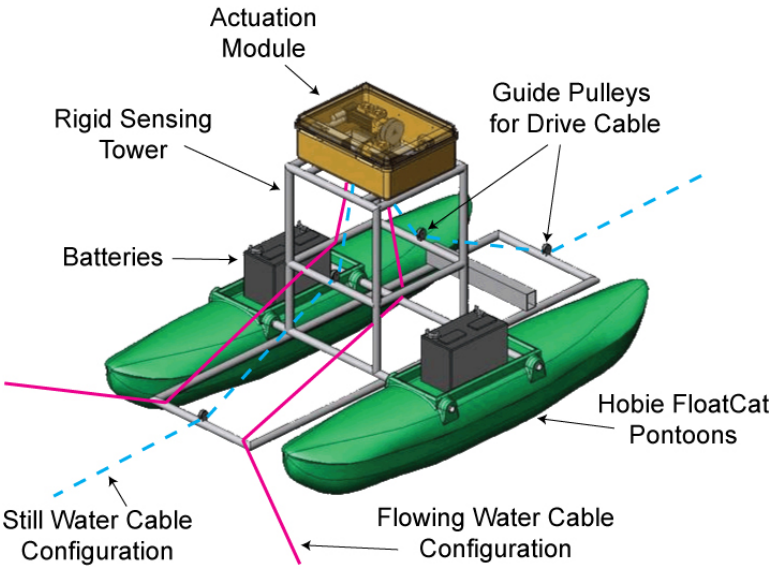


Figure 2.4: Schematic drawing of the NIMS-AQ environmental sensing platform

larger cross-sections of the aquatic environments and providing increased spatial coverage through multiple cross-sections spread over a large spatial domain. This resulted in the development of NIMS-AQ, a tethered system suited for aquatic applications. NIMS-AQ improves upon the previous NIMS designs by providing reduced setup time, lower tension requirements on the supporting cableway, reduced dependency on the specific support infrastructure, and reduced physical footprint. These characteristics make it an ideal platform for several important aquatic sensing applications.

The current version of NIMS-AQ was designed as a prototype system to test the fundamental system requirements including actuation, sensor interfaces, and network communications. The next version of NIMS-AQ will inherit several of these system components and features with an additional capability to provide three-dimensional mobility. It will use multiple cables, similar to the three dimensional NIMS platform [BSB06], to provide the ability to sample the surface of aquatic environments, along with vertical actuation of the sensor payload. Such

capabilities demand a further reduction in setup times and accurate localization. Using sonar for autonomous depth profiling and to provide accurate localization will result in autonomous operation of NIMS-AQ to provide complete three dimensional characterization of aquatic environments with high fidelity.

2.2.1 Remote Access and Control

For the NIMS-AQ system, data and power routing was greatly simplified as the platform design allowed both the control module and power sources (two deep-cycle marine batteries) to reside on the mobile node itself. This removed the requirement on festooning as the sensors could be connected directly to the serial interfaces in the control module. Similarly, using an inverter and power strip with the batteries allows the sensors to be powered directly with individual adapters.

With NIMS-RD, the operator would control the operation of the system by connecting to the control module located on the shore via a local ad-hoc network. Since the control module on NIMS-AQ resides on the mobile node, connectivity may be lost if the mobile node travels outside of the wireless range for simple wireless devices. This was solved by incorporating a wireless router with high-gain antennas to the wireless device on the control module.

2.3 Water Quality Sensors

Sensors are selected based on specific environmental application requirements. Our selection of aquatic sensors includes water quality multisensors [hac], a nitrate sensor [isu], a flow sensor [stk], a fluorometer [cyc], ultrasonic sonar [son07], and a water sampling device as shown in Figure 2.3. The environmental variables measured by each sensor is summarized in Table 2.1.

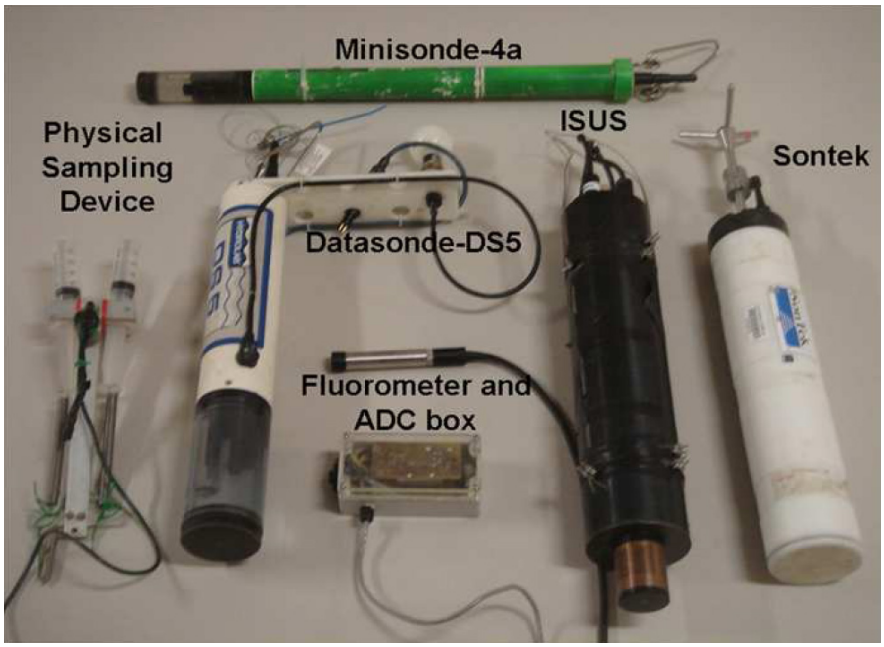


Figure 2.5: Aquatic sensors used on NIMS-RD and NIMS-AQ

The physical sampling device was developed for collecting water samples at arbitrary locations within the transect. It uses dual spring-loaded syringes capable of collecting a combined eighty milliliters of fluid. A linear solenoid actuator combined with relay switching enables a trigger mechanism where the spring-loaded syringe mechanism operates, drawing a water sample. After retrieval of the collected water samples at the shore, the sampling system is prepared for a next sample by deflecting the springs in preparation for future solenoid trigger events.

Remote control of the physical sampling device was enabled via communication with the relay control board in the SensorBox. The relay control board is a commercially available 8-relay RS-232 serial board with 10A single-pole double-throw relays. The software, written in Python, is an implementation of the communication protocol programmed into the relay controller's firmware.

Sensors	
Sensor	**Variable(s) Measured**
Hydrolab DS5	Temp, pH, ORP, SpCond, Res, Sal, TDS, Depth, PAR, Turbidity, LDO, Chlorophyll, Rhodamine
Satlantic MBARI-ISUS	Nitrate Concentration, Salinity
SonTek Argonaut-ADV	3-axis water velocity, Heading/Pitch/Roll, Temp, Pressure
Fluorometer	Chlorophyll
Physical Sampler	Extracts 40-80 mL of water

Table 2.1: Sensors commonly used for NIMS-RD aquatic deployments

CHAPTER 3

Horizontal Calibration and Localization For Cabled Systems

3.1 The Effects of Sag on NIMS RD

In order for NIMS to operate as a convenient tool for researchers and scientists, a correlation must be determined between the position measured by the motor's internal encoder, measured in motor ticks, and physical distances, for example in meters. Without cable sag, this correlation would be a simple ratio of encoder ticks to meters. However, cable sag in a real system, no matter how slight, is unavoidable as a result of finite cable tension, payload mass, and gravity. This is illustrated in Figure 3.1.

We can show that for aquatic deployments, the error resulting from this sag is negligible. The maximum sag displacement occurs when the node is in the center

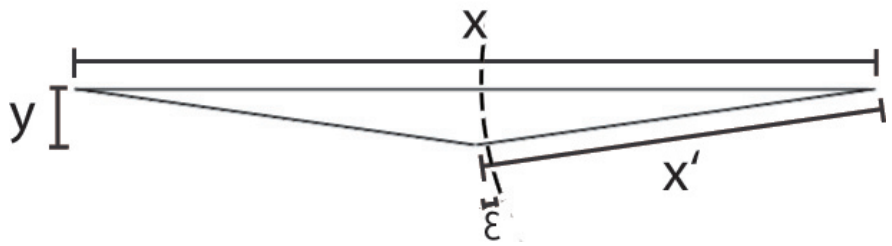


Figure 3.1: Horizontal position error due to sag displacement (not drawn to scale)

of the transect as shown in Figure 3.1. The actual length of the horizontal cable can be modeled by the hypotenuse as

$$x' = \sqrt{\frac{x}{2}^2 + y^2}$$

The absolute error resulting from the sag displacement is the difference between the hypotenuse and the long leg of the triangle:

$$\epsilon = x' - x = \sqrt{\frac{x}{2}^2 + y^2} - \frac{x}{2}$$

The percent error due to sag displacement is the error divided by the total length of the transect:

$$\%error = \frac{\epsilon}{x} \times 100\%$$

Using the above equations, the percent error is modeled for transect lengths ranging from 2 meters to 100 meters, and maximum sag displacements ranging from 0 meters to 10 meters, as shown in Figure 3.2. We can set a liberal limit on the maximum sag displacement as no more than 1/10th of any given length of transect. This estimate is consistent with our observations from past field deployments. The red line in Figure 3.2 shows this relationship. Modifying the previous equations, we can calculate an upper bound on the percent error using the relationship between maximum sag displacement and transect length.

The maximum displacement as a function of the transect length becomes

$$y = \frac{x}{10}$$

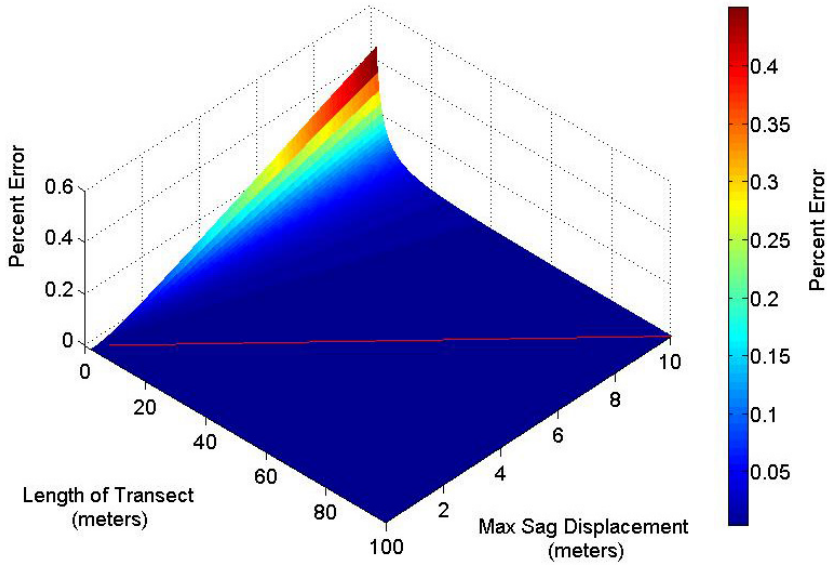


Figure 3.2: Percent error in cable position due to cable sag

The resulting cable length becomes

$$x' = \sqrt{\frac{x}{2}^2 + \frac{x}{10}^2} = x\sqrt{26}/10$$

with absolute error

$$\epsilon = x' - x = x\frac{\sqrt{26}}{10} - \frac{x}{2} = x\frac{\sqrt{26}-5}{10}$$

and percent error

$$\%error = \frac{\epsilon}{x} \times 100\% = \frac{x\frac{\sqrt{26}-5}{10}}{x} \times 100\% = 0.9902\%$$

Using a liberal bound for the maximum sag displacement as of 1/10th of the transect length produces a maximum percent position error of less than 1%. As an example, a trasect length of 70 meters would not be expected to have more

than a 7 meter vertical displacement due to sag and no more th an 70 cm error in position, depending on the amount of tension on the cable and the weight of the payload. From observations in past field deployments, a 70 meter transect is likely to have far less than a 7 meter vertical displacement which would result in far less than 70 cm error in position.

This error bound falls within the requirements for aquatic monitoring applications which, due to flow effects, do not produce phenomena distributions with such high spatial variance. Thus, horizontal calibration of the motors may be performed without requiring the auxiliary calibration cable to remain floating above the shuttle. Previous methods involved using an auxiliary calibration demarcated with visual markers at regular intervals. This required manual calibration of the motors which, although it was sufficiently accurate, required a significant amount of time and effort. Since the error in position due to sag is negligible, the auxiliary cable may be lowered to the level of the shuttle. This allows for a calibration design which uses sensors placed on the shuttle to detect the markers. Such a design would yield greater accuracy and can be automated to reduce the amount of system setup time.

Additionally, a scenario is conceivable where in the event of a failure, such as a power loss, the system may lose position information. This is known as the "kidnapped robot" problem. In order to enable the appropriate fault recovery procedures, the system must be able to quickly determine its location within the transect. This is possible only if there is a means for the system to detect some absolute positional landmarks.

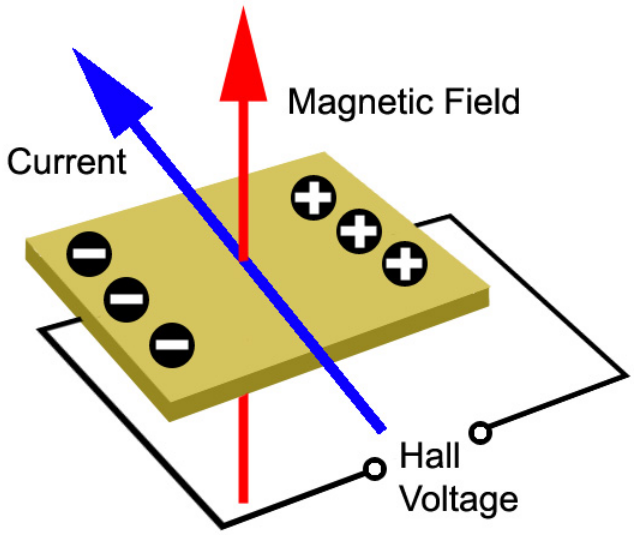


Figure 3.3: The Hall effect

3.2 The Hall Effect And Its Application to Horizontal Localization

The Hall effect describes the potential difference in voltage created when a magnetic field is present on an electric current traveling across a conducting plate. The current consists of electrons that travel in an approximately straight path. In the presence of a magnetic field, this path becomes curved due to the Lorentz force resulting in a potential difference that can be measured perpendicular to the path of the electric current. This is illustrated in Figure 3.2.ling across a conducting plate. The current consists of electrons that travel in an approximately straight path. In the presence of a magnetic field, this path becomes curved due to the Lorentz force resulting in a potential difference that can be measured perpendicular to the path of the electric current. This is illustrated in Figure 3.2.

Hall effect sensors and magnets are often used in position and motion sensing systems for extreme environments where reliability is critical. Part of the benefit

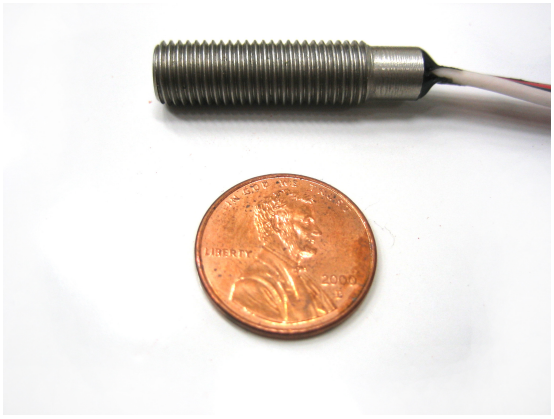


Figure 3.4: Hall effect sensor from Sensor Solutions

is that there are no moving parts involved, neither with the magnet nor with the sensor, so life expectancy is greatly increased. The Hall effect sensor used here is a switch type sensor shown in Figure 3.2. It is comprised of an NPN transistor housed within a tiny stainless steel cylinder. It outputs a high voltage when the sensor is off and a low voltage when the sensor is on (i.e. in the presence of a magnetic field).

3.3 Hardware Implementation

In robotic systems, it is extremely important to have accurate localization in order to enable autonomous operations. If the robot is unaware or uncertain of its location in the environment, it can be difficult to determine what actions to make next. Uninformed actions may result in system failures if they unintentionally exceed certain boundary conditions. In addition, environmental sensor data which require correlation to locations in space will become invalid. This can result in a significant cost whether the application is monitoring chlorophyll distributions in a lake environment or collecting sediment samples on a Martian surface.

In the NIMS cable-based system architecture, mobility is constrained along the axis of the main support cable. In the NIMS-RD system, the node is driven by a closed-loop horizontal cable which is attached to the node at both ends. On one end of the support cable, the horizontal cable is drawn around a motor spool which is covered by a high-friction surface. A counter mass on the opposite end of the support cable provides tension which serves to maintain friction on the cable against the motor spool. Despite this, several factors can cause the horizontal cable to slip on the motor spool, causing the system to lose its position. In some cases, this behavior can result in a catastrophic failure, e.g. unresolved continuous cable slip allowing the motor spool to burn through and snap the horizontal cable.

