Decentralized Sensor Placement and Mobile Localization on an Underwater Sensor Network with Depth Adjustment Capabilities

by

Carrick Detweiler

B.A., Middlebury College (2004)
M.A., Massachusetts Institute of Technology (2006)

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

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2010

Author .....
Department of Electrical Engineering and Computer Science
July 15, 2010

Certified by .....
Daniela Rus
Professor
Thesis Supervisor

Accepted by .....
Terry P. Orlando
Chairman, Department Committee on Graduate Theses

Decentralized Sensor Placement and Mobile Localization on an Underwater Sensor Network with Depth Adjustment Capabilities

by

Carrick Detweiler

Submitted to the Department of Electrical Engineering and Computer Science
on July 15, 2010, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

Over 70% of our planet is covered by water. It is widely believed that the underwater world holds ideas and resources that will fuel much of the next generation of science and business. Unfortunately, underwater operations are fraught with difficulty due to the absence of an easy way to collect and monitor data. In this thesis we propose a novel underwater sensor network designed to mitigate the problems of underwater sensing and communication. A key feature of this system is the ability of individual nodes to control their depth in water. This single degree of freedom allows the network to cooperatively optimize placement for communication and data collection while minimizing time and energy use. The sensor network also enables a GPS-like system for localizing underwater robots to aid in data retrieval and sensing.

We develop a gradient-based decentralized controller that dynamically adjusts the depth of a network of underwater sensors to optimize sensing for modeling 3D properties of the water. We prove that the controller converges to a local minimum, and implement the controller on our underwater sensor network, where each node is capable of adjusting its depth. We verify the algorithm through simulations and in-water experiments.

Most applications require that we associate a location with the sensed data. We have developed an underwater mobile robot localization algorithm that allows underwater robots to act as mobile sensors in the sensor network by using ranging information. The algorithm is a minimalist, geometric-based algorithm that only relies on knowing an upper bound on the robot speed and known static node locations. We prove that the algorithm finds the optimal location of the robot and analyze the algorithm in simulation and in water with our underwater sensor network.

Thesis Supervisor: Daniela Rus
Title: Professor

Acknowledgments

I owe a great many thanks to a number of people for their help with this project. This work would not have been possible without the above-and-beyond support of my adviser, Daniela Rus. She supported and encouraged me throughout my graduate career, teaching me how to make the most out of my research. More than anything, Daniela taught me that if you can dream it, you can make it possible. She stood by me and always made possible the projects I pursued that had little clear relation to my main research. Her positive outlook and enthusiasm are contagious, and I will sincerely miss our conversations and travels. I am, however, reassured that you never really leave Daniela's Distributed Robotics Lab (DRL), and I know we will have many fruitful collaborations in the future.

I also want to thank my thesis committee, John Leonard and Sam Madden. Their feedback and suggestions greatly improved this project. They also have given me ideas to think about as I go forward, which will certainly aid the beginning of my academic career.

My work has been funded in part by DSO, NSF grant IIS-0426838, Sinapore DSTA, EFRI-0735953, and the MURI Antidote project. I value their support and contribution to this work. They made the development of the systems possible as well as many of the field experiments.

I have been extremely fortunate to be able to perform numerous experiments in real-world conditions as well as tests in local pools and rivers. I performed countless test experiments in the MIT pools and would like to thank all the lifeguards and staff at MIT Aquatics for their assistance. The staff at the MIT Sailing pavilion have always been supportive and willing to make room for my experiments in the Charles River.

I am also grateful for the times I have been able to spend at the UC Berkeley Gump Marine Biology Station in Moorea, French Polynesia. Michael Hamilton made these trips possible, and the scientists and staff at the station were always welcoming and helped us develop algorithms and platforms that are useful to the scientific

community.

I also have had the wonderful opportunity work with Bob Chen, Bernie Gardner, Mingshun Jiang, and Francesco Peri from UMass Boston. We worked on developing algorithms and models that are applicable to scientific applications. In addition, they enabled a deployment in the Neponset river that yielded interesting results from both a mobile sensor network and scientific perspective.

There are too many people in CSAIL at MIT that I would like to thank. Henry delivered more packages to me than I can possibly count. The wonderful groups in OPS and TIG, in particular Anthony Zolnik and Jack Costanza, were always quick to respond to any infrastructure and technology problems I encountered, especially when the roof started to rain on me. Ron Wiken maintains a wonderful stockroom, which I could only dream of as a child. I made use of it many many times when an order from McMaster was just too slow. He also cares for the machine shop that I spent many nights in.

There are, of course, many people in DRL that made my work possible. Iuliu Vasilescu and I worked extensively on developing the underwater sensor network and robot. He taught me much of what I know about electronics. He also taught me that anything is possible; it is always worth trying because most of the time you will succeed. We traveled together to more places than I can count. Marek Doniec has also been instrumental in developing and testing the underwater systems. His enthusiasm for work and travel are refreshing, and it is always a pleasure to work and play with him.

I worked with Marty Vona on many projects over the years. He set the bar extremely high for quality and detail in coding and analysis, and whenever I write any code, I always ask myself what Marty would think of it. Mac Schwager helped me with a number of mathematical challenges, including providing some inspiration for the decentralized depth controller in this thesis.