For this reason, a critical feature of the system is the ability to measure node position using an external sensor. One method of doing this takes advantage of the constrained mobility of the system. Markers can be placed statically along a cable, either the main support cable or an auxiliary calibration cable. As the horizontal shuttle travels along the cable, there is a very high probability that the markers will be detected. The variety of application environments for NIMS presents many different and often harsh environmental conditions. For this reason, magnetic markers and hall effect sensors were chosen because they are inexpensive, rugged, and robust to varying environmental conditions. As the horizontal shuttle travels along the cable, there is a very high probability that the markers will be detected. The variety of application environments for NIMS presents many different and often harsh environmental conditions. For this reason, magnetic markers and hall effect sensors were chosen because they are inexpensive, rugged, and robust to varying environmental conditions.

The magnet calibration and localization system uses commercially available

components. The markers are nickel-plated neodymium 1/4" long and 1/8" in diameter. The sensors are Hall switch sensors and are commercially available from Sensor Solutions [hal07]. The sensor data is relayed to the control module using a Bluetooth enabled ADC, commercially available from Roving Networks [blu07].

The cable uses magnetic markers arranged according to an N-bit Gray code, also known as a reflected binary code. A Gray code is a binary sequence where each successive value differs from the previous value by a single bit. The following sequence is an example of a 3-bit Gray code: 000, 001, 011, 010, 110, 111, 101, 100. This coding scheme was chosen because of its simplicity with regard to implementation and its ability to facilitate error correction. The encoded values correspond to physical positions stored in a hash table.

Depending on the direction of travel the shuttle is moving, as it passes over the code it would not be able to differentiate between 0001 and 1000, nor would it be able to detect 0000. A start bit is required to signal the start of the code. This means that n+1 bits are required to produce a Gray code encoding of 2n values, or that n bits are required to encode $2n-1$ values. If we desire a marker at every x meters, over a transect of length L meters, we would need $log_2(L/x)+1$ bits. $(L/x = 2n-1)$ For example, a 70 meter long transect with a marker located at 5 meter intervals would require 5 bit Gray code to fully encode the cable with unique markers.

A separate modular housing was fabricated which can be attached to the NIMS-RD shuttle or the NIMS-AQ platform as needed. The housing consists of a long rectangular tube containing two fixed guide pulleys on either end as well as holes on each side of the housing for the Hall effect sensors. Because the cylindrical magnets can lie on any side of the cable as it twists, a single Hall sensor

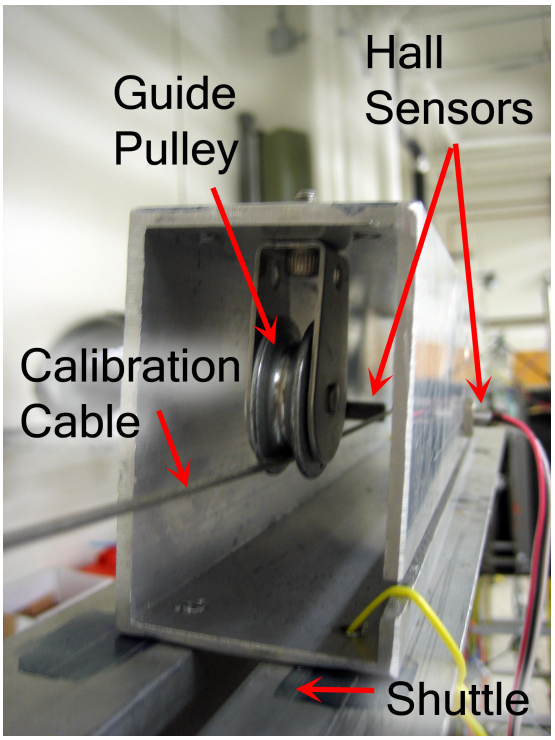


Figure 3.5: Modular housing attached to the shuttle with calibration cable, guide pulley, and hall sensors shown

may not be able to detect the magnet if it is on the opposite side of the cable so two Hall sensors are directed at opposing sides of the cable. The guide pulleys guarantee that the auxiliary calibration cable passes through the gap between the Hall sensors. A slot running along the length of the housing allows the auxiliary cable to be fed in from the side rather than through one end. This allows the cable to be set up or removed quickly and easily without requiring the calibration cable to be detached from its anchor points. The setup is illustrated in Figure 3.3.

The magnets themselves are attached to the cable using super glue and heat shrink. Figure 3.3 shows a magnet arrangement for the code `111101` and the corresponding output signal from the Hall sensor.

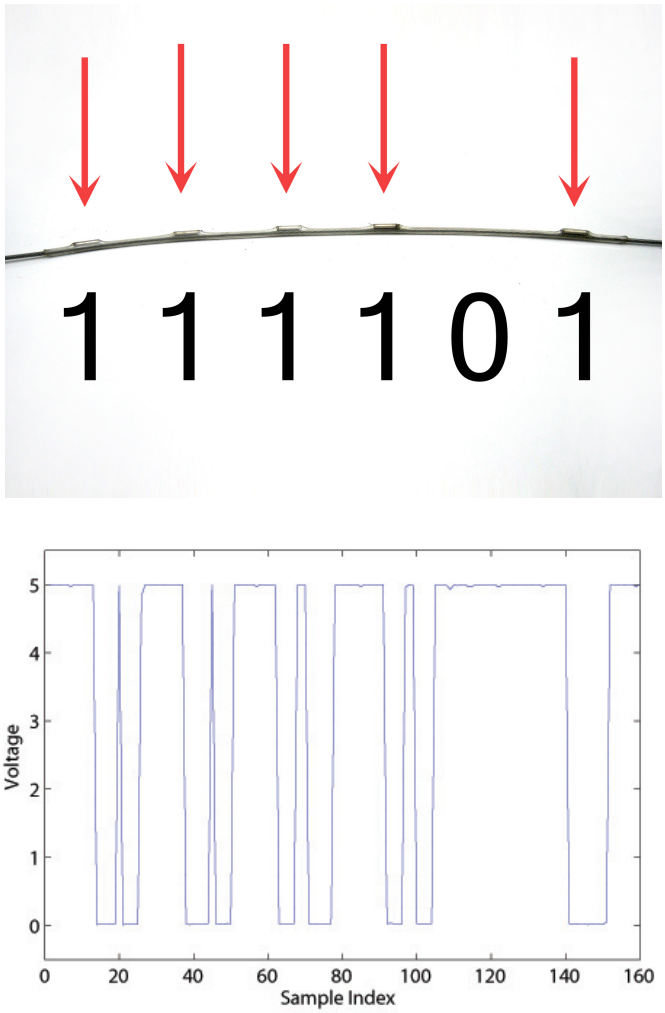


Figure 3.6: Magnet arrangement representing 111101 and the corresponding output signal

The magnets used on the calibration cable are cylindrical magnets and are aligned with the north and south poles lengthwise parallel to the cable. As the magnetic flux density is greater at the poles than at the center of the magnet, the resulting signals often contain gaps from the pulses corresponding to each magnet. This is easily rectified by observing that the length of these gaps is typically 1 or 2 samples in length while the spacing between the magnets is much greater. This allows for straightforward identification of the gaps as corresponding to the low magnetic flux density region of the magnet as opposed to the spacing between successive magnets that indicates a `0` bit.

3.4 Software Implementation

The software is implemented in two layers: communication and processing. The communication layer, implemented in the `adc.py` class, is responsible for intializing a bluetooth socket to the remote bluetooth ADC device. The ADC class allows the user to control sampling via two functions, `start_stream()` and `stop_stream()`. `start_stream()` spawns a thread for a function `sample()` which continuously reads incoming ADC values at some specified sample rate (default of 200 Hz). `sample()` reads the incoming packets and stores the data for each channel in a size-N sliding window multivariable array consisting of ADC values, sequence number, and timestamp. The `sample()` function also checks the state of a globally defined flag variable which, if toggled by the `stop_stream()` function, will cause the thread to exit.

The processing layer, implemented in the `hall.py` class, is responsible for monitoring the output from the ADC and processing the signal. This is implemented within a function, `monitor()`, which monitors the voltage output from the hall sensors. If the voltage falls below 4V, indicating that a magnet has

been detected, the data stream thread is exited. Since two Hall effect sensors are used in this design, the final output signal is taken as a logical OR of the separate signals after signal inversion (since a low voltage represents the presence of a magnet). Two subsequent functions, `scan()` and `decode()`, scan the signal buffer to identify the values of interest and decode the signal by analyzing the number of a consecutive high- and low-value samples.

As the shuttle travels across the transect, the Hall sensors detect the magnets and decode the signals, returning the corresponding position value from the hash table. Simultaneously, the NIMS control software logs motor shuttle position data in terms of motor ticks. When a magnet is first detected, the system time corresponding to that detection is stored. This time parameter is used to correlate the physical position value with the motor tick value corresponding to the initial detection of the magnet.

CHAPTER 4

Sonar Preliminaries

4.1 Overview

Sonar (short for SOund Navigation And Ranging) refers to any system which uses sound waves to determine range, regardless of what medium is used to propagate the sound waves. Sonar works by transmitting an acoustic signal that is reflected by a surface and received by the transducer. The time elapsed between emission of the signal and reception of the echo (time of flight) is measured. If the speed of sound in the medium is known, the distance traveled (range) can be calculated using the following formula

$$d = v \times t \tag{4.1}$$

where d is the distance traveled, v is the speed of sound in the medium, and t is half of the total traveling time.

Figure 4.1 displays the sonar transducer that we use in the current system. It is a miniature, ultrasonic, side-scanning sonar unit that is commercially available from Imagenex Corp. The sensor head and electronics are housed in a cylindrical, watertight body 2 inches in diameter and 3 inches in height. The small size helps minimize the physical footprint of the sensor payload and that of the NIMS AQ system as a whole, in order to perform sampling with minimal disturbance to the environmental phenomena. It operates in one of two frequencies - 675 Hz

Figure 4.1: Miniature, ultrasonic scanning sonar developed by Imagenex Corp

and 850 Hz - and has multiple tunable parameters as well as a 360deg internally rotating transducer head. High frequency echo sounders such as this have less penetration depth than low frequency echo sounders but have higher resolutions.

Figure 4.2(a) shows an example of an unprocessed sonar output signal. The domain is a measure of distance, calculated as a ratio of the relative index of the signal (out of 500 data points) and the maximum range setting of the sonar. The physical range can be calculated from the time-series value using the following equation:

$$r = \frac{n}{500} \times r_{max}$$

where r is the range of the reflecting surface, r_{max} is the maximum range of the sonar transducer, and n is the relative index of the peak in the echo. For example, a strong peak occurring around the 200th data point with a maximum range setting of 10 meters indicates a reflecting surface located at 4 meters.

4.2 Sonar Signal Processing

The sonar outputs time series data of the signal strength of the reflected echo. An ideal output signal from the sonar would consist of a delta function corresponding to the two-way travel time between the sonar and the reflection surface. Estimating the range from such a signal would require no more than a simple algorithm using only Equation 4.2. However, output signals from the sonar can vary considerably with different environments and even within the same environment if the subsurface structure and composition is heterogeneous. This requires robust processing of the signals in order to reliably extract accurate range information in the presence of noise, weak reflections, multiple reflections and other non-ideal conditions. To cope with variation in signal output, a procedure involving a series of computational filters is employed to reduce the probability of an incorrect range estimation. These filters will be referred to as the nearest neighbor (NN), multiple reflections (MR), and expected range (ER) filters. The procedure is illustrated by an example output signal in Figure 4.2. The effectiveness of this approach will be discussed with regard to spatial mapping.

In the first step of the procedure, a cubic spline smoother [Rei67] is applied to the time-series output signal. A cubic spline is a piecewise polynomial function of third degree polynomials defined as follows

$$s(x) = \begin{cases} s_0(x) & \text{for } x \in [x_0, x_1] \\ s_1(x) & \text{for } x \in [x_1, x_2] \\ \vdots & \\ s_n(x) & \text{for } x \in [x_{n-1}, x_n] \end{cases} \tag{4.2}$$

where $s_i(x)$ is a third degree polynomial defined as

$$s_i(x) = a_i(x - x_i)^3 + b_i(x - x_i)^2 + c_i(x - x_i) + d_i$$

for $i = 1, 2, \ldots, n-1$ and n is the number of observations in the time series. Cubic spline smoothing removes the requirement that the spline passes exactly through every data point but retains the requirements that the spline, first derivative, and second derivative be continuous:

$$\begin{aligned} s_{i-1}(x_i) &= s_i(x_i) \\ s'_{i-1}(x_i) &= s'_i(x_i) \\ s''_{i-1}(x_i) &= s''_i(x_i) \end{aligned}$$

For convenience, a Python library was used containing an implementation of the cubic spline smoother [pyt07b]. While we neglect the details for calculation of the spline coefficients, it important to note how the cubic spline can be used to smooth a data set. There are two opposing criteria for this: (1) how close the spline comes to resembling the actual data and (2) the amount of curvature in the spline. The similarity between the spline and the data can be quantified using a chi square measure as

$$\chi^2 = \sum_{i=0}^{n} \frac{[S(x_i) - y_i]^2}{\sigma_i^2} \tag{4.3}$$

The amount of curvature can be quantified using the integral of the square of the second derivative

$$\int |S''(x)|^2 dx \tag{4.4}$$

Combining equations 4.3 and 4.4, we can require that the cubic spline minimize

$$W = \rho\chi^2 + \int |S''(x)|^2 dx \tag{4.5}$$

The constant ρ is called the smoothing coefficient and it determines the trade off between the amount of curvature and the closeness of the fit to the data. Setting $\rho = 0$ removes the constraint on χ^2 and allows the spline to become a straight line while setting ρ very large puts more weight on minimizing χ^2 regardless of the amount of curvature. For our purposes, the cubic spline smoother uses a smoothing coefficient selected pre-runtime that effectively filters high frequency noise while preserving the structure of the signal. This allows for straightforward identification of local maxima. Figure 4.2(b) shows the spline function superimposed on the data with the maxima values identified. This set of maxima values contains a corresponding range value which will have a high probability of corresponding to the first reflection of the transmitted pulse off of the reflecting surface of interest.

In aquatic environments, suspended sediments can form gradients above the actual floor. These sediment gradients can cause scattering of the acoustic pulse and interference that will result in the output signal containing highly col

After a set of maxima has been identified, there may be several maxima values which are highly colocated but only one of which corresponds to the actual reflection off the floor. To resolve this, the NN filter is applied. By assuming a minimum distance between the floor and water surface, a sliding window can be set where the strongest maximum will be selected over neighboring maxima falling within this window. This is shown in Figure 4.2(c). In our current procedure, the window size is static, however a dynamic window size that varies based on

```
Algorithm:nearestNeighborsFilter
Input: nn_window, sonar signal: S(t), set of local maxima: m_1,...,m_N
Output: Reduced set of local maxima: m_1,...,m_N'
begin
    foreach maxima m_i, i ∈ [1, N − 1] do
        foreach maxima m_j, j ∈ [i + 1, N] do
            if |m_j − m_i| < nn_window then
                if S(m_j) > S(m_i) then
                    remove m_i;
                else
                    remove m_j;
    return m_1,...,m_N';
end
```

Algorithm 1: The NearestNeighborFilter: Algorithm for filtering colocated maxima.

the expected depth and environmental characteristics may yield more accurate results. This will be investigated in future work.

Often, the first maximum in the time series will correspond to the reflecting surface and will have the highest intensity, however this is not always the case. Depending on the characteristics of the fluid environment, reverberation can occur resulting in unusually high reflections occurring in the space immediately surrounding the sonar. In some environments characterized by surfaces with high reflection coefficients, the pulse emitted from the sonar will resonate between the floor and the water surface producing multiple reflections increasing in intensity. In order to identify the periodicities corresponding to resonant frequencies of the signal-space we apply the MR filter. This is done by computing the periodogram of the output signal using the Fast Fourier Transform (FFT) [Sch98].

$$S(f) = \frac{\Delta}{n}\left(\left(\sum_{t=-n}^{n-1} x_t cos(2\pi f t\Delta)\right)^2 + \left(\sum_{t=-n}^{n-1} x_t sin(2\pi f t\Delta)\right)^2\right) \tag{4.6}$$

where n is the number of observations in the time series, Δ is $\frac{n+1}{2}$ for n odd or

Algorithm:multipathReflectionsFilter
Input: mr_window, set of local maxima: $m_1,\ldots,m_{N'}$, Fourier transform: $F(t)$
Output: Reduced set of local maxima: $m_1,\ldots,m_{N''}$
begin
  foreach *maxima* m_i, $i \in [1, N'-1]$ **do**
    foreach *maxima* m_j, $j \in [i+1, N']$ **do**
      foreach *frequency* f_k **do**
        if $||m_j - m_i| - f_k| < mr\_window$ **then**
          remove m_j;
  return $m_1,\ldots,m_{N''}$;
end

Algorithm 2: The MultipleReflectionsFilter: Algorithm for filtering maxima corresponding to multiple reflections.