Elizabeth Basha and I sat next to each other for a number of years. We collaborated on a variety of projects, but more importantly, she was always willing to talk though the problems I encountered. In addition, she helped me (re)write numerous

papers and parts of this thesis. Even though we will now be sitting half-way across the country from each other, I know that we will continue to collaborate and help each other out. There are also many other past and present members of the Distributed Robotics Lab and CSAIL that I would like to thank, including Alex, Brian, Daniel, Eduardo, Jan, Keith, Kyle, Lauren, Nikolaus, Paulina, Sejoon, Seungkook, Stefan, Steve, Wil, and many others. Kathy Bates joined our lab just in the past year, but she has had a wonderful influence on our group, and I will miss her smile as I walked into the lab everyday and much more.

I would not be who I am today without the positive influence of my parents, Carol and Rick Detweiler, and my sisters Natasha Detweiler and Jerusha Detweiler-Bedell. They provided support and guidance throughout my life and my time at MIT. They were always excited by what I was working on and were always there for me, but they understood that sometimes I just didn't want to talk about work. I can't thank them enough.

My in-laws, Jane and Moose Hillebrecht, as well as "Mama" Fontanella have also been extremely supportive through this long process. It was always nice to get out of the city for a little while and enjoy a home-cooked meal, not to mention all of the times they took care of Cooper while Courtney and I were out of the country. Cooper, my dog, has helped keep life in perspective. We go on the same walk twice a day, but he is always excited to go. He has helped me appreciate the small things in life and made sure that I took some breaks from my work to enjoy playing ball and going on walks.

And more than I can ever express, I am grateful for the love and support of my wife, Courtney Hillebrecht. She is always there for me when I need to talk or when I need help proofreading. She always encouraged me and picked up the slack when I was overwhelmed, despite the fact that she was finishing her Ph.D. at the same time. She is my best friend and much, much more. Finally, our daughter, Nola Hillebrecht Detweiler is the most wonderful thing to ever happen to me. I am thankful that she delayed coming until after my defense, but I don't know what life would be without her. To Nola, I dedicate this thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

More than 70% of our planet is covered by water, and it holds ideas and resources that will fuel much of the next generation of science and business. However, underwater operations are fraught with difficulties due to the absence of an easy way to collect and monitor data. Although underwater sensors do exist and can alleviate some of the problems associated with manual data collection, they are not networked, and their use introduces many additional challenges:

- Deploying, retrieving, and using the sensors is labor intensive;
- Collecting the data is subject to very long delays;
- Radio does not work underwater and acoustic communication bandwidth is limited (300 bits per second);
- GPS does not work underwater;
- Power is limited to the batteries carried;
- The manual aspects of the sensors leads to error;
- The spatial scope for data collection with individual sensors is limited;
- Individual sensors are unable to perform operations that require cooperation, such as tracking relative movement and locating events.

These challenges have limited research, industrial, and military underwater applications and kept them small and narrow in scope. What is required is a low-cost, versatile, high-quality, easily deployable, self-configurable platform for underwater sensor

networks that will (a) automate data collection and scale-up in time and space, (b) speed-up access to collected data and (c) be easy to use.

Easily deployable and usable underwater sensor networks promise to increase the scope and efficiency of underwater operations through automation and communication. They will enable many applications that are now difficult or impossible, such as mapping the ocean floor, modeling coral reefs, monitoring pollution, prospecting for natural resources, and monitoring the safety and security of harbors. Marine biologists and environmental monitoring organizations, for example, could deploy sensor networks to collect precise data with higher temporal and spatial density and far less labor intensity than current alternatives.

In this thesis we propose a novel underwater sensor network. The network enables algorithms to improve sensing, provides localization information to mobile underwater robots, and allows near real-time feedback to scientists. A key feature of the sensor nodes, called AquaNodes, is a depth adjustment system that allows the sensor nodes to change their depth in the water column. More specifically, the research in this thesis investigates:

- A decentralized algorithm to optimize depths for sensing and data inference;
- A localization algorithm for mobile underwater robots;
- The development of the underwater sensor network hardware platform with depth control;
- Field experiments in the context of underwater monitoring applications;

The nodes are anchored and move up and down in the water column using a depth adjustment system. The AquaNodes use this depth degree of freedom to cooperatively configure the nodes of the system to optimize sensing and data collection. In addition to enabling algorithms for optimizing sensing, the depth adjustment system allows:

- Easy deployment and retrieval of the nodes, as they can adjust their depth to return to the surface on command;
- Radio and GPS use at the surface;
- Sampling and sensing the full water column.

Algorithmically, we utilize the depth adjustment system to develop and analyze a decentralized depth control algorithm that coordinates the one degree of freedom in each of the nodes in our 3D sensor network to improve the positioning for data collection for data field reconstruction. This algorithm is provably convergent and requires little communication and processing.

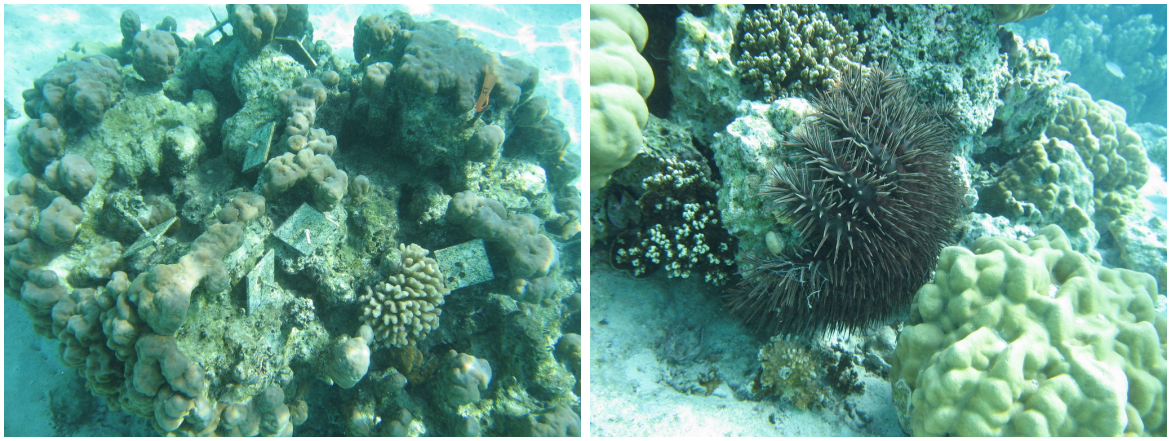
In addition to the decentralized depth adjustment algorithm, we have also developed an underwater mobile robot localization algorithm, which runs on our underwater network. Using this algorithm the network can act as a GPS-like system for underwater vehicles and divers (since GPS is unavailable underwater). This algorithm address the challenges of underwater localization. For example, mobile nodes only get ranges from the network every few seconds, this is sufficient for the algorithm. In addition, as dead-reckoning information is difficult to obtain underwater, dead-reckoning is not needed for the algorithm.

We extensively analyze and test both of these algorithms in simulation and in lab, pool, and real-water experiments. Simulations stress the algorithms under varying conditions and setups. The lab and pool tests verify the functionality of the algorithms on the hardware platform and the deployments in rivers, lakes, and oceans test the algorithms in real-world environments.

The rest of this chapter is organized as follows. Section 1.1 gives examples of current environmental sensing systems and approaches. Section 1.2.1 gives an overview of the sensor network and robot platform. This is followed by an overview of the algorithms developed in this thesis (Section 1.2.2) and the corresponding simulations and experiments (Section 1.2.3). Finally, Section 1.3 gives an overview of the contributions, and Section 1.4 outlines the rest of the thesis.

1.1 Current Environmental Sensing

One example where underwater sensors networks are needed is coral reef environments. Coral reefs are extremely sensitive ecosystems, and many have died-out recently due to subtle changes in the environment. Scientists are still trying to understand the impact of natural and human-induced changes on these ecosystems. Marine



(a) (b)

Figure 1-1: (a) Coral reef with tiles placed by a scientist to monitor growth. (b) Crown-of-thorns starfish attacking the reef.

biologists have found that pollution and minor temperature variations can have devastating impacts on coral reefs. Any improvements in the temporal and spatial fidelity of the measurements helps scientists detect, react, and learn about the impact on coral reef ecosystems.

Consider, for example, the coral reef off the island of Moorea, French Polynesia. This reef is a varied and complex ecosystem, which requires frequent observation and measurement. As such, the University of California, Berkeley maintains the Gump Marine Biology Station on the island to study the coral reef [1]. The biologists there make use of a variety of techniques to study the environment, however, all are labor intensive and/or result in a large lag between data collection and retrieval.

To study the growth of corals at different locations the marine scientists deploy small square tiles (see Figure 1-1(a)). The biologists periodically retrieve the tiles and bring them into the lab to photograph and measure growth. This is an extremely labor intensive process. A diver recovers and analyzes about 20 tiles a day and at any time there are over 200 tiles deployed. Due to the large time commitment, the tiles typically are only analyzed a few times a year. Automating the tile monitoring and data collection has the potential to greatly reduce manual labor involved in deploying and recovering tiles, thereby freeing scientists to study and analyze the

collected data. In addition, with an underwater sensor network, the data can have much higher spacial and temporal data. For instance, tiles can be photographed hourly to capture time-varying properties that scientists would otherwise miss.

Another case where scientists and local communities can be aided by advanced underwater sensor networks is in detecting and monitoring the explosive growth of crown-of-thorns starfish in coral reef environments. Figure 1-1(b) shows a crown-of-thorns attacking a coral reef in Moorea. The crown-of-thorns is an invasive species that eats coral and has few predators. As such, an infestation of crown-of-thorns can devastate coral reefs. Being able to actively monitor coral reefs for crown-of-thorns and count their numbers would help local communities react appropriately to potential outbreaks.

Monitoring and studying coral reef and other underwater environments is currently performed using a number of methods: (1) divers manually observing and measuring; (2) data loggers on moorings; (3) remotely operated vehicles (ROVs); or (4) autonomous underwater vehicles (AUVs). Collecting data manually is time-consuming and prone to errors. Static sensors, often at the surface or at a fixed depth are one of the most common methods for collecting time-series data in the ocean. These are only able to capture data at a single point in the water column. To obtain a larger spatial sampling, oceanographers must deploy multiple buoys. The buoy's depth is determined a-priori and they must be positioned precisely, otherwise the sensing coverage of the region may not be ideal. Current coverage of bodies of water is minimal. For example all of Massachusetts Bay is covered by only three NOAA surface buoys to measure tide, temperature, wind, etc. [4].