$\frac{n+2}{2}$ for n even, and the frequency f is the range of Fourier frequencies ($\frac{1}{n}$, $\frac{2}{n}$, ..., $\frac{1}{2}$).

The periodogram is an estimate of the spectral density of a stochastic process. It allows straightforward identification of dominant frequency components hidden within the sonar echo, the inverse of which indicates dominant periods of reflection. This works because any function can be decomposed into a weighted sum of sinusoidal component functions, or Fourier series. The periodogram exploits this property in order to find the hidden periodicities in a signal. By correlating all frequencies 1/n, 2/n, ... , 1/2 with the data in order to measure the intensity, it can quantify the importance of a particular frequency in the series. Knowing these periodicities, we can identify and eliminate maxima corresponding to multiple reflections, regardless of their relative intensities as shown in Figure 4.2(d).

Assuming depth profile information and accurate localization are available, we can apply the ER filter. The ER filter operates by setting a window corresponding to the expected range such that maxima falling outside this window are eliminated as shown in Figure 4.2(e). Although this filter relies on localization information which may not always be accurate, it can produce the greatest percent reduction in the size of the point cloud, as discussed in the results.

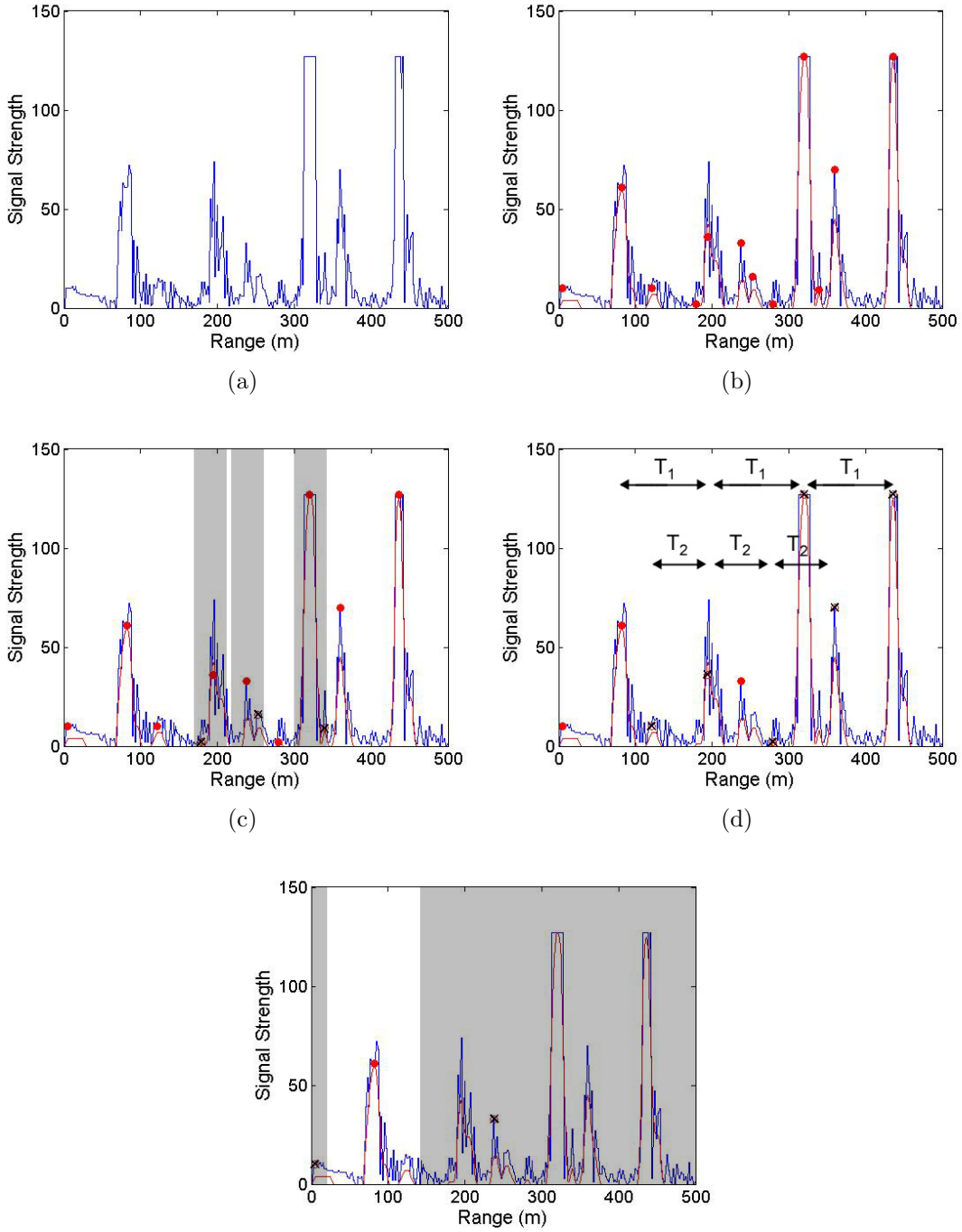


Figure 4.2: Procedure for estimation of range from example reflection signal after application of each filter. The raw reflection signal is shown in Figure(a). Figure(b) shows the reflections signal superimposed with the cubic spline smoother and the local maxima identified. Figures (c)-(e) show the signal after application of the NN, MR, and ER filters respectively.

Algorithm:expectedRangeFilter

Input: er_window, exp_range, set of local maxima: $m_1,\ldots,m_{N''}$
Output: Reduced set of local maxima: $m_1,\ldots,m_{N'''}$
begin
    foreach *maxima* m_i, $i \in [1, N'']$ **do**
        if $|m_i - exp\_range| > er\_window$ **then**
            remove m_j;
    return $m_1,\ldots,m_{N''}$;
end

Algorithm 3: The ExpectedRangeFilter: Algorithm for filtering maxima falling outside the window of an expected range.

Additionally, we can use cluster analysis to further increase the probability of an accurate range estimate over multiple sonar readings. This approach is useful since a single sonar reading may not yield a single range estimate after filter processing, nor would a range estimate from a single sonar reading necessarily be accurate. Taking multiple readings produces a set of range estimates, of which the largest cluster very likely contains a modal value which is accurate. Clustering provides a better estimate over simply taking the mode of the set of range estimates because there are inconsistencies in the sonar output signals due to the nature of the device; i.e. environmental effects such as water flow cause the system and the sonar to sway.

The histograms shown in Figure 4.3 illustrate the benefit of using cluster analysis. With a single sonar reading, it is unclear whether the reflecting surface is located at 10 meters or 21 meters. With five sonar readings, it can be seen that the reflecting surface is located near 10 meters and the reflection at 21 meters is most likely due to multipath reflection, despite the modal value for this data set being 21 meters. With ten sonar readings, this becomes obvious despite the modal value still being 21 meters. In this sense, the clustering is analogous to computing a histogram using larger bin sizes. For convenience, a Python library containing an implementation of the hierarchical clustering algorithm was used

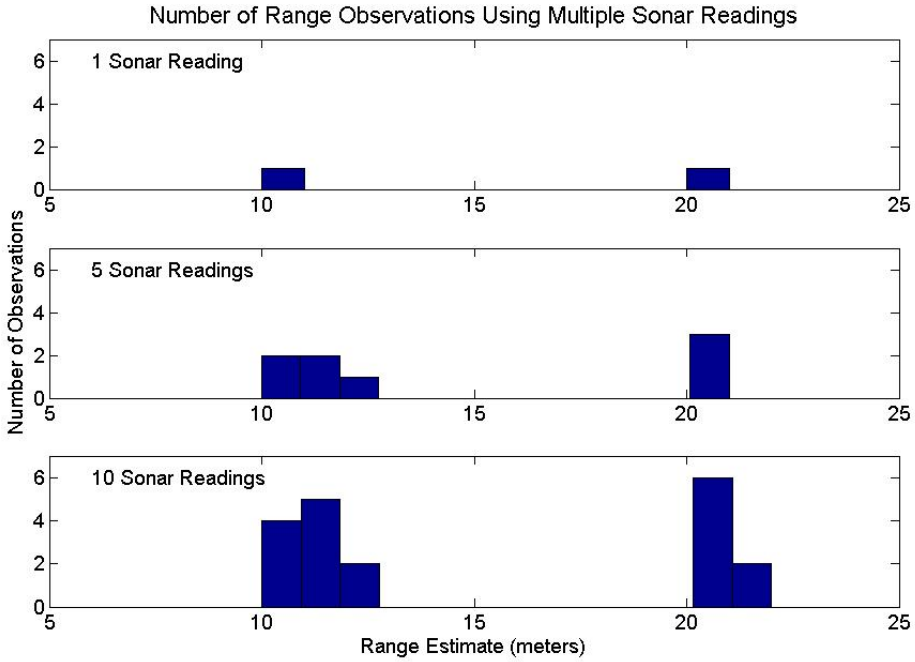


Figure 4.3: Example of how cluster analysis is used to produce a more accurate range estimate using range estimates from multiple sonar readings

[pyt07a].

CHAPTER 5

Vertical Calibration and Localization for Cabled Systems

5.1 Overview

Operation of the NIMS-RD and NIMS-AQ systems for aquatic applications requires an accurate vertical profile of the subsurface in order to generate a set of sampling points that explores the transect as well as to determine motion bounds. For vertical localization, the system can rely on internal motor encoders as neither significant sag nor slippage occurs on the vertical motor. Profiling the floor of the aquatic environment remains a challenge, however. Such measurements can be, and have been in the past, performed manually using a plumbing device. Typically this device would consist of a weight attached to a demarcated cable or a measuring stick attached to a plate (to prevent the stick from digging into the sediment). At other times, the plumbing device was the sensor payload itself. However, manual depth profiling relies only on operator intuition as to when the plumbing device has made contact with the subsurface by using cable tension or pressure as a proxy. This method is prone to inaccuracies and is inconsistent, particularly in flowing water where it can be extremely challenging to maintain a constant position while performing the manual measurement. Moreover, it is a very laborious and time-consuming process; to profile the depth of a 70 meter

river transect in a kayak at 2 meter resolution can require over an hour.

The need for expediency is critical when considering both the time and labor costs required for these types of deployments as well as the urgency in capturing transient environmental phenomena [SBS07]. Unfortunately, the time that is required for calibration, coupled with the time required for setting up the infrastructure can easily add up to half of a day before environmental sampling can begin. These reasons motivate the use of a sonar sounding system which can provide consistently accurate range information from any point within the transect. Integration with the NIMS control software enables the system to perform depth calibration autonomously and expediently.

5.2 Previous Methods for Calibration

In order to collect sensor data that accurately models the phenomena distribution, NIMS must be provided with a set of position bounds (depth profile) that allows it to interpolate a set of sampling points which fully explores the phenomena space. On one hand, overly conservative bounds will not allow the system to sample data which fully models the phenomena distribution. On the other hand, an incorrect depth profile could allow the sensor payload to make contact with the subsurface. As the subsurface structure and composition is unknown, the risk of causing damage to the sensors or allowing the cable to become entangled is real. In addition, such behavior could cause disturbances in the environment such that sensor data would not accurately reflect the phenomena distributions.

Previous methods for vertical calibration of the NIMS system relied on manual tools and multiple operators working in a synchronized fashion. Typically, a plumbing device was used which consisted of a measuring tape with a weight

attached to one end. Other tools used in the past have included a rigid measuring stick with a large circular plate and a depth sensor (included as part of the Hydrolab multisensor package) attached to the sensor payload.

While these methods can yield accurate measurements, they are typically prone to inaccuracies and inconsistencies due to the nature of the environment and the method of sampling. These methods required that one operator control the position of the NIMS platform from the shore while at least one other operator performed the depth measurements on a kayak. The depth measurements were performed by lowering the measuring device until contact with the subsurface was observed. In the case of the measuring tape or the sensor payload, this method used cable slack as a proxy for subsurface contact as well as operator intuition to determine a suitable threshold for cable tension. While the rigid measuring device reduced the uncertainty associated with the cable slack, the large drag coefficient made it difficult to maintain position from an unstable platform (the kayak), particularly in flowing water. In all of these cases, uncertainty in the sampling methods required the use of conservative bounds for motion.

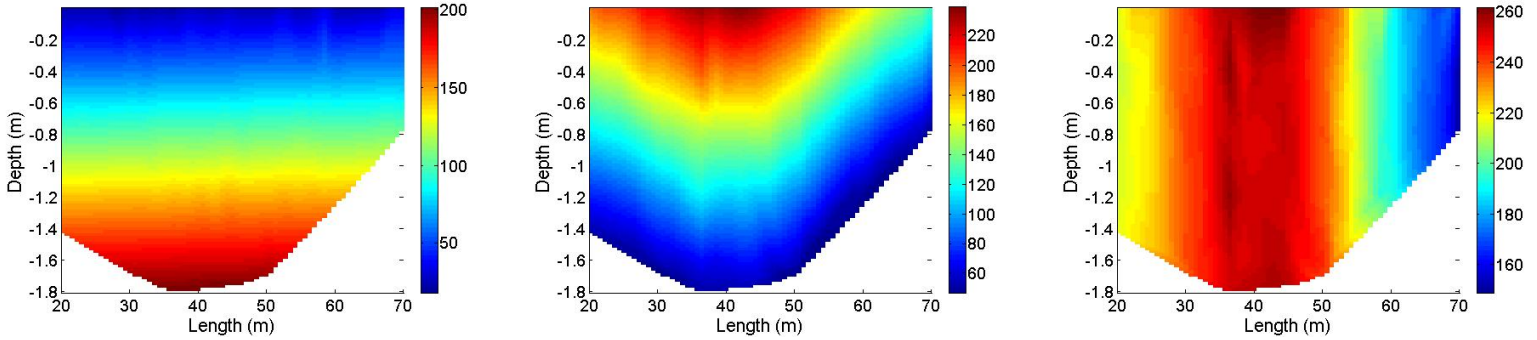
Furthermore, these manual methods are very time consuming and require a good deal of effort on the part of the deployment team. While design validation is important from an engineering perspective, these methods do not add much benefit to the design of the system and in this experimental applications, it is important that the scientific objectives be met as well. The fundamental time constraint becomes more critical if the phenomena of interest is transient [SBS07]. This motivates the design of a system that can perform this calibration procedures accurately and expediently and without requiring much physical strain from the operators or users.

5.3 Range Estimation

Equipped with the sonar signal processing methods described in the previous chapter, we can use the sonar to quickly and accurately determine the depth profile of the aquatic environment. In the case of NIMS-RD, the sonar is easily attached to the sensor payload. On the NIMS-AQ platform, the sonar can either be attached to the sensor payload or be hard mounted to the platform itself.

An experiment was performed at the UC Merced campus lake to determine the accuracy of the sonar for randing estimation in shallow, natural environments. Figure 5.2 shows a satellite image of the transect location as well as the NIMS AQ platform operating on the lake with the sonar and sensors attached. The sonar was mounted on the NIMS-AQ platform at a fixed depth 6 cm below the water surface. Sampling was performed at 2 meter intervals along the 70 meter transect. Simultaneously, a sensor payload consisting of a Hydrolab water quality sonde, which is equipped with a depth sensor, was lowered vertically from the platform. Contact with the lake bed was determined when a significant decrease in tension in the vertical cable was observed. Similarly, a third measurement was produced by lowering a weight attached to a flexible measuring tape until the tension in the measuring tape was observed to reach a low threshold. In the case of the Hydrolab, position was controlled by the NIMS AQ platform whereas in the case of the measuring tape, position was controlled by an operator in a kayak. Although these methods use cable slack as a proxy for indicating contact of the payload with the sediment surface, they provide reasonable estimates of the depth at each position. Particularly for the tape measure method, the operator can make use of tension control via his hands and intuition whereas the computer controlled motors, which lack tension feedback, cannot.

The results demonstrate that the sonar is accurate to within ± 10 cm of the



(a) (b) (c)

Figure 5.1: Colormaps of subset of (a) range estimates from sonar to water surface, (b) range estimates from sonar to lake bottom, (c) and full depth of water column

Figure 5.2: Satellite map of the UC Merced campus lake with experimental transect labeled and NIMS AQ operating on the lake with the sonar and other sensors attached

ground truth measurements provided by the tape measure, as shown in Figure 5.3. In addition, although the Hydrolab depth measurements are inaccurate, they are consistent with the measurements taken from the sonar and the tape measure.