At the other end of the spectrum, ships, ROVs, and AUVs are used as sensing platforms. These are expensive and have short deployment times. While they obtain samples over large geographic regions, it is impractical to leave them in one location for large periods of time. Water column profiling sensors fall between moored buoys and ROVs/AUVs in terms of spatial coverage and capabilities. These are moored in a fixed location but are able to change depth to obtain measurements at numerous positions. Further, profilers have fixed schedules and do not communicate with other

sensors to determine optimal profiling trajectories. The operator decides the schedule for profiling before the deployment, determining the trade-off between battery life and longevity of the system.

1.2 Advancing Environmental Sensing

In this thesis we mitigate the problems of traditional environmental sensing approaches by combining the strengths of AUVs and moored profiling sensors and adding networking and computation capabilities. We propose and implement a novel underwater sensor network and robot system with supporting algorithms to aid scientists. The network utilizes the networking and computation capabilities to create a powerful, adaptive system. This system of hardware and algorithms addresses many of the shortcomings of current state of the art underwater sensor systems. Our system is:

- Easy to deploy: the nodes can be thrown off the side of a boat;
- Able to adjust the sensor node depths using a winch-based depth adjustment system to improve positioning for reconstructing sensory fields;
- Statically localized using a static acoustic localization algorithm or by surfacing to use GPS;
- Dynamically localized to allow the robot moving through the network to obtain position information for collecting sensor information and finding the sensors;
- Networked acoustically to provide low-bandwidth, real-time information from the system;
- Networked via radio at the surface to provide periodic higher-bandwidth feedback;
- Networked optically to allow the robot to download large data payloads from the sensor nodes.

Deploying traditional sensor systems in coral reef environments requires divers to precisely position the sensors so the scientist knows where the data is coming from and so the sensor can be located for recovery. Our system, on the other hand, is easily deployed by tossing the sensor nodes off the side of a small boat. The sensor nodes then localize themselves by either going to the surface using the depth adjustment

system and then using GPS or by obtaining inter-node ranges and running a static localization algorithm. Once the nodes are positioned, they use an acoustic modem to communicate and run a decentralized depth adjustment algorithm to improve their sensing positions for reconstructing sensory fields. If they detect an abnormal change in the sensor readings, perhaps an increase in pollution, for example, the sensor nodes can contact the shore and request further investigation by the robot.

The robot localizes itself in the network using the mobile robot acoustic localization algorithm. This allows it to position itself in the area of increased pollution to take more detailed readings. It may, for instance, take water samples for later analysis or take photos to determine if the pollution has caused coral reef bleaching–a sign of poor reef health. The robot also can download data directly from the sensor network nodes to obtain the detailed sensor data time-series from the past weeks or months.

This system greatly improves upon the current state of the art in underwater sensing by creating a flexible and powerful network of sensors and robots. Leveraging communication, computation, and mobility allows the system to adapt to changing conditions. In addition, the system provides feedback to scientists in near real-time so they can make informed decisions. In the rest of this section we give an overview the platform, algorithms, and experiments developed in this thesis.

1.2.1 Platform Overview

Figure 1-2 shows a picture of our underwater robot, AMOUR, and the underwater sensor nodes, AQUANODES. All are triple-networked. They have: (1) acoustic communication for long-range low-bitrate; (2) radio for above-water long-range medium-bitrate; and (3) optical communication for short-range high-bitrate data transfer. This thesis focuses on the development of the underwater sensor network platform. We give an overview of AMOUR and note that many of the specifics from the AQUANODES apply directly to AMOUR. They were developed in parallel and share many common components. For instance, the communication, GPS, and logging facilities on AMOUR are all components shared with the AQUANODES (often an AQUANODE is placed externally on the robot for these purposes).

Figure 1-2: Picture of AMOUR with AQUANODES.

The AQUANODE is cylindrically shaped with a diameter of 8.9cm and a length of 25.4cm. It weights 1.8kg and is 200g buoyant. It contains the three communication systems, GPS, and on-board batteries, as well as logging, sensing, and processing capabilities. All AQUANODES contain temperature and pressure sensors. In addition, they have inputs for high-precision analog sensors, as well as a variety of digital sensor interfaces. The on-board batteries allow for deployments ranging from 2 days with constant acoustic communication to months with only periodic communication and sensing.

At the heart of the AQUANODES is a generic sensor network platform hardware and software system we designed in part for this thesis. The platform was designed as a flexible, multi-functional sensor network platform that enables a wide range of heterogeneous applications in the air, on the ground, and in the water. It provides support for a wide range of communication and sensor systems not found in other sensor network systems.

In addition to the AQUANODES, the platform is also used in a variety of other projects such as river flood monitoring [14] and cow virtual fencing [35, 87]. The software is a custom-designed, extensible, multi-tasking, non-preemptive schedule-

based operating system. Over 80% of the code used for the AquaNodes is code that is core to the sensor board software system.

The underwater robot, Amour, is cylindrically shaped with a diameter of 17.8cm and a length of 72.4cm. It weighs approximately 17.5kg and is 3kg buoyant, depending on the configuration. A buoyancy and docking module can be attached to the robot to allow it to deploy and pick up AquaNodes. The on-board Lithium-Ion battery allows for operation of over 8 hours. It has inputs for a variety of sensors, including camera systems.