In the next experiment, the sonar was attached to the sensor payload and sampled with the sonar head trained upward at the water surface and downward at the UC Merced lake floor, for each raster point. The purpose of this experiment was to verify that range estimates from the sonar are consistent for different depths. Figures 5.1(a) and 5.1(b) show the range estimates from the sonar to the lake bed and the range estimates from the sonar to the water surface, respectively. Figure 5.1(c) shows the full depth of the water column calculated at each raster point by adding the range measurements to the water surface and to the lake floor. The distribution of depth values is clearly consistent with the depth profile resulting in a horizontal stratification of the depth values. From this data we can infer that the calculation of depth using the sonar does not vary with depth and the sonar can reliably report range information regardless of where it is located within the transect. This information can further be used to more accurately localize the position of the sensor payload within the transect.

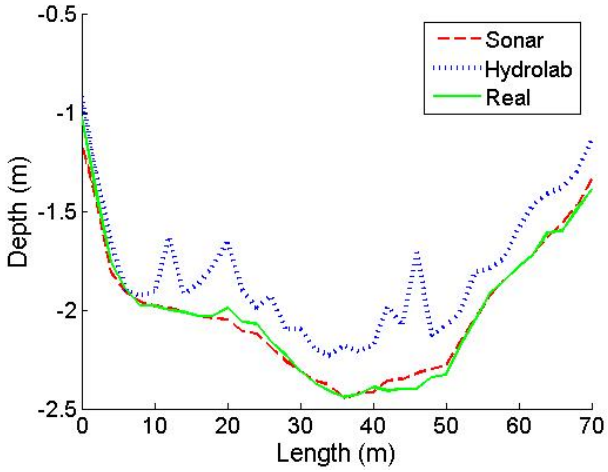


Figure 5.3: Comparison of depth measurements from sonar, Hydrolab, and tape measure

5.4 System Integration

The initial experiment performed at Merced demonstrated the accuracy of the sonar device for performing sounding in natural, shallow-water environments. While the depth measurements proved to be very accurate, the process itself was performed manually, by commanding NIMS-AQ to move to a specific location and then operating the sonar. From those results, another experiment was performed at Lake Fulmor near the James Reserve in the San Jacinto Mountains to test sonar integration with the NIMS-AQ control software in order to perform depth profiling autonomously. This required operating NIMS-AQ in file-based mode, through which it receives motion commands via a file interface. A python script commanded the NIMS-AQ platform to move to a set horizontal position coordinates spanning the length of the transect as well as collect and process sonar readings at each location. The range measurements from the sonar readings were correlated with the NIMS-AQ horizontal positions using a time parameter. The depth profile is shown in Figure 5.4. Whereas our traditional methods of depth

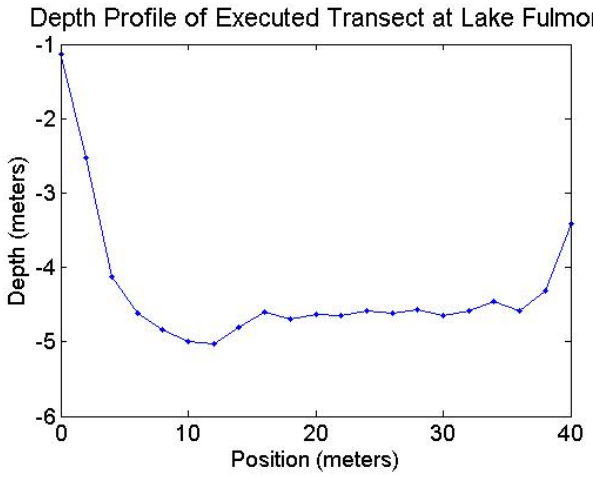


Figure 5.4: Depth profile of transect executed at Lake Fulmor using autonomous depth profiling

profiling would have required approximately an hour to complete and coordination between at least two operators, the system integrated with the sonar was able to perform depth profiling autonomously in about five minutes - a significant time reduction.

CHAPTER 6

Environmental Characterization with Miniature Sonar

6.1 Spatial Characterization

NIMS allows actuation to be exploited for spatial characterization of the subsurface environment. In a sense, spatial characterization is merely an extension of the system's ability to detect range using the sonar. Mobility of the NIMS AQ (or NIMS RD) platform is constrained to a 1D line, however the sonar is able to acquire range information r at specified angles θ, which, along with the NIMS AQ position $(x_p,\ z_p)$, creates a cylindrical coordinate system. We can transform these cylindrical coordinates (x_p, r, θ) to Cartesian coordinates (x, y, z) taking into account that in our system the cylindrical coordinate system is effectively rotated 90° on its side, using the following equations

$$\begin{aligned} x &= x_p \\ y &= r\cos(\theta) \\ z &= r\sin(\theta) + z_p \end{aligned}$$

6.1.1 Aquatic Laboratory Subsurface Reconstruction

In many natural and large body aquatic environments, it is often difficult to observe the floor visually due to the absorption of light and quality of water. In order to verify the accuracy of the sonar, an experiment was performed in a swimming pool. The swimming pool provided an environment which allowed us to control the placement of several objects within the pool and make ground truth measurements of the positions and orientations in order to compare the results. A large water bottle and several rugged equipment cases varying in size and shape were distributed throughout a section of the pool. A scan of the area was performed by moving the sonar manually and collecting sonar reflection data over a 45deg arc. This data was then processed and fitted with a surface in Matlab using a 'nearest' interpolation with the `griddata` function. The experimental setup and results are illustrated in Figure 6.1. Objects appearing in the surface reconstruction accurately correspond to objects in the environment in size, position and orientation.

6.1.2 Lake Subsurface Reconstruction

The results of the spatial mapping experiment in the swimming pool enabled us to proceed with operation in a natural environment. In order to reconstruct 3D maps of the lake bottom, the sonar was mounted at a fixed depth on the NIMS AQ platform and oriented such that the scanning plane of the sonar was normal to the transect plane and to the water surface. The sonar collected reflection data at 3° intervals over a 90° arc centered at normal incidence to the lake bottom, and repeated this sampling at 2 meter intervals along the 70 meter transect.

Figure 6.2 shows an example surface reconstruction of the lake bed on part of the executed transect. By comparing the depth of the surface as a function of

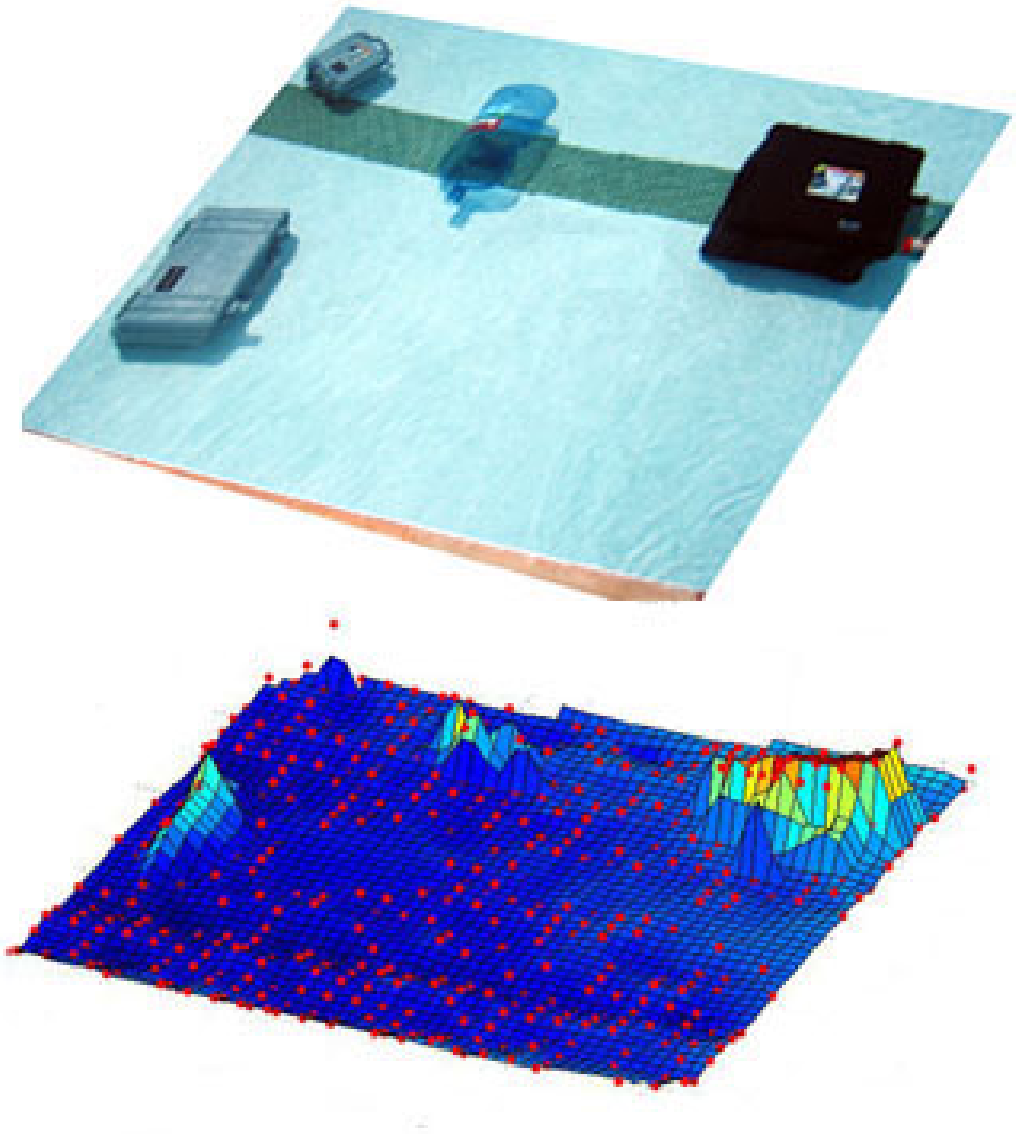


Figure 6.1: Obstacle arrangement and surface reconstruction from pool mapping experiment

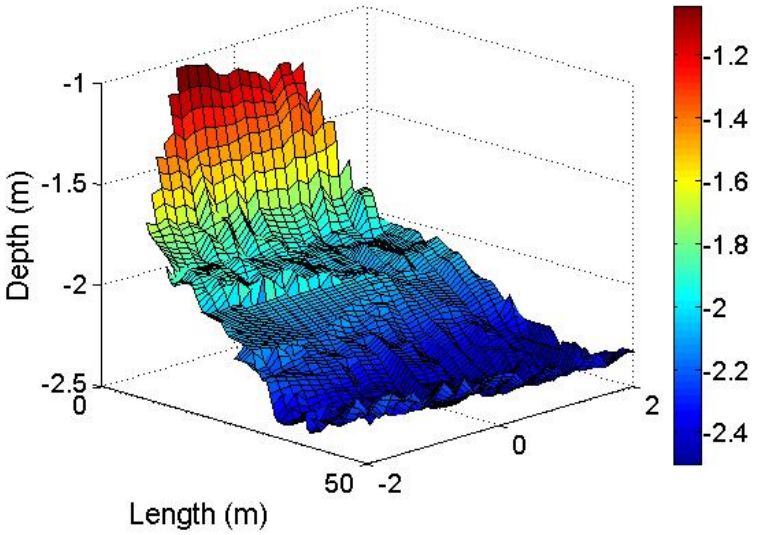
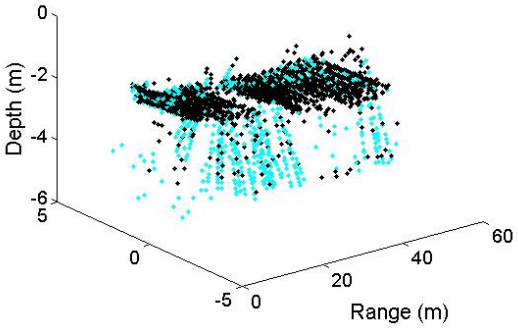


Figure 6.2: Example of the spatial map acquired using the NIMS-AQ platform on a part of the executed transect. The depth reported by the sonar as indicated by the surface color is consistent with the position of the z-axis.

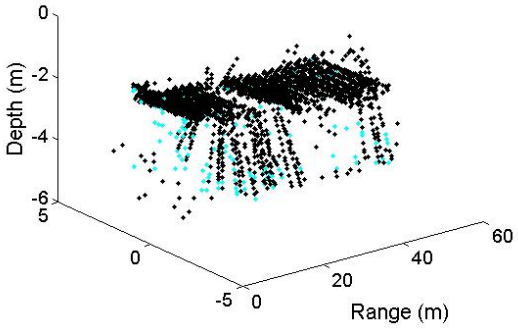
the length with the data from the depth profiles collected by the sonar and the tape measure in Figure 5.3, it is sufficient to conclude that the reconstruction is accurate. In addition, since the transect was chosen near the center of the lake, we would not expect to observe much of a 'bowl' effect but rather that the lateral variation be relatively small (i.e. flat along the y-axis) as shown. However, a slight down-sloping gradient is observable in the positive direction on the y-axis in the surface reconstruction, consistent with the position of the transect.

The performance of each computational filter can be measured by the percent reduction in the number of points in the point cloud resulting from the application of each filter. The size of the point cloud and percent reduction in the number of points resulting from each of these filters is summarized in Table 6.1. The point clouds resulting from each filter are illustrated in Figures 6.3(a) to 6.3(f).

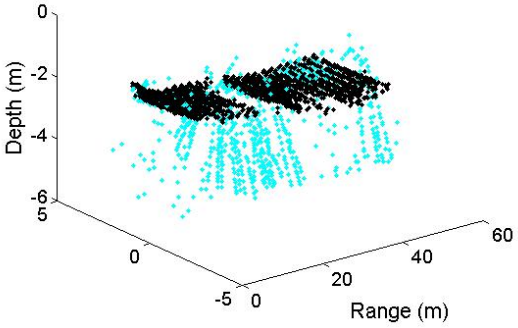
In this example, the contribution of the MR filter to the number of points



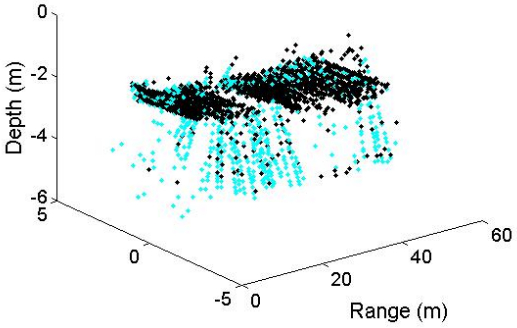
(a) Point cloud after applying NN filter



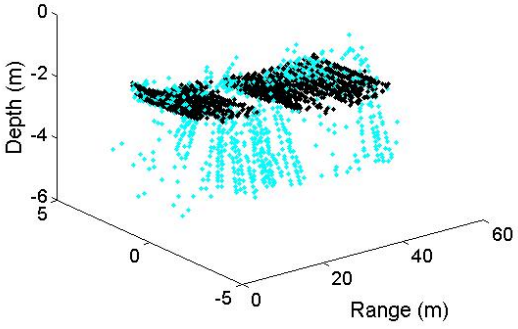
(b) Point cloud after applying MR filter



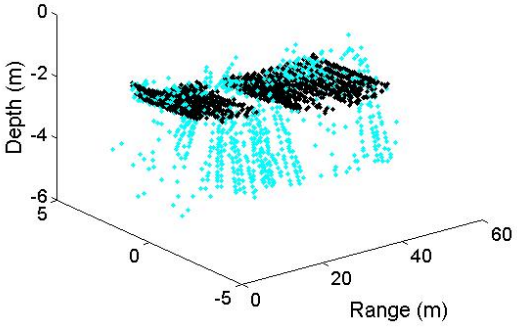
(c) Point cloud after applying ER filter



(d) Point cloud after applying NN and MR filters



(e) Point cloud after applying NN and ER filters



(f) Point cloud after applying NN, MR, and ER filters

Figure 6.3: Resulting points clouds (dark points) after applying NN, MR, and ER filters to the unfiltered point cloud (light points)

Table 6.1: Number of points in the point cloud and percent reduction in size after application of nearest neighbors (NN), multiple reflections (MR), expected range (ER) filters.

Filters Applied	Number of Points	Percent Reduction
$\emptyset$	2741	0.00%
NN	2049	25.25%
MR	2058	24.92%
ER	1925	29.77%
NN, MR	2032	25.87%
NN, ER	1710	37.61%
NN, MR, ER	1705	37.80%

reduced is minuscule. However, in a natural environment such as a lake, this is to be expected as most of the pulse energy will be absorbed by the lake floor to prevent resonance.