1.2.2 Algorithms Overview

We develop and analyze two different algorithms in this thesis. The first is a decentralized, adaptive algorithm for adjusting the depths of the AquaNodes to optimize their placement in support of computing maximally detailed models of the full 3D volume of the water. The algorithm is a gradient descent-based algorithm with guaranteed stability properties. We prove that the controller converges to a local minimum of the system.

The decentralized depth adjustment algorithm positions the nodes so that they are in good locations to collect data to model the values of the system over the whole region, not just the particular points where there are sensors. This is critical as the size of the ocean prohibits a brute-force deployment of enough sensors to cover all points of interest. The sensor nodes use a covariance function that describes the relationship between the possible positions of the sensor nodes and the region of interest. Because the algorithm is computationally efficient and requires little memory, we are able to implement the algorithm on the AquaNodes.

The second algorithm is an acoustic mobile robot localization algorithm that localizes the robot underwater where GPS is unavailable. The localization algorithm is a minimalist, geometric-based algorithm with provable optimality. It uses periodic, asynchronous range measurements to the AquaNodes. We prove that the location regions found are optimal given the range measurements. That is, the regions are the smallest regions where all points are reachable by the robot.

The challenge in underwater localization is that the ranges are obtained separately over time and the robot may move many meters between range measurements. In addition, expensive dead-reckoning systems are cost-prohibitive on our inexpensive robot platform. As such, the only information available to the localization algorithm is an upper bound on the maximum speed the robot could travel. The algorithm handles these constraints and is computationally efficient. In practice, it runs in constant time per location update.

1.2.3 Experiments Overview

In addition to theoretical analysis of the decentralized depth adjustment and localization algorithms, we also performed numerous simulation and in-water experiments. For the depth adjustment algorithm we perform a number of simulations to show that the controller finds configurations at or very near the global minimum of the system. In addition, we compare the algorithm to entropy minimization approaches as well as a centralized solver. We show that entropy and similar algorithms require storage space and computation time that exceed the capacity of the AquaNodes and other embedded systems. Similarly, we find that centralized solvers require significantly more computation time then our decentralized algorithm. We also examine the various algorithm parameters and show that for reasonable and a wide range of values the algorithm converges quickly.

We then use a data-driven physics-based model of the Neponset river, which feeds into Boston harbor, to numerically derive a covariance model that describes the relationship between different points in the water for use by the decentralized depth adjustment algorithm. We implemented this and other models on the AquaNodes and show in pool and river experiments that the controller converges rapidly with this real-world model. We also test changing covariance models and show that the controller converges and performs well when changing covariance functions.

We tested the underwater mobile robot localization algorithm in simulation and in rivers, lakes, and oceans. In simulation, we studied the effect of a number of parameters, including ranging rate, error, number of nodes, and maximum node speed.

We characterize the errors in these cases and show that the error is reasonable and the algorithm performs as expected. We implemented a discretized, grid-based version of the localization algorithm on AMOUR. We performed numerous experiments showing that the algorithm performs well in rivers, lakes, and oceans. We compare the results to GPS ground-truth gathered by driving the robot at the surface. The errors in the acoustic localization algorithm are typically under 2.5 meters. This error is well within the expected noise in the GPS reading.

1.3 Thesis Contributions

This thesis contributes to the fields of sensor networks and underwater robotics. The underwater sensor network is the first underwater sensor network developed and deployed that uses communication and computation capabilities to improve sensing and data retrieval. In particular, this thesis develops a depth adjustment algorithm to improve sensor placement and a localization algorithm to enable underwater robots to participate in data collection and retrieval. This thesis makes a number of theoretical contributions, including:

- A decentralized depth adjustment algorithm for improving the positioning of sensor nodes for sensing. The algorithm is provably convergent and stable;
- Underwater mobile robot localization algorithm with minimal assumptions and requirements;
- Analysis of energy trade-offs between the three communication methods (acoustic, radio, and optical).

This thesis also contributes to the field from a system development and experiment perspective:

- Development of a generic sensor network hardware platform that is used for AQUANODES and in a variety of other projects;
- Development of a flexible sensor network operating system;
- Implementation of a sensor network operating system that is easily customizable for a variety of projects;
- A novel underwater sensor network platform, called AQUANODES;