6.1.3 Occupancy Grid Mapping

As the sensor scans the environment and the reflection signals are processed, these points are added to or removed from a dynamically allocated hash table using an occupancy grid algorithm [Elf89, ME85]. The (x, y, z) triplet is used as a key for its associated probability value, i.e. the probability that the coordinate is occupied in the spatial map. An infinite number of key values is possible yet it is not feasible nor effective to store each unique key that arises. To satisfy this, the space is discretized into a finite number of uniformly sized cells. For each point falling within the space bounded by the cell, the probability of the cell being occupied increases. If the probability of the cell decreases past a certain threshold, the key corresponding to the cell is removed from the hash table to ensure that hash table contains only information about the occupancy of the map and not about the vacancy. Hence, the information about the 3D map of

Algorithm:updateMap

Input: $Threshold$, $minValue$, i, locations: $l_1, \ldots, l_i$, k cells: $c_1, \ldots, c_k$, Cell Probability: $P(c_1), \ldots, P(c_k)$, Location Intensity: $I(l_1), \ldots, I(l_i)$

Output: Updated probabilities: $P(c_1), \ldots, P(c_k)$

```
begin
    foreach location l_j in sonar line of sight do
        if I(l_j) > Threshold then
            if c_m contains l_j and c_m exists then
                Increment P(c_m);
            else
                add c_m to cell list;
        else
            if c_m contains l_j and c_m exists then
                if P(c_m) > minValue then
                    Decrement P(c_m);
                else
                    remove c_m from cell list;
    return P(c_1), ..., P(c_k);
end
```

Algorithm 4: The UpdateMap: Algorithm for updating the cell probabilities in the map.

the aquatic floor is saved efficiently in memory. The cell update algorithm is described in Algorithm 4.

The UpdateMap algorithm was implemented in Python and tested in a pool environment using several obstacles arranged as shown in Figure 6.4. Scans were performed by manually moving the sonar across the field from right to left at 5 cm intervals, and scanning 180deg at 6deg intervals with the sonar at each location, one sonar reading per angle. As the sonar readings are processed, cells are added to and removed from the hash table in real-time. If the cell for new range estimate already exists, the probability corresponding to that cell is adjusted accordingly.

Figure 6.5 shows the results of each scan. The continuous red line near the bottom of each graph represents the edge of the pool. In Figure 6.5(a), there is a large mass which clearly corresponds to the large pelican case. A smaller mass beneath it corresponds to the two smaller pelican cases. Since only a single sonar reading was taken for each angle (for the purpose of expediency), cluster analysis

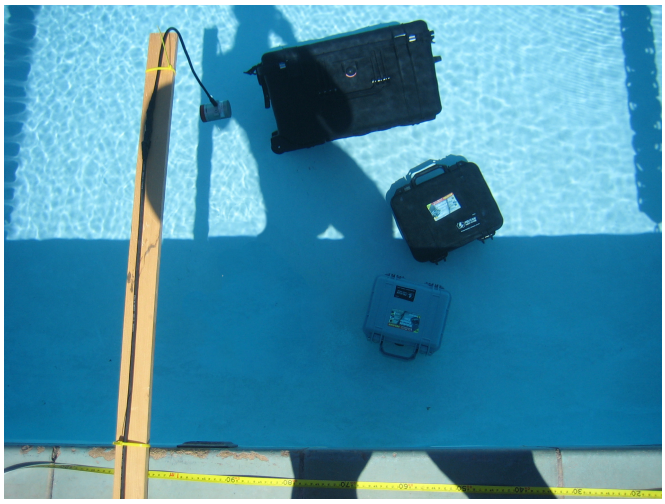


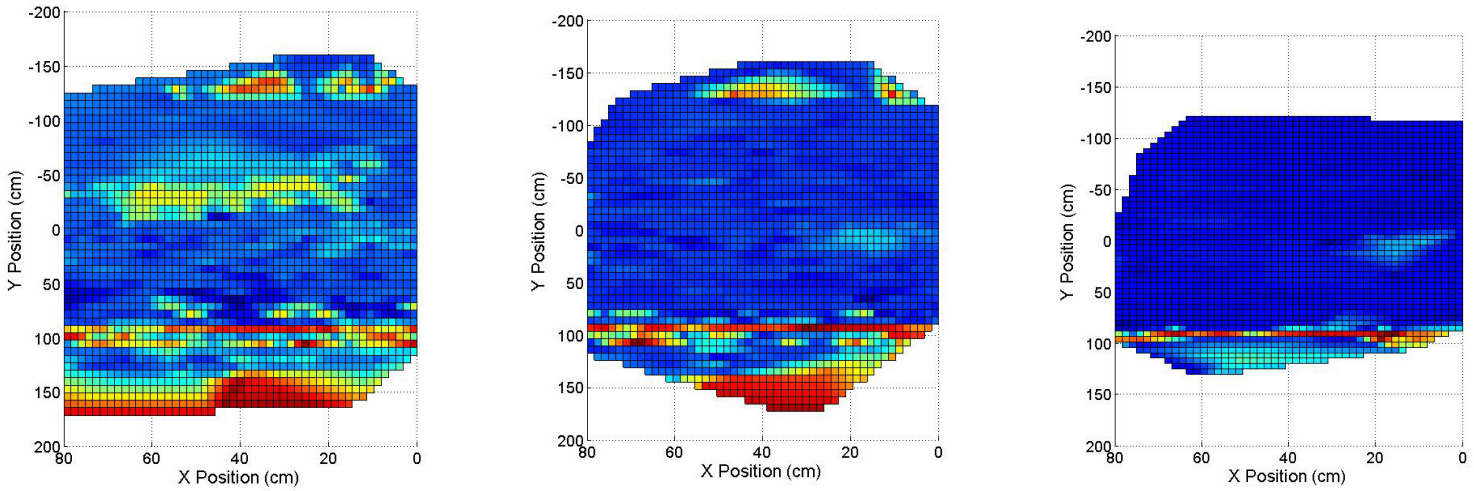
Figure 6.4: Obstacle arrangement for gridmapping experiment

Table 6.2: Size of occupancy grid for each scan

Scan	Number of Occupied Cells
Initial scan	1015
Final scan	855
Final scan, high probabilities	179

could not be performed and there is considerable noise as a result.

In the first scan, the large black pelican case at the top was present but was removed for the second scan. Figure 6.5(b) reflects the absence of the large pelican case. Although much of the noise is still present, by removing the cells from the final scan which have lower probabilities of occupancy, most of the noise is eliminated while the overall structure remains accurate, as shown in Figure 6.5(c). Table 6.2 summarizes the size of the occupancy grid for each scan. These results demonstrate that the algorithm is capable of both detecting changes in the environment and dynamically adjusting the probabilities of occupancy of individual cells in real-time.



(a) Surface reconstruction using points from initial scan

(b) Surface reconstruction using points from final scan

(c) Surface reconstruction using points from final scan with higher probabilities

Figure 6.5: Top view surface reconstructions from grid mapping experiment performed in swimming pool (note: x- and y-axes are not proportional)

6.2 Semantic Characterization

Another application of the sonar sounding capability is the characterization of subsurface sediments in aquatic environments. Traditional approaches toward river ecology tend to treat river systems and ground water systems as separate entities due to the marked differences between the two environments. This is also due, in part, to the disciplinary focuses of scientists studying these systems: most groundwater studies are conducted by hydrologists while most river studies are conducted by ecologists [VFS90].

Recent advances have shown that the interface between groundwater and rivers plays a vital link in the ecology of both systems. This interface is known as the hyporheic zone. The hyporheic zone is the volume of sediment and porous space lying beneath and around rivers and streams. It now understood that the hyporheic zone is responsible for the exchange of water, nutrients, other materials and biota between surface water and ground water. Recent advances in the understanding of the hyporheic zone, using improved mathematical models with an emphasis on the dynamic ecotone model, are helping to shed light on the surface and subsurface flows and solute transport interactions. The ability to more effectively locate hyporheic zones would aid in understanding the spatial and temporal dynamics of hyporheic zone fluctuations [VFS90].

The hyporheic zone is defined as a medium of high porosity and mainly consists of coarse gravels or sand [VFS90] [KJ06]. Much work has been done in the field of marine acoustics toward developing sonar systems that are capable of characterizing subsurface sediments. However, traditional approaches for estimating the physical properties of subsurface sediment have made use of low frequency, wide bandwidth sonar systems, e.g. in the range between 100 Hz and 10 kHz [Sch04, TMN02]. At these frequencies, the pulses emitted from the sonar

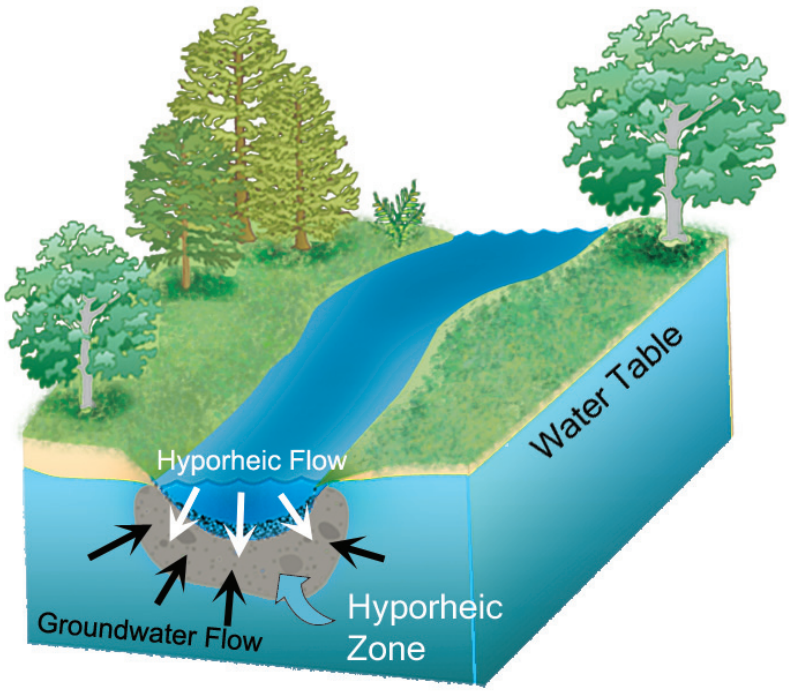


Figure 6.6: Groundwater system involving the hyporheic zone

are able to penetrate the uppermost sediment layer to sediment layers located tens of meters below. This allows for the calculation of the attenuation rolloff of the sediment layer which is calculated as the least squares slope of the attenuation as a function of frequency. The attenuation rolloff is highly correlated to the permeability of the sediment layer, which is obtained from a plot of attenuation rolloff against permeability, generated using an estimated porosity measurement [Sch04].

Another important value in the inversion procedure is the reflection coefficient [Sch04] of the sediment and is calculated using

$$R = 10 \log \left(\frac{\langle I_{seabed/water} \rangle r^2{}_{seabed/water}}{\langle I_{air/water} \rangle r^2{}_{air/water}} \right) \tag{6.1}$$

The reflection coefficient is highly correlated to the porosity of the sediment layer. An estimate of porosity is obtained from a reflection coefficient-porosity plot generated from the expected permeability. This estimate is then used to generate a new attenuation rollof-permeability plot in order to refine the permeability estimate. The porosity and permeability calculations are reiterated until they converge on a solution. They are then used to calculate the mean grain size and other acoustic properties of the sediment [Sch04].

Sonars used for sediment classification are typically large submersibles designed to be towed on ships or attached to the ship's hull. These systems are designed for deep water oceanographic studies such as locating the remains of ancient shipwrecks or locating oil for off-shore drilling [MB01, GS89]. Clearly, it is not feasible to operate such sonar devices in shallow water studies using the NIMS platform. While its small profile makes it ideal for use on lake and river environments and on the NIMS AQ platform, it is capable of operating only at 675 kHz and 850 kHz. At these frequencies, it is not feasible to determine the attenuation rolloff. In addition, at this high frequency, pulses emitted from the sonar will not penetrate the top sediment layer making it unfeasible to determine the attenuation. We can, however, apply the formula for determining the reflection coefficient which will provide an estimate of the composition of the subsurface sediment.

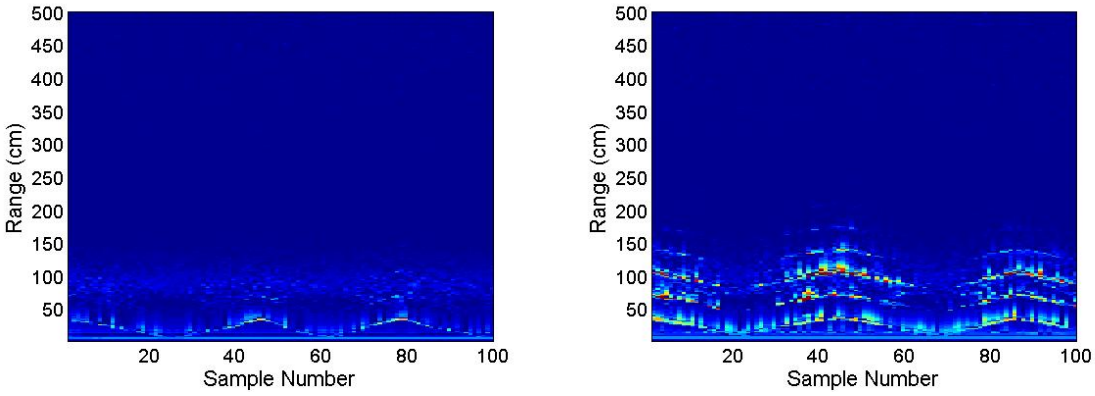
Another approach for semantic characterization using high frequency echosounding involves analysis of the line shape of the reflected echoes [FMB93]. While different sonars produce different reflection characteristics, several features that may be qualified in the echogram are roughness of the line surface, line thickness, and multiple echoes. These features are consistent with characteristics of the ensonified region of the subsurface. For example, a sandy or gravelly surface

would result scattering which would not likely produce clean multiple echoes in the echogram.

Semantic mapping experiments were conducted in both controlled laboratory setups and outdoor lake environments at the UC Merced campus. Experimental results from each experiment will be discussed.

6.2.1 Mesocosm Experiments

Experiments to test the feasibility of semantic mapping using the Imagenex sonar were first conducted in a variety of mesocosms. A 22-liter bucket was filled with water and a thin layer of sand on the bottom while a second bucket was filled with water and a thin layer of clay. The sonar was submerged and sonar readings were logged at different depths to determine what, if any, differences could be discerned from the resulting echograms.



(a) Sand (b) Clay

Figure 6.7: Echograms from buckets filled with sand and clay

Figures 6.7(a) and 6.7(b) show the echograms resulting from the buckets filled with sand and clay. Each echogram consists of 100 sonar readings taken while the sonar was lowered and raised vertically in the water column, resulting in

the sinusoidal line shape for the range. In the echogram from the sand bucket, the initial reflection is strong and clean. Although subsequent reflections can be observed, they are more faint and become progressively noisier. This is consistent with the qualities of sand in that as the grain size is larger than other sediments, the surface is rougher and likely to cause scattering.

On the other hand, the bucket with clay consisted of a thick modeling clay which settled into a smooth surface at the bottom of the bucket. The echogram depicts a strong initial reflection followed by distinct subsequent reflections occurring regularly with a period corresponding to the water depth. This is also consistent with the characteristics of the clay surface as it is a smooth, solid surface making it a good specular reflector which would not produce much scattering.

Another experiment was performed to compare the output signals from several different sediments and sediment combinations using a swimming pool as a simulated environment. Various sediments and combinations of sediment were placed into large sealable bags which were submerged in the swimming pool. This enabled sonar data collection with different sediments in a large water volume however the plastic bags were required in order to prevent contamination of the pool. The sediments consisted of sand, clay (undissolved modeling clay) and mud (modeling clay dissolved in water). The sonar was suspended approximately 80 cm above the bags and sonar readings were recorded. In addition, bags of different sediment compositions were placed atop each other to determine if subsurface penetration using the sonar device was possible.

Figure 6.8 shows the sonar output signals from each sediment bag and combination of bags. The first plot shows the sonar output signal from the pool floor with no bags present. The second plot shows the sonar output signal from

bag filled with water but no sediment. The purpose of observing the empty bag was to estimate the transfer function corresponding to the bag. By comparing the sonar output signals for the bag and for the pool floor, it appears the bag slightly reduces the echo strength but does not significantly distort the signal. The output signal for the bag containing sand consists of a clearly discernible first reflection followed by almost negligible multiple reflections. The bag containing clay produces an output signal with a strong first reflection followed by multiple reflections which are decreasing in strength. This behavior is consistent with results from the previous mesocosm experiment. The layering of sand on top of clay produces an output signal which is very similar to that of sand alone. This suggests there is no significant subsurface penetration in sand. The bag containing the mud mixture produces an output signal which has multiple reflections similar to that of both clay and the pool floor but of an echo strength which lies in between. Similarly, the layering of mud on top of clay produces an output signal similar to clay with discernible but weaker multiple reflections. As the consistency of the mud is fairly fluid, this suggests that sonar penetration occurred but that there is also some absorption of the signal due to the particulates in the mixture.