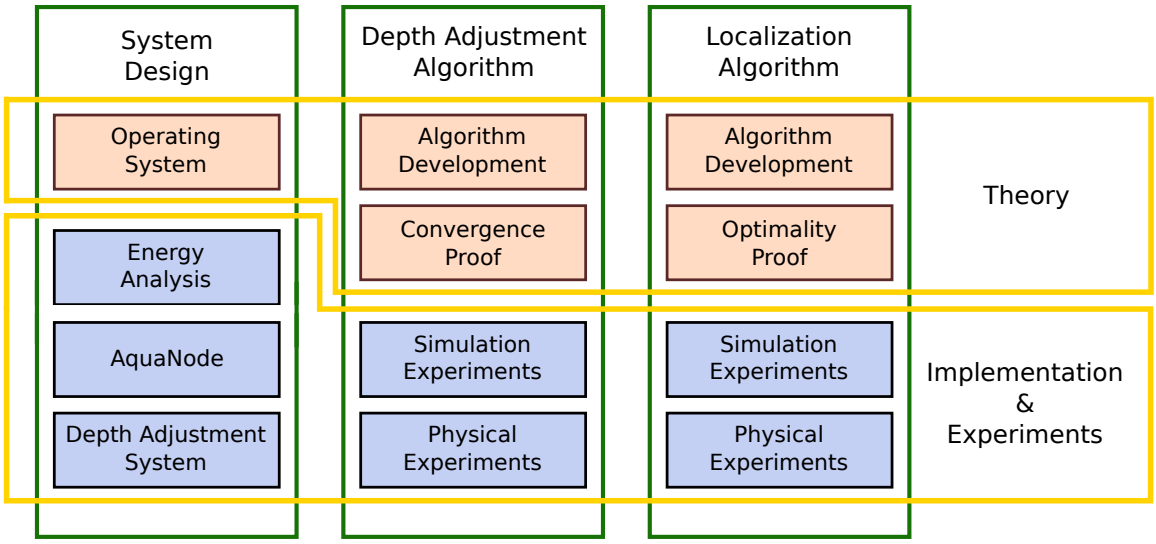


Figure 1-3: Thesis overview.

- A novel winch-based depth adjustment system for the AquaNodes;
- Simulations verifying the depth adjustment algorithm;
- Experiments and implementation of the decentralized depth adjustment algorithm on the AquaNode platform. These experiments verify the algorithm under real-world communication, motion, and sensing;
- Simulations verifying the localization algorithm;
- Experiments and implementation of the localization algorithm showing its performance on embedded hardware in-situ.

1.4 Thesis Outline

This thesis develops an underwater sensor network and develops novel algorithms that improve sensing and enable localization of underwater vehicles. Figure 1-3 shows a block diagram outline of this thesis. There are three main components to this research. The first is the design of a novel generic sensor network hardware and software system and the specific AquaNode implementation with the depth adjustment system. The second is the development of two novel algorithms that give underwater sensor networks novel capabilities for placing the nodes optimally for the purpose of

data collection (the decentralized depth adjustment algorithm) and for computing the location of mobile nodes in the network (the localization algorithm). Each of these components include theory and implementation sections. This is reflected in the layout of this thesis.

We start by detailing related work in Chapter 2. Following this, we discuss our generic sensor network platform and the specific AquaNode implementation and analyze the platform and communication systems in Chapter 3. Chapter 4 introduces the decentralized depth adjustment algorithm and proves algorithm convergence. Chapter 5 follows the presentation of the algorithm and characterizes and verifies the algorithm using detailed simulations and pool and river experiments. In addition, this chapter utilizes a covariance model based on a physics-driven hydrodynamic river model. Chapter 6 introduces the underwater mobile robot localization algorithm and proves its optimality. Next, Chapter 7 analyzes the localization algorithm using simulations and real-world experiments. Finally, we conclude and discuss lessons learned in Chapter 8.

Chapter 2

Related Work

2.1 Overview

The technology to develop underwater sensor networks has only recently become available. Acoustic communication, a key component of underwater networks, has been a focus of underwater research since World War II [96]. However, the thrust of acoustic communication underwater was, for a long time, focused on point-to-point systems, enabling communication between submarines and ships. These systems were bulky and required large amounts of power. It was not until the past decade or so that acoustic communication was used to create networks of autonomous underwater sensors. Our AquaNode platform is among the first underwater sensor networks with the ability to communicate, sense, and act autonomously.

There are, however, a number of theoretical challenges that must be overcome in order to make underwater sensor networks useful to scientists. The first is to develop algorithms and techniques for optimally placing the sensors to collect data. This is critical in underwater systems as it is unlikely that there will ever be enough sensors to fully cover the underwater environment. Thus, choosing positions is critical. A second theoretical challenge is to develop algorithms that are able to localize static and mobile nodes underwater. Having localization data is critical to give position information to the sensed values. This is difficult with inexpensive underwater systems that do not

have expensive inertial sensors that cost orders of magnitudes more than the devices themselves. In this thesis we develop solutions to both of these theoretical challenges.

A key feature of the AquaNode platform is its ability to dynamically adjust its depth in the water. We use this ability to optimally place the sensors to gather and reconstruct sensory data. Few researchers have explored the optimal placement of sensors underwater using depth adjustment. There is, however, much work in optimizing placement for sensors on land. In the sensor network community, these systems typically preplan the positions before deployment. In the distributed robotics community, the robots are typically assumed to have full 2D or 3D degrees of freedom to plan their paths for sensing. Some of these trajectories are preplanned, although, many are adjusted online. Our work falls between a fully autonomous and a pre-planned system in that our system has very constrained motion (more like a sensor network) but distributedly plans the placement and motion online.

Researchers have studied underwater localization extensively and it is still an active area of research. The motivation from this work comes from Naval research in localizing vehicles and targets underwater. Early systems, which are still used, employ statically deployed buoys that respond to pings from the moving vehicle. These systems are not networked. Our system takes advantage of the networking capabilities of the sensor network to share information on locations and ranges in the network.

In this chapter we go into the details of previous work related to this thesis. We start by discussing other underwater sensor network systems. We then discuss prior work in sensor placement for optimizing sensing. Finally, we present other approaches to underwater localization.

2.2 Prior Work in Underwater Sensor Networks

At the heart of our underwater sensor node is a generic sensor network platform that was developed largely for this thesis. We build on several years of important work in designing and fielding sensor network systems on a variety of platforms including the Mote [6, 51, 72, 73, 79], Fleck [105], Cricket [83], Meraki [15, 70], and others [37, 54,

60]. We start this section by describing a small subset of available research sensor network platforms as well as a selection of applications in which they are used. We then discuss particular underwater sensor network systems.

The Berkeley Mote was one of the first widely used sensor network platforms [51]. Many systems descend from the original Berkeley Mote. These range from the Intel Mote, operating on a 12 MHz ARM7 CPU and with increased RAM [73], to the Telos Mote, designed to be extremely low power [79]. The Epic platform provides an open-source platform designed to be modular and hierarchical [37]. Extending the Mote concepts but with new platforms, the MIT Cricket is a sensor network node that adds the capability to obtain ranges between sensors [83] and the CSIRO Fleck node includes solar charging capabilities as well as a longer range radio [105].

Most Motes run TinyOS, an operating system developed for use on the original Berkeley Mote [68]. Its small footprint and processor usage come from its heritage of operating on processors with relatively limited capabilities. Our operating system takes a similar approach to TinyOS in that the core is based on a non-preemptive multi-tasking scheduler. However, our operating system takes advantage of the greater memory and processing capabilities of our platform. In addition, TinyOS is abstracted in such a way that attempts to be accessible to scientists that have little programming experience. We do not abstract to this level and as such give programmers more flexibility in implementation decisions.

Another class of sensor network nodes handles demanding processing tasks, such as image processing, but at the cost of using significantly more power. The Intel Mote 2 uses an Intel XScale core which can be clocked at up to 416 MHz [6, 72]. The Meraki sensor network [70] (a commercial spinoff of the MIT RoofNet project [15]) runs linux and can form adhoc 802.11 mesh networks. Cell phones have also been used as sensor networks [60]. In the AquaNodes the acoustic modem has a 600MHz DSP processor that can handle demanding processing tasks such as the signal processing for acoustic communication. This processor is shutdown when these processing capabilities are not needed, which saves energy. This multi-tiered approach enables deployment times of months while still enabling complex processing tasks.

While all of these platforms provide good options for sensor network research, none supports a full range of communication and sensing options while also supporting complicated algorithms. As these are vital requirements for the AquaNode platform, we developed our own hardware and software platform that enables the study of underwater environments.

The deployment of underwater sensor systems provides the opportunity to study the ocean at a scale never before possible. One method used to study the underwater environment is deploying trial and longer-term "ocean observatories." The Mediterranean Operational Oceanography Network (MOON) aims to create an ocean observatory in the Mediterranean Sea for monitoring and forecasting weather, environmental monitoring, and marine research [3]. MOON is part of the Global Ocean Observing System (GOOS) which was created in the 1980s to monitor all of the world's oceans [11]. GOOS has achieved 60% of their target ocean coverage with sensors placed mostly on the surface. While these sensors provide good whole-ocean coverage, they do not provide the detailed measurements that are desirable in many areas. The Ocean Observatories Initiative (OOI) combines deep-sea buoys, cabled underwater networks, as well as independent AUVs and sensors [12, 55]. The goal is to provide high resolution information in certain areas and also to provide coverage for GOOS in the United States. Patrikalakis *et al.*developed the Poseidon system that allowed easy exchange and search of oceanographic information, including the ability to forecast and adaptively sample the ocean [78].

A number of smaller scale ocean observatories also have been deployed to better understand smaller scale ecosystems. Jannasch *et al.* have developed and deployed a system of statically moored ocean sensors to monitor environmental processes off the coast of California in Monterey Bay [56]. The mooring buoys are at the surface and provide real-time data feedback via a radio link. Glenn *et al.* are improving the 20 year old LEO-15 cabled observatory located off the coast of New Jersey in 15 meters of water. The new system includes more support for integration with AUVs and other types of sensors [46]. Curtin *et al.* present the Autonomous Oceanographic Sampling Network (AOSN), which combines static sensors and AUVs [30]. One of the

goals of AOSN was to adapt the trajectories of the AUVs based on the readings from the static sensors. The Proudman Oceanographic Laboratory has deployed an ocean observatory in the Irish Sea [52]. Frye *et al.* have deployed a deep ocean observatory, located off the coast of Vancouver Island, which uses ocean floor sensors that are acoustically linked to a surface buoy. The surface buoy transmits collected data via a satellite link [42]. The Monterey Bay Aquarium Research Institute (MBARI) has designed a mooring system for deep sea deployment. This system provides power and communication links to multiple sea bottom sensors from a surface buoy [25]. Howe and McGinnis have developed a subsurface moored platform for the ALOHA Observatory north of Oahu. This platform has a variety of sensors and supports docking of an ROV/AUV and has a water-column profiler which can travel along the mooring line [53]. Our system aims to be easy to integrate with any of these ocean observatory systems, while providing new capabilities such as easy deployment and recovery, automatic self localization and tracking, multi-model communications, enhanced routing, and optimization of the acoustic communications channel through mobility.