Yet another experiment was performed using larger mesocosms consisting of a tall vertical water tanks. This allowed sediment characterization to be performed in a volume and water depth more comparable to those expected in our typical deployment sites but without the need to use plastic bags. The water tanks were filled first with sand and then a clay mixture and sonar data was collected. Characteristic output signals for each sediment type are shown in Figure 6.9. Range differences in the initial reflection are a result of a difference in the size of the tanks and consequently the maximum water depth. As seen in previous results, the output signal resulting from the sand reflection is characterized by a

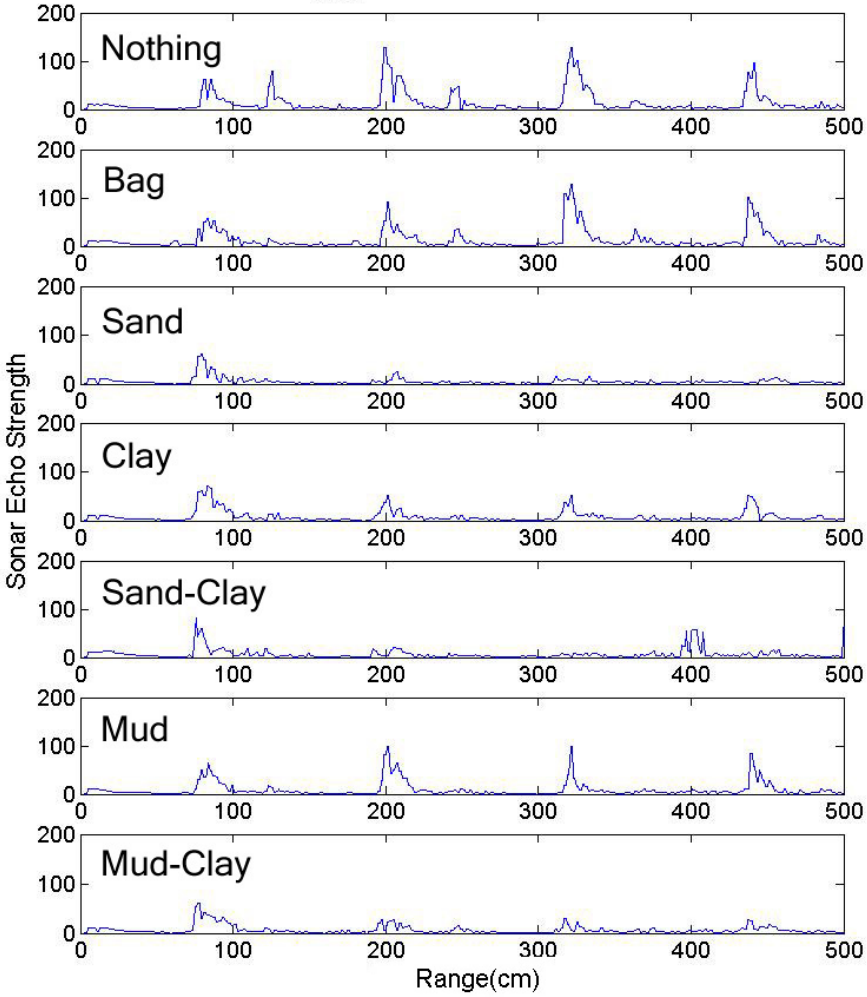


Figure 6.8: Sonar output signals from sediment comparison experiment

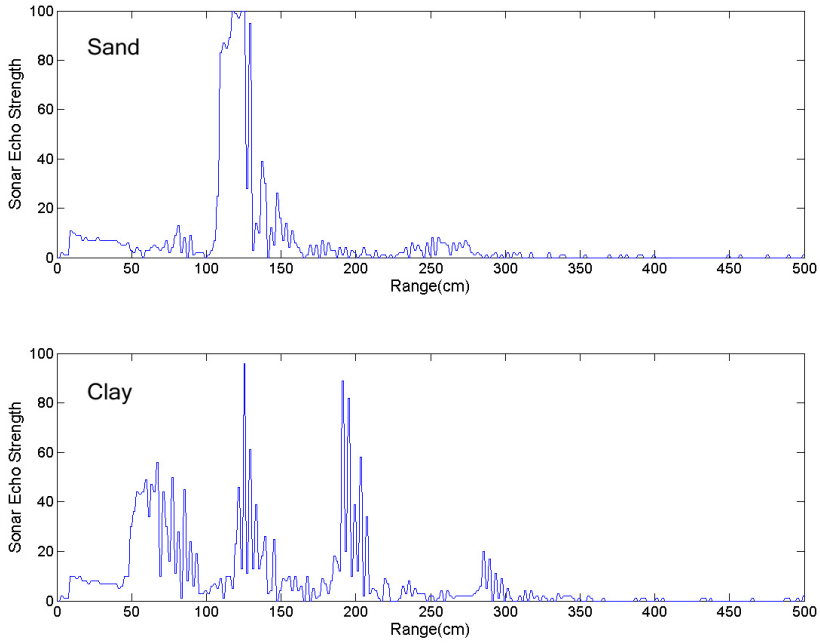


Figure 6.9: Sonar output signals for sand and clay from large mesocosm experiments

single reflection while the clay produces multiple reflections.

6.2.2 Field Experiment

An experiment was conducted in the field to collect sonar reflection data in a natural environment using the NIMS-AQ platform. The experiment took place at the UC Merced campus lake. The transect where NIMS-AQ was deployed is shown in Figure 5.2.

To test this in the lake environment, the sonar was attached to the sensor payload on NIMS-AQ. Sampling was performed with the internal transducer head trained upward at the water surface and downward at the lake bed. The reflection coefficient was calculated as a ratio of the intensities of the reflected and incident sonar pulses using Equation 6.1 and depends on the range of the sonar to the

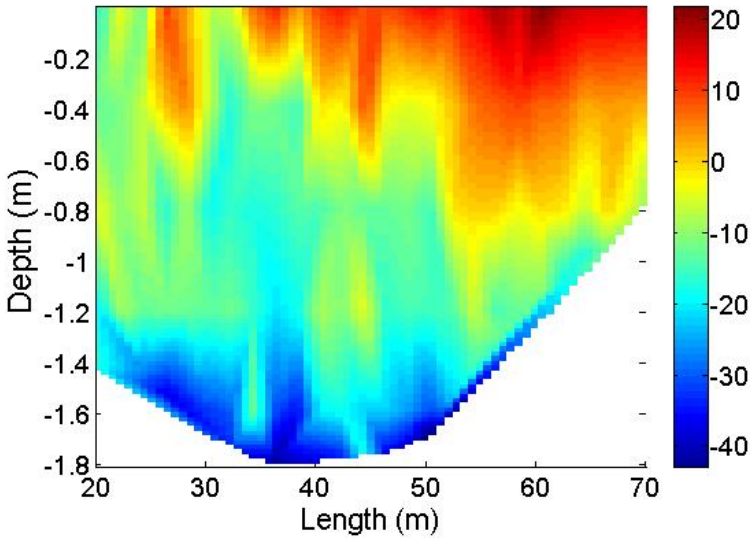


Figure 6.10: Reflection coefficient distribution along the lake transect.

lake bed and to the water surface. Since it is not feasible to measure the incident pulse intensity, the water surface reflection intensity is used as a proxy. Figure 6.10 shows the reflection coefficient of the sediment calculated at each raster point in the transect. While the reflection coefficient varies with depth, this variation is inconsistent and the distribution of the reflection coefficient along a constant depth is also not consistent with the distribution of the depth measurements. This suggests that the reflection coefficient is not solely a function of depth and could represent some physical or acoustic property of the lake sediment.

Further inspection reveals that there is a general trend on the reflection coefficient. At any given depth, with the depth held constant, the reflection coefficient tends toward lower values near the deepest sections of the lake, gradually increasing toward the near end and increasing sharply at the far end at approximately 50 meters. This suggests that the sediment occupying the deeper regions of the transect is absorbing more of the pulse energy whereas the shallower regions are reflecting more of the pulse energy.

Figure 6.11: Soil core sampler

Table 6.3: Qualitative summary of the composition of soil samples acquired from executed transect.

Distance	Description
50 m	mud
55 m	mud
65 m	mud with dense rocks
68 m	mud with dense rocks

Samples of the lake sediment were collected at several points along the transect using a soil core sampler shown in Figure 6.11 [soi07]. Due to the difficulty of operating the core sampler in the deeper parts of the lake, only samples close to the banks could be extracted. Table 6.3 summarizes qualitatively the composition of the sediment samples. From this information we can infer that the concentration of rocks decreases toward the center of the transect and is consistent with the reflection coefficient distribution in Figure 6.10 calculated from the sonar output.

An analysis of the echogram characteristics yields the same conclusion. In the center of the transect where the lake is the deepest, the line characteristics are more faint and do not produce a double echo. Toward the far end of the transect

starting at around 55 meters, the echogram depicts a double echo which remains present through the end of the transect. The presence of the double echo, together with the increased line thickness suggests that the subsurface composition is a coarse grained material or flat rock which is consistent with the collected samples [Dam80, SL66].

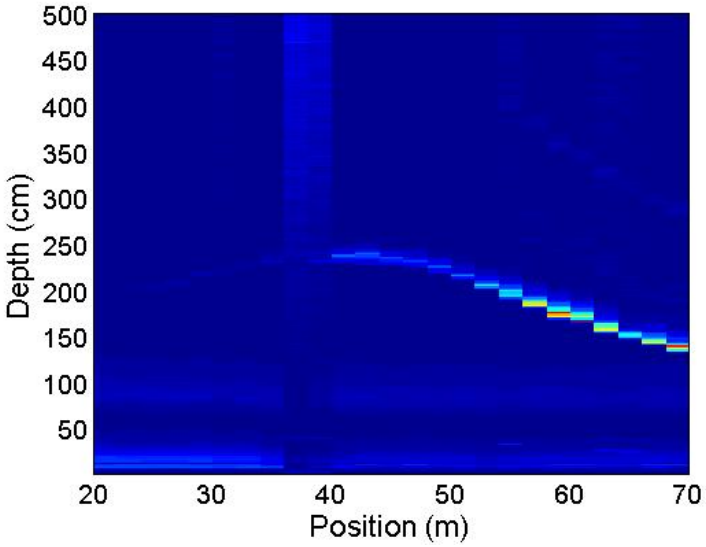


Figure 6.12: Echogram from part of the executed transect at the UC Merced campus lake

(a) 50 meters

(b) 55 meters

(c) 65 meters

(d) 68 meters

Figure 6.13: Sediment samples extracted from the transect at the UC Merced campus lake

CHAPTER 7

Conclusion and Future Work

The subject of this paper is autonomous horizontal and vertical calibration of NIMS-RD and NIMS-AQ platforms and environmental characterization for aquatic environments. For horizontal calibration, an overview of the design was provided. This design exploited the constrained mobility of the actuated sensing platforms to implement a magnetically encoded auxiliary cable along with Hall effect sensors which were robust to extreme environmental conditions. For vertical calibration, the design and implementation of a sonar sounding system and integration with control systems was provided, including sonar signal processing methods. These methods use a series of computational filters and clustering to accurately extract range information from the sonar readings.

The magnetic calibration system can be extended for localization purposes however it is capable of providing only low frequency corrections at discrete points. Were the system to experience continuous and undeterred cable slip, the magnetic calibration would not be able to quickly detect such faults. Future work for horizontal calibration and localization will require integration with an external optical encoder attached to an idler wheel which could perform fine-grained position corrections. The external optical encoder is not sufficient to maintain accurate localization itself since cable slip can still occur on the idler wheel. Therefore the external optical encoder and the magnet calibration system would operate simultaneously with complementary objectives: the external

optical encoder provides high resolution, low fidelity position corrections while the magnets and Hall sensors provide low resolution, high fidelity position corrections. Similarly, the depth profiling system can leverage the spatial mapping algorithms toward more robust localization of the sensor payload in the water.

Semantic characterization experiments were performed in both laboratory environments and in the field. It is clear that relationships exist between the composition of subsurface sediments and the behavior of acoustic signals at the sediment-water interface. Sonar reflection data was analyzed and sediment samples were collected which are consistent with the literature. These initial results clearly demonstrate that different subsurface compositions produce sonar output signals with distinctly different characteristics. More exhaustive experiments in more varied natural environments are required to determine the level of variation in output signal and learning algorithms may be used to quantify these characteristics.

Real-time semantic characterization of the aquatic subsurface would enable features to be extracted which could add greater accuracy to the localization of the sensor payload. Similarly, real-time analysis of environmental sensor data (e.g. chlorophyll) could yield phenomena distributions with high spatial variance and low temporal variance which could provide localization information.

APPENDIX A

Source Code

A.1 relay.py

```
import serial
import os
import time

# Relay class for National Control Devices R810Pro Relay Control Board
# www.controlanything.com

#Author: Victor Chen

class relay:
    # initializes the relay object
    def __init__(self,SER=None):
        if os.name == 'nt':
            print 'WINDOWS Based Machine'
            if SER != None:
                self.ser=SER
            else:
                self.ser = serial.Serial('COM1',9600,timeout=0,xonxoff=1)
        elif os.name == 'posix':
            print 'UNIX Based Machine'
            if SER != None:
                self.ser = SER
            else:
                self.ser = serial.Serial('/dev/bhnS3',9600,timeout=0.5,xonxoff=1)

    def toggle_all(self):
        # toggles all relays
        self.ser.write(chr(254)+chr(31))

    def on(self, number):
        # turns on a single relay 0-7
        self.ser.write(chr(254))
        self.ser.write(chr(number+8))

    def all_on(self, number):
```

```
        # turns on all relays
        self.ser.write(chr(254))
        self.ser.write(chr(30))

    def all_off(self, number):
        # turns off all relays
        self.ser.write(chr(254))
        self.ser.write(chr(29))

    def off(self, number):
        # turns off a single relay 0-7
        self.ser.write(chr(254))
        self.ser.write(chr(number))

    def test(self):
        # tests communication, returns true if successful
        self.ser.write(chr(254))
        self.ser.write(chr(33))
        msg=self.ser.read(self.ser.inWaiting())
        if msg == 'U':
            return True
        else:
            return False
```

A.2 adc.py

```
import time
import bluetooth
import thread

'''
Bluetooth ADC class for Roving Networks BlueSentry AD data acquisition and
control module

Make sure PyBluez (python bluetooth library) is installed!

Author: Victor Chen
'''

class adc:

    def __init__( self, addr=None ):

        #self.addr = '00:A0:96:0A:E1:2A'
        self.addr = '00:A0:96:10:9E:DC'
        if addr != None:
            self.addr = addr
        try:
            self.connect()
```

```
        except:
            self.discover()

        self.data = {}
        self.stream_flag = 0
        self.windowsize = 3000
        self.sample_rate = 200
        self.truncated = ''

    def connect ( self ):
        '''
        Connects to the bluetooth devices with the specified address
        Default settings:
            5V power output active
            ASCII output mode
            100 Hz sample rate
        '''

        self.sock = bluetooth.BluetoothSocket( bluetooth.RFCOMM )
        try:
            self.sock.connect((self.addr, 1))
            self.sock.settimeout(0)
            self.sock.send('!p3ad4650\n')
            time.sleep(1)
            self.sock.send('q')
            time.sleep(1)
            x = self.sock.recv(100)
            print x

        except:
            print 'Failed to connect to address: ' + self.addr
            print 'Try discover()'

    def discover( self ):
        '''
        Discovers all nearby bluetooth devices and allows user to
        select the appropriate bluetooth address to connect to
        '''

        print 'Discovering bluetooth devices...'
        self.devices = []
        self.devices = bluetooth.discover_devices()
        if self.devices == []:
            print 'No devices detected within range'
        else:
            for i in range(len(self.devices)):
                print '[' + str(i) + '] ' + self.devices[i]

            ch = int(raw_input('Select device: '))
```

```
            while ch < 0 | ch > len(self.devices):
                print 'Invalid selection'
                ch = int(raw_input('Select device: '))

            self.addr = self.devices[ch]
            self.sock.close()
            self.connect()

    def sample( self, channels=[0] ):
        '''
        Polls the bluetooth ADC and returns a single sample for
        the channel(s) specified
        '''

        # if one channel specified, converts channel to list
        if type(channels) is int:
            channels = [channels]

        try:
            self.sock.send('*')
            time.sleep(.001)
            recv = self.sock.recv(200).strip('\r\n').split(' ')

            #print recv
            vals = []
            for i in channels:
                vals.append(recv[i+1])

        except:
            print 'Connection lost or device not connected'

    def __stream__( self, channels=[0] ):
        '''
        Samples ADC continuously
        '''


        # reset data structure
        self.data = {}
        self.truncated = ''
        self.data['sequence']=[]
        self.data['time']=[]
        for i in channels:
            self.data[i]=[]

        truncated = self.truncated


        #self.sock.send('!')
```

```
self.sock.send('$')

while(self.stream_flag):

    time.sleep(1.0/self.sample_rate)
    print self.sock.recv(1000)

    '''
    recv = ''
    if truncated != '':
            recv = truncated + self.sock.recv(2000)
            truncated = ''
    else:
            recv = self.sock.recv(2000)
            print recv

    ''
    #print 'recv='
    #print recv

    # searches for '-' and '\r' tokens in each receieved packet as
    # indicators of a single data frame

    if recv != '':
        markera = []
        markerb = []
        for a in range(len(recv)):
                if recv[a] == '-':
                        markera.append(a)
                if recv[a] == '\r':
                        markerb.append(a)

        # stores each complete data frame to a list
        data = []
        for b in range(len(markerb)):
                data.append(recv[markera[b]:markerb[b]])

        # stores any truncated data frames to be appended to the next packet
        if len(markera) != len(markerb):
                truncated = recv[markera[len(markera)-1]:]
                self.truncated = truncated


        #print 'packet='
        #print packet

        # iterate through the list and extract from each packet the
        # sequence number and sensor output for each channel
        for i in range(len(data)):
```

```
                        #print 'data=' + str(packet[i])

                        print data[i]
                        sequence = int(data[i].split(' ')[0].strip('-'),16)
                        self.data['sequence'].insert(0,sequence)
                        self.data['time'].insert(0,time.time())

                        for channel in channels:
                            val = data[i].split(' ')[channel + 1]
                            #print 'val=' + str(val)
                            self.data[channel].insert(0,int(val,16)*5.0/65535)


                # removes old elements from the dataset to maintain window size
                for key in self.data.keys():
                    self.data[key][self.windowsize:]=[]
        '''

    def start_stream( self, channels=[0] ):
        '''
        Creates a stream thread to sample continuously
        '''

        # if streaming, stop stream
        if (self.stream_flag):
            self.stop_stream()

        if type(channels) is int:
            channels=[channels]

        # ensures adc is not outputting and clears receive buffer
        self.sock.send('!')
        #self.clear_recv_buffer()


        self.stream_flag = 1
        self.stream_id = thread.start_new_thread(self.__stream__,(channels,))

    def stop_stream( self ):
        '''
        Stops the stream thread by toggling stream_flag, and sending
        the "stop continuous output" command
        '''

        if (self.stream_flag):
            try:
                self.sock.send('!')
                self.stream_flag = 0
                self.clear_recv_buffer()
```

```
            time.sleep(.1)

        except:
            print 'Connection lost or device not connected'

def set_sample_rate( self, rate ):
    '''
    Configure the sample rate D for the ADC given by the formula
    D = 18000/rate
    '''
    if (self.stream_flag):
        self.stop_stream()
        time.sleep(.1)

    # compute the sample rate
    value = hex(18000/rate).replace('0x','')
    for i in range(4-len(value)):
        value = '0' + value

    # transmit configuration
    try:
        self.sock.send('d' + value + '\n')
        self.sample_rate = rate

    except:
        print 'Connection lost or device not connected'

def clear_recv_buffer( self ):
    '''
    Clears the receive buffer
    '''
    try:
        recv = self.sock.recv(10000)
        while len(recv) != 0:
            recv = self.sock.recv(10000)
    except:
        print 'Buffer cleared'


def set_channels( self, channels=4 ):

    try:
        self.sock.send(str(channels))
    except:
        print 'Connection lost or device not connected'

def close( self ):
    try:
        self.stop_stream()
```

```
            self.sock.send('~')
            self.sock.close()
        except:
            print 'Connection lost or device not connected'
```

A.3 hall.py

```
import adc
import copy
import time
import math
import thread

class Hall:

    def __init__( self ):

        try:
            self.sensor = adc.ADC()

            # sampling frequency
            # make sure the sensor object uses the same value
            # i.e. sensor.set_sample_rate(self.sample)
            self.sample_rate = self.sensor.sample_rate

        except:
            try:
                self.sensor.discover()
            except:
                'No device within range'

        # signal threshold (threshold between '0' logic and '1' logic)
        self.threshold = 1.0

        self.detections = {}
        self.detections['time'] = []
        self.detections['pos'] = []

        # bits/code (modify according to num of bits to expect)
        self.numbits = 12

        # meters/bit (current magnets are 1 cm long, however
        # may need to increase to account for normal distribution of
        # mag field and signal convolution
        self.bitlength = 0.015

        # velocity in meters/second
        # may need to determine conversion to rotations/second
        self.velocity = .1
```

```
        # channel numbers
        self.channels = [0,1]

        self.sample_rate = 200
        self.set_sample_rate(self.sample_rate)

        # verify this equation
        self.samplesperbit = self.bitlength * self.sample_rate / self.velocity
        self.samplespercode = self.samplesperbit * self.numbits

        self.direction = 1

        self.samplespercode = 90
        self.samplesperbit = 10

        self.current_pos = 0


        # load lookup table
        lookupfn = 'hall_lookup.txt'
        try:
            #print 'here'
            lookupfd = open(lookupfn,'r')
            lines = lookupfd.readlines()
            #print lines
            keys = []
            pos = []
            for line in lines:
                keys.append(line.strip().split('\t')[1])
                pos.append(int(line.split('\t')[0]))
                #print keys
                #print pos

            self.codes = {}
            for i in range(len(keys)-1):
                if i == 0:
                    self.codes[keys[i]] = ['',pos[i],pos[i+1]]
                elif i == len(keys):
                    self.codes[keys[i]] = [pos[i-1],pos[i],'']
                else:
                    self.codes[keys[i]] = [pos[i-1],pos[i],pos[i+1]]
                #print self.codes

        except:
            print 'Hall lookup file invalid'

    #def calibrate ( self ):
    def start_monitor( self ):
```

```
        self.monitor_id = thread.start_new_thread(self.monitor)

    def stop_monitor( self ):
        self.monitor_flag = 0

    def monitor( self ):
        '''
        monitors the output and looks for transition indicating a start bit
        '''

        self.sensor.start_stream(self.channels)
        time.sleep(1)

        sample = []

        self.monitor_flag = 1
        while(self.monitor_flag):
            self.start_time = self.sensor.data['time'][0]
            for channel in self.channels:
                sample.append(self.sensor.data[channel][self.sensor.data['time'].index(self.start_time)])
            #print min(sample)
            if min(sample) < self.threshold:
                self.scan()
                flag = 1
            time.sleep(1.0/self.sensor.sample_rate)
            sample = []


    def scan( self ):

        end = 0
        data = []
        while (not end):

            if self.sensor.data['time'].index(self.start_time) > 1.2 * self.samplespercode:
                data = copy.deepcopy(self.sensor.data)
                end = 1
                self.sensor.stop_stream()

        signal = []
        end = 0
        count = 0
        for i in range(data['time'].index(self.start_time)):
            sample = []
            for channel in self.channels:
                sample.append(data[channel][i])
            if min(sample) < self.threshold:
                signal.append(1)
            else:
```

```
            signal.append(0)

    if (self.direction):
        signal.reverse()

    print 'signal='
    print signal

    self.decode(signal)

def decode( self, signal ):

    # clean up signal, remove any spikes
    # e.g. based on samples/bit 111100011111111 -> 111111111111111
    # since 000 less than some percent of samples/bit

    #for i in range(1,len(signal)-):
    # if signal[i] != (signal[i-1] and signal[i] != signal[i+1]:
    # signal[i] = signal[i-1]

    val = 1
    zero_index = []
    # record index of each '0' bordered by a '1'
    for i in range(len(signal)):
        if signal[i] != val:
            if signal[i] == 0:
                zero_index.append(i)
            if signal[i] == 1:
                zero_index.append(i-1)
            val = not val

    # if length of '0' segment is less than some threshold,
    # replace with '1's
    while len(zero_index) >= 2:
        start = zero_index.pop(0)
        stop = zero_index.pop(0)
        if (stop - start) < 0.3 * self.samplesperbit:
            for i in range(start,stop+1):
                signal[i] = 1


    print 'signal='
    print signal

    code = ''
    val = 1
    index = 0
    count = 1
```

```
# 'squeeze' signal into corresponding bit pattern


while index < len(signal):
    if signal[index] != val or index == len(signal)-1:
        print 'val='+str(val)
        #print 'index='+str(index)
        print 'count='+str(count)

        if val == 1 and count > .3 * self.samplesperbit:
            for i in range(round(float(count)/self.samplesperbit)):
                code += '1'
        if val == 0:
            for i in range(round(float(count)/self.samplesperbit)):
                code += '0'

        val = signal[index]
        count = 1

        '''
        for i in range(round(float(count)/self.samplesperbit)):
            code += str(val)
        val = signal[index]
        count = 1
        '''

    else:
        count += 1

    index += 1

print 'code='
print code

if code[0] != '1':
    code = '1' + code

# if code length less than number of bits expected, pad with '0's
while len(code) < self.numbits:
    code += '0'

final_code = ''
for i in range(1,len(code)):
    if code[i-1] == '1' and (code[i] == '0' or code[i] == '1'):
        final_code += str(1)
    if code[i-1] == '0' and code[i] == '0':
        final_code += str(0)

final_code = final_code[0:6]
```

```
            print 'final_code='
            print final_code

            '''
            # verify position
            if (self.direction and self.codes[final_code][0] == self.current_pos) \
                or (not self.direction and self.codes[final_code][2] == self.current_pos):
                 self.detections['pos'].append(self.codes[final_code][1])
                 self.detections['time'].append(self.start_time)
            #else:
                 #if self.directionself.dections['pos'].append(self.codes[final_code][
            '''

            print self.codes[final_code]
            print self.start_time

            self.sensor.stop_stream()

            #self.monitor()

        def set_sample_rate( self, sample_rate ):
            self.sample_rate = sample_rate
            self.sensor.set_sample_rate(sample_rate)

        def close( self ):
            self.sensor.close()
```

A.4 sonar.py

```
import os
import serial
import time
import re
import numpy
import scipy.signal.bsplines as bsplines
import numarray.fft as fft
import scipy.stats as stats
import cluster
from ntimestamp import nimsTimeStamp

'''
Sonar class for Model 852 Ultraminiature Side-Scan Sonar from Imagenex Corp.
www.imagenex.com

Author: Victor Chen
'''

class sonar:
    def __init__( self, SER = None, PORT = None ):
```

```
if os.name == 'nt':
    print 'WINDOWS Based Machine'

    if PORT == None:
        PORT = 'COM1'

    if SER != None:
        self.ser = SER
    else:
        self.ser = serial.Serial(PORT, 115200)
elif os.name == 'posix':
    if PORT == None:
        PORT = '/dev/ttyUSB0'

    print 'UNIX Based Machine'
    if SER != None:
        self.ser = SER
    else:
        self.ser = serial.Serial(PORT, 115200, timeout=0.5 )

'initialize default sonar settings'
'Range = self.range'
'[5,10,20,30,40,50] meters'
self.range = 5

'Gain = self.gain'
'[0 to 40] db'
self.gain = 10

'Train Angle = self.train'
'[0 to 140] degrees'
'self.train = (train angle + 210) / 3'
self.train = 70

'Sector Width = self.width'
'[0 to 120] degrees'
'self.width = (width / 3)'
self.width = 0

'Step Size = self.step'
'[0,1,2]'
'self.step = 0 --> no step'
'self.step = 1 --> 3 degrees/step'
'self.step = 2 --> 6 degrees/step'
self.step = 0

'Pulse Length = self.pulse'
'[1 to 255] usec'
self.pulse = 10
```

```
        'Data Points = self.points'
        '[25, 50]'
        'self.points = 25 --> 252 data points'
        'self.points = 50 --> 500 data points'
        self.points = 50

        'Delay = self.delay'
        '[0 to 255] msec'
        'self.delay = delay / 2'
        self.delay = 10

        'Frequency = self.freq'
        '[0,135]'
        'self.freq = 0 --> 675 kHz'
        'self.freq = 135 --> 850 kHz'
        self.freq = 0

        'log filename'
        self.logfilename = 'sonar_'+str(time.time())+'.txt'


    '''
    transmits switch data command and receives a single sonar reading
    '''
    def sample( self, train=-1, timeout=.2, logfilename=-1 ):

        'if log filename is not specified, uses the default filename'
        'otherwise, default filename is changed to filename specified'
        if logfilename == -1:
            logfilename = self.logfilename
        else:
            self.logfilename = logfilename


        'if train angle is not specified, uses default train angle'
        'otherwise, default train angle is changed to train angle specified'
        if train == -1:
            train = self.train
        else:
            self.train = train


        'create empty array for sonar data'
        self.echodata=[]

        'check serial port'
        if self.ser.isOpen() == True:
            self.ser.close()
```

```
if self.ser.isOpen() == False:
    self.ser.open()

'send switch data command'
self.ser.write(chr(254))
self.ser.write(chr(68))
self.ser.write(chr(16))
self.ser.write(chr(self.range))
self.ser.write(chr(0))
self.ser.write(chr(0))
self.ser.write(chr(0))
self.ser.write(chr(0))
self.ser.write(chr(self.gain))
self.ser.write(chr(0))
self.ser.write(chr(20))
self.ser.write(chr(train))
self.ser.write(chr(self.width))
self.ser.write(chr(self.step))
self.ser.write(chr(self.pulse))
self.ser.write(chr(0))
self.ser.write(chr(0))
self.ser.write(chr(0))
self.ser.write(chr(0))
self.ser.write(chr(self.points))
self.ser.write(chr(0))
self.ser.write(chr(0))
self.ser.write(chr(0))
self.ser.write(chr(0))
self.ser.write(chr(self.delay))
self.ser.write(chr(self.freq))
self.ser.write(chr(253))

'read sonar return data'
print 'Reading'
time.sleep(timeout)
self.recv = self.ser.read(self.ser.inWaiting())

try:
    tries = 0

    'look for start of header'
    'if not found, wait and try again'
    while (self.recv.find('I') == -1 and tries < 2):
        time.sleep(timeout)
        self.recv = self.ser.read(self.ser.inWaiting())
        tries += 1
        print tries
```

```
        'look for end of termination byte'
        'if not found, packet truncated, wait and try again'
        if self.recv.find('\xfc') == -1:
            #print 'truncated'
            time.sleep(timeout)
            self.recv += self.ser.read(self.ser.inWaiting())

        'close the serial port'
        self.ser.close()

        'error checking'
        if self.points == 25:
            self.rawdata = self.recv[self.recv.index('I'):self.recv.index('I')+265]
        if self.points == 50:
            self.rawdata = self.recv[self.recv.index('I'):self.recv.index('I')+513]

        'convert hex string to integer values and store in sonar data array'
        self.data = []
        for i in range(0,self.rawdata.__len__()):
            self.data.append(ord(self.rawdata[i]))

    except:
        return False

    'calculate angle from data header'
    highbyte=(self.data[6]&0x3E)>>1
    lowbyte=((self.data[6]&0x01)<<7)|(self.data[5]&0x7F)
    headpos=(((highbyte<<8)|lowbyte)-1400)*.15

    'verify header is correct'
    train = train*3-210
    if headpos != train and headpos != train-360 and headpos != train+360:
        return False

    'verify termination byte present'
    if 252 not in self.data:
        return False

    'verify packet length'
    if not (len(self.data) == 513 or len(self.data) == 265):
        return False
    print 'length',len(self.data)
    self.data = self.data[:self.data.index(252)+1]


    'save to logfile'
    if logfilename != 0:
        logfile = open(logfilename,'a')
        logfile.write('1 ')
```

```
            logfile.write(nimsTimeStamp()+' ')
            logfile.write(str(self.range)+' ')
            logfile.write(str(self.gain)+' ')
            logfile.write(str(self.train)+' ')
            logfile.write(str(self.pulse)+' ')
            logfile.write(str(self.freq)+' ')
            for val in self.data:
                logfile.write(str(val)+' ')
            logfile.write('\n')
            logfile.close()

        return True


'''
performs signal processing on sonar data for range extraction
'''
def process( self, train=-1, samples=1, logfilename=-1, smoothness=20):

    'filter constants'
    min_range = 10
    nn_window = 10

    'determine log filename'
    if logfilename == -1:
        logfilename = self.logfilename
    else:
        self.logfilename = logfilename

    if train == -1:
        train = self.train
    else:
        self.train = train

    'collect samples'
    for i in range(samples):
        success = self.sample()
        while(success == False):
            success = self.sample()

        data = numpy.zeros(len(self.data[12:-1]),int)+numpy.array(self.data[12:-1])
        data = numpy.array(data)+numpy.array(self.data[12:-1])
    self.data_ave = numpy.array(data)/samples

    'compute spline'
    spline = bsplines.cspline1d(data,smoothness)
    sig = self.data[12:-1]

    'log data if logging enabled'
```

```
if logfilename != 0:
    logfile = open(logfilename,'a')

    '"2" indicates spline data'
    logfile.write('2 ')

    logfile.write(nimsTimeStamp()+' ')
    logfile.write(str(smoothness)+' ')

    'write placeholder data for data length consistency'
    for i in range(17):
        logfile.write('0 ')

    'write spline data'
    for val in spline:
        logfile.write(str(val)+' ')
    logfile.write('\n')

    'close logfile'
    logfile.close()


'identify maxima'
maxima = []
for i in range(1,len(spline)-1):
    if (spline[i] > spline[i-1]) and (spline[i] >= spline[i+1]):
        maxima.append(i)

'nearest neighbors filter'
remove = []
for k in range(len(maxima)-1):
    if maxima[k] > min_range:
        for m in range(k+1,len(maxima)):
            if k != m and abs(maxima[k]-maxima[m]) < nn_window:
                if spline[maxima[k]] > spline[maxima[m]] or sig[maxima[k]] > sig[maxima[m]]:
                    remove.append(maxima[m])
                else:
                    remove.append(maxima[k])
            if maxima[m]-maxima[k] > 100 and spline[maxima[m]]/spline[maxima[k]] < 2:
                remove.append(maxima[m])
    else:
        remove.append(maxima[k])

'eliminate nearest neighbors'
temp = []
for i in maxima:
    if i not in remove:
        temp.append(i)
maxima = temp
```

```
        'compute periodogram'
        X = fft.fft(spline)
        N = len(X)
        X = list(X)
        X.pop(0)
        X = numpy.array(X)
        power = numpy.square(abs(X[1:N/2]))
        freq = numpy.array(range(N/2))/float(N/2)/2
        period = 1/freq

        'find dominant period components'
        T = []
        n=1
        while len(T) < 4:
            if (power[n] >= power[n-1] and power[n] >= power[n+1]) and power[n] > max(power)/4:
                T.append(period[n])

            n = n+1
            if n > len(power)-2:
                break

        'multiple reflections filter'
        remove = []
        if len(maxima) > 1:
            for k in range(len(maxima)-1):
                for j in range(k+1,len(maxima)):
                    for m in range(len(T)):
                        if abs(abs(maxima[k]-maxima[j])-T[m]) < 20:
                            if maxima[j] not in remove:
                                remove.append(maxima[j])

        'remove maxima corresponding to multiple reflections'
        temp = []
        for i in maxima:
            if i not in remove:
                temp.append(i)
        maxima = temp


        self.spline = spline

        return maxima

    '''
    collects results from multiple sonar readings
    uses cluster analysis to determine range value
    '''
    def get_range( self, train=-1, samples=10 ):
```

```
        'collect range values from multiplle sonar readings'
        ranges = []
        for i in range(samples):
            ranges.extend(self.process(train))
        #print ranges

        'create hierarchical clustering'
        cl = cluster.HierarchicalClustering(ranges, lambda x,y: abs(x-y))
        #print cl.getlevel(5)
        clusters = cl.getlevel(5)
        clusters.sort()

        'remove small clusters'
        i = 0
        while(i < len(clusters)):
            if len(clusters[i]) < .4 * samples:
                clusters.pop(i)
            else:
                break
            i = i+1
        #print clusters

        'compute range estimate as modal value within largest cluster'
        val = int(stats.mode(clusters[0])[0][0])

        return val


    '''
    computes the full depth of the water column by estimating
    the range from the sonar to the water surface and to the subsurface
    '''
    def get_depth( self ):
        up = self.get_range(130)
        down = self.get_range(70)

        return up+down


    '''
    closes the serial port
    '''
    def close( self ):
        self.ser.close()
```

A.5 gridmap.py

```
import os
```

```
import time
import numpy
import sonar as sensor


class gridmap:
    def __init__( self ):

        'create sonar object'
        self.sonar = sensor.sonar()

        'initialize map'
        'resolution in m'
        if self.sonar.points == 50:
            self.res = float(self.sonar.range)/500
        elif self.sonar.points == 25:
            self.res = float(self.sonar.range)/252

        'resolution in cells/meter'
        self.res = 100

        'length of transect in m'
        self.length = 4

        'sonar range in m'
        self.range = self.sonar.range

        'map filename'
        self.mapfilename = 'map_' + str(time.time()) + '.txt'

        'NIMSRD position filename'
        self.posfilename = '/home/nims/cfg/position.txt'

        self.donefilename = '/home/nims/cfg/done.txt'

        self.cells = {}

        self.threshold_min = 10
        self.threshold_max = 15


        self.min_cell = (0,-self.range*self.res,-self.range*self.res)
        self.max_cell = (self.length*self.res,self.range*self.res,0)

        self.range_min = 20
        self.range_max = 40


    def scan( self, minangle=55, maxangle=85, samples=1, logfilename=-1, smoothness=20, xpos=0,zpos=0
```

```
'''
Scans the range specified by [minangle,maxangle], taking the
specified number of samples at each train angle

minangle and maxangle must be specified in terms of the sonar
value:
sonar_angle = (real_angle + 210)/3
'''


'sample'
for angle in range(minangle, maxangle):
    maxima = self.sonar.process(angle,samples,logfilename,smoothness)

    x=xpos

    'get the current x position'
    if xpos == 'fake':
        position = self.getpos()
        x = position[0]

    'determine x index of nearest cell'
    xi = int(xpos*self.res)

    theta = (angle*3-210)*numpy.pi/180 - numpy.pi/2

    'updated cells'
    updated = []

    print '\n'
    print 'angle='+str(theta*180/numpy.pi)
    print 'maxima='+str(maxima)



    for val in maxima:
        if val >= self.range_min and val <= self.range_max:
            'calculate y and z coordinates'
            y = val*numpy.cos(theta)/500*self.range
            z = val*numpy.sin(theta)/500*self.range

            'determine y index of nearest cell'
            yi = int(y*self.res)

            'determine z index of nearest cell'
            zi = int(z*self.res)
            print 'pos = '+str(val), 'val= ',self.sonar.data[12+val]
            print 'calc at '+str((x,y,z))
            print 'cell at '+str((xi,yi,zi))
```

```
            if (xi,yi,zi) >= self.min_cell and (xi,yi,zi) <= self.max_cell:
                '''
                if self.sonar.data[val+12] <= self.threshold_min and (xi,yi,zi) in self.cells.keys():
                    print 'val less than threshold and cell in keys'
                    if self.cells[(xi,yi,zi)] == 1:
                        print 'popping cell',(xi,yi,zi)
                        self.cells.pop((xi,yi,zi))
                    else:
                        print 'decrementing cell',(xi,yi,zi)
                        self.cells[(xi,yi,zi)] = self.cells[(xi,yi,zi)] - 1
                    updated.append((yi,zi))
                '''


                if self.sonar.data[val+12] >= self.threshold_max:
                    print 'val greater than threshold'
                    if (xi,yi,zi) in self.cells.keys() and self.cells[(xi,yi,zi)] < 127:
                        print 'incrementing cell',(xi,yi,zi)
                        self.cells[(xi,yi,zi)] = self.cells[(xi,yi,zi)] + 1

                    else:
                        print 'adding cell',(xi,yi,zi)
                        self.cells[(xi,yi,zi)] = 1
                    updated.append((yi,zi))


    print 'other cells not in maxima'
    'update other cells'
    for val in range(len(self.sonar.data[12:-1])):
        if val > self.range_min:
            'calculate y and z coordinates'
            y = val*numpy.cos(theta)/500*self.range
            z = val*numpy.sin(theta)/500*self.range

            'determine y index of nearest cell'
            yi = int(y*self.res)
            #yi = int(y*self.range/self.res)+self.MAP.shape[1]/2

            'determine z index of nearest cell'
            zi = int(z*self.res)

            if (yi,zi) not in updated:
                if (xi,yi,zi) >= self.min_cell and (xi,yi,zi) <= self.max_cell:
                    #print (xi,yi,zi)
                    #if self.sonar.data[val+12] <= self.threshold_min and (xi,yi,zi) in self.cells.keys()
                    if (xi,yi,zi) in self.cells.keys():
                        print 'val less than threshold and cell in keys'
                        if self.cells[(xi,yi,zi)] == 1:
```

```
                print 'popping cell',(xi,yi,zi)
                self.cells.pop((xi,yi,zi))
            else:
                print 'decrementing cell',(xi,yi,zi)
                self.cells[(xi,yi,zi)] = self.cells[(xi,yi,zi)] - 1


        '''
        elif self.sonar.data[val+12] >= self.threshold_max:
            print 'val greater than threshold'
            if (xi,yi,zi) in self.cells.keys() and self.cells[(xi,yi,zi)] < 127:
                print 'incrementing cell',(xi,yi,zi)
                self.cells[(xi,yi,zi)] = self.cells[(xi,yi,zi)] + 1

            else:
                print 'adding cell',(xi,yi,zi)
                self.cells[(xi,yi,zi)] = 1
            updated.append((yi,zi))
        '''



def save_map( self, mapfilename = -1 ):
    'determine map filename'
    if mapfilename == -1:
        mapfilename = self.mapfilename
    else:
        self.mapfilename = mapfilename

    'open file'
    fd = open(mapfilename,'w')

    for key in self.cells.keys():
        line = str(key[0])+' '+str(key[1])+' '+str(key[2])+' '+str(self.cells[key])+'\n'
        fd.write(line)

    'close file'
    fd.close()

def load_map( self, mapfilename = -1 ):
    'determine map filename'
    if mapfilename == -1:
        mapfilename = self.mapfilename
    else:
        self.mapfilename = mapfilename

    'open file, read file, close file'
    fd = open(mapfilename,'r')
    lines = fd.readlines()
```

```
        fd.close()

        self.cells = {}
        for line in lines:
            line = line.strip().split(' ')
            self.cells[(int(line[0]),int(line[1]),int(line[2]))] = int(line[3])



def clear_map( self ):
    self.cells = {}

def close( self ):
    self.sonar.close()


def move( self, xpos=-1, ypos=-1, vel=-1, acc=-1 ):

    'open move file'
    fd = open(self.movefile,'w')

    'check current position, velocity, acceleration'
    position = self.getpos()

    'if parameters not specified, use old paramters'
    if xpos == -1:
        xpos = position[0]
    if ypos == -1:
        ypos = position[1]
    if vel == -1:
        vel = position[2]
    if acc == -1:
        acc = position[3]

    'update move file'
    fd.write(str(xpos)+' '+str(ypos)+' '+str(vel)+' '+str(acc))

    'close move file'
    fd.close()

def getpos( self, pos=False ):


    if pos == False:
        'open position file'
        fd = open(self.posfilename,'r')

        'read position file'
        data = fd.readlines()[0].strip().split(' ')
```

```
        'close position file'
        fd.close()

        'return position information'
        return float(data[0]),float(data[1]),float(data[2]),float(data[3])

def checkdone( self ):
    'open done file'
    fd = open(self.donefilename,'r')

    'get status'
    status = int(fd.readlines().strip())

    'close done file'
    fd.close()

    'return True if done, False otherwise'
    if status == 1:
        return True
    else:
        return False
```

References

[blu07] Roving Networks. "http://www.rovingnetworks.com/bluesentry.htm." 2007. Online.

[BSB06] Per Henrik Borgstrom, Michael J. Stealey, Maxim A. Batalin, and William J. Kaiser. "NIMS3D: A Novel Rapidly Deployable Robot for 3-Dimensional Applications." In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Beijing, China, 2006.

[cyc] Cyclops 7 Submersible Fluorometer. "http://www.turnerdesigns.com/." Online.

[Dam80] J.E. Damuth. "Use of High-Frequency (3.5-12 kHz) echograms in the study of near-bottom sedimentation processing in the deepsea." *International Journal of Marine Geology*, **38**:51–75, 1980.

[Elf89] A. Elfes. "Using occupancy grids for mobile robot perception and navigation." *Computer*, **2**(6):46–57, 1989.

[FMB93] G. Forsgren, L. Malmgren, L Brydsten, and M. Jansson. "Characterization of Sediments by High-Frequency Echo-Sounding." *Environmental Geology*, **21**:14–18, 1993.

[GS89] J. A. Grant and R. Schreiber. "Modern swathe sounding and subbottom profiling technology for research applications: The Atlas Hydrosweep and Parasound Systems." *Marine Geophysical Researches*, **12**(1,2), 1989.

[hac] Hydrolab MS4a and DS5X. "http://hydrolab.com/index.asp." Online.

[HAG07] Thomas C. Harmon, Richard F. Ambrose, Robert M. Gilbert, Jason C. Fisher, Michael Stealey, and William J. Kaiser. "High-Resolution River Hydraulic and Water Quality Characterization Using Rapidly Deployable Networked Infomechanical Systems (NIMS RD)." *Environmental Engineering Science*, **24**(2):151–159, 2007.

[hal07] Sensor Solutions. "http://www.sensorso.com/home/part.php?part=HS-EHS1-F2-S8-R25&connectWireCode=R25-3." 2007. Online.

[isu] MBARI-ISUS. "http://www.satlantic.com/." Online.

[JBK07] Brett L. Jordan, Maxim A. Batalin, and William J. Kaiser. "NIMS RD: A Rapidly Deployable Cable Based Robot." In *IEEE International*

Conference on Robotics and Automation (ICRA), Rome, Italy, April 2007.

[KJ06] C. M. Kazezyilmaz-Alhan and M. A. Medina Jr. "Stream solute transport incorporating hyporheic zone processes." *Journal Of Hydrology*, **329**:26–38, 2006.

[MB01] David A. Mindell and Brian Bingham. "A High-Frequency, Narrow-Beam Sub Bottom Profiler for Archaeological Applications." *Proceedings of IEEE Oceans 2001 Conference*, 2001.

[ME85] Hans P. Moravec and Albert Elfes. "High resolution maps from wide angle sonar." In *IEEE International Conference on Robotics and Automation*, pp. 116–121, 1985.

[PBG05] R. Pon, M. Batalin, J. Gordon, M. Rahimi, W. Kaiser, G. Sukhatme, M. Srivastava, and D. Estrin. "Networked Infomechanical Systems: A Mobile Wireless Sensor Network Platform." In *IPSN*, pp. 376–381, 2005.

[pyt07a] Python Cluster Library. "http://python-cluster.sourceforge.net/." 2007. Online.

[pyt07b] SciPy Spline Library. "http://www.scipy.org/Cookbook/Interpolation." 2007. Online.

[Rei67] Christian H. Reinsch. "Smooth by Spline Functions." *Numerische Mathematik*, **10**(3), 1967.

[SBC07] Amarjeet Singh, Maxim A. Batalin, Victor Chen, Michael J. Stealey, Brett Jordan, Jason Fisher, Tom Harmon, Mark Hansen, and William J. Kaiser. "Autonomous robotic sensing experiments at San Joaquin river." In *IEEE International Conference on Robotics and Automation (ICRA)*, Rome, Italy, April 2007.

[SBK08] M. J. Stealey, M. A. Batalin, and W. J. Kaiser. "NIMS-AQ: A novel system for autonomous sensing of aquatic environments." In *IEEE International Conference on Robotics and Automation (ICRA)*, p. submitted, 2008.

[SBS07] Amarjeet Singh, Maxim A. Batalin, Michael J. Stealey, Bin Zhang, Amit Dhariwal, Beth Stauffer, Stephani Moorthi, Carl Oberg, Arvind Pereira, Victor Chen, Yeung Lam, Dave Caron, Mark Hansen, William J. Kaiser, and Gaurav Sukhatme. "Human Assisted Robotic

Team Campaigns for Aquatic Monitoring." *Journal of Field Robotics, Special Issue on Teamwork in Field Robotics*, pp. accepted, to appear, 2007.

[Sch98] Arthur Schuster. "On the Invesigation of Hidden Periodicities." In *Terr. Mag. Atmos. Elect.*, pp. 13–41, 1898.

[Sch04] Steven G. Schock. "Remote Estimates of Physical and Acoustic Sediment Properties in the South China Sea Using Chirp Sonar Data and the Biot Model." *IEEE Journal of Oceanic Engineering*, **29**(4), 2004.

[SL66] D. T. Smith and W. N. Li. "Echo-sounding and Sea-floor Sediments." *International Journal of Marine Geology*, **4**:353–364, 1966.

[soi07] Soil Core Sampler. "http://www.soilmoisture.com/prod_details.asp?prod_id=76&se 2007. Online.

[son07] Imagenex. "http://www.imagenex.com/Downloads/What_s_New/852_Ultra-Miniature/852_ultra-miniature.html." 2007. Online.

[stk] SonTek Argonaut-ADV. "http://www.sontek.com/product/aadv/aadvov.htm." Online.

[TMN02] A. Turgut, M. McCord, J. Newcomb, and R. Fisher. "Chirp sonar sediment characterization at the northern Gulf of Mexico Littoral Acoustic Demonstration Center experimental site." *Oceans '02 MTS/IEEE*, **4**, 2002.

[VFS90] H. M. Valett, S.G. Fisher, and E. H. Stanley. "Physical and Chemical Characteristics of the hyporheic zone of a sonoran desert stream." *Journal of the North American Benthological Society*, **9**(3):201–215, 1990.