Our platform contains acoustic, optical and radio communications systems. The Woods Hole Oceanographic Institute (WHOI) acoustic modem [41] and the Benthos modem [5] are two commercial acoustic modems that have similar performance to our acoustic modem. We built our own systems for a number of reasons. At the time, the available acoustic systems cost more than the total cost of our nodes,[1] were larger than our node, and did not give access to the low-level interfaces to customize the communication protocols. Acoustic communication is extremely challenging due to the path loss, noise, multi-path, and delay [8, 97, 77]. Most effort has gone into improving the physical and MAC layers of acoustic communication to improve throughput and efficiency. We, however, take a different approach. Instead of trying to improve the physical and MAC layers of the acoustic system, we change the placement of the sensors online to improve the throughput and efficiency.

[1] Our nodes cost approximately $1k, while commercial acoustic modems cost over $3k.

Our optical communications system is detailed in [36]. The recent development of high powered blue-green lasers and LEDs has enabled the development of underwater optical communication systems. A number of studies explore the theory of optical transmission in water and suggested possible optical modem designs [27, 93, 28, 45]. Tsuchida *et al.* report an early underwater analog communication system for wirelessly monitoring crayfish neuronal activity in [98]. Schill *et al.* create an underwater optical system by combining blue-green LEDs with an IrDA physical layer with the intent of communicating between a swarm of underwater robots [85]. Hanson and Radic proposed the use of waveguide modulated optically lasers for high speed optical comms [50]. Farr *et al.* discuss the possibility of using optical communication for control of underwater vehicles and present the results of an early prototype optical communication system [39]. High-speed underwater optical communication systems are not commercially available at this time, so we built our own system. We improve upon existing systems by increasing speed, while reducing size and are able to integrate the system inside the AquaNodes.

The focus of this thesis is not on the physical communications systems themselves but rather on the novel combination of these systems into one platform to improve overall communications efficiency and reliability.

2.2.1 Depth Adjustment

In this thesis we develop a depth adjustment system for our underwater sensor network. As far back as 1964 researchers constructed devices to sample water conditions at particular depths. Joeris devised a device which could take samples at particular depths by being lowered via a winch from a boat at the surface of the water [58]. Spiess devised a device for supporting multiple packages of oceanographic equipment at a particular depth. This device used two vertical anchor lines with a horizontal line suspended between. A buoyant device moved up and down on the vertical lines to adjust the depth of the sensor packages [94]. Springer *et al.* use a winch from a surface platform to automatically lower and raise a hydrological sensor developed by the Center for Applied Aquatic Ecology at North Carolina State University [95].

The LEO-15 platform developed jointly by WHOI and Rutgers University has a bottom mounted winch system for water column profiling [46]. Howe and McGinnis have developed a water column profiler which travels along the mooring cable of their system and is able to recharge inductively at the surface platform [53]. Bokser *et al.* developed an underwater sensor (based on the Mica2 Mote) that can adjust its depth by moving a piston to change its buoyancy [20]. Pompili *et al.* mention using a bottom-based winch or inflatable buoy to adjust the depth of sensors to more easily deploy 3D sensor networks underwater [81, 80].

Our system differs from this prior work in that we use the depth adjustment capabilities of our system for more than just water column profiling. We use depth adjustment with feedback in an autonomous cooperative system of sensor nodes that interact, sense, and decide on best node placement. Utilizing the depth adjustment system in conjunction with an underwater sensor network enables algorithms that improve sensing and communications. In addition, this system makes localization and recovery/deployment of large systems much easier than traditional underwater sensor networks.

A few papers discuss changing the depth of a moored underwater sensor to improve sensing or communication. Akyildiz *et al.* discuss the benefit and challenges associated with adjusting the depth of underwater sensor nodes [8, 9]. They note that depth adjustment allows greater sensor coverage, however, other factors need to be taken into consideration, such as maintaining network communication connectivity. Akkaya *et al.* propose adjusting the depths of nodes in an underwater sensor network to reduce overlap to improve sensor coverage [7]. These papers present some theory, but do not implement depth adjustment systems for underwater sensor networks.

2.3 Prior Work in Sensor Placement

The algorithm developed in this paper is closely related to previous work in sensor placement and robot path planning to optimize sensing. Here we summarize a few of the many related papers in this area. Cayirci *et al.* try to distribute underwater sensors to maximize coverage of a region [24]. They break the area of interest

into cubes and adjust the depths to place at most one sensor in each cube. The algorithm assumes near-ideal communication between nodes by way of surface buoys and they do not test the algorithm on a hardware platform. Akkaya *et al.* simulate optimizing sensor positioning in an underwater sensor network with depth adjusting capabilities [7]. Their approach is to evenly spread out the nodes while maintaining communication links using a decentralized graph coloring algorithm. Our approach is different in that we account for sensing covariances, use realistic communication assumptions, and implement the algorithm on a real underwater sensor network.

Pompili *et al.* present different deployment strategies and analyze the number of nodes needed to provide full sensing coverage in an underwater sensor network [80]. Alam and Haas find from a theoretic perspective that a Voronoi tessellation is the best approach for node placement in 3D to maintain communication while maximizing spacing between nodes for sensing [10].

Ko *et al.* develop an algorithm for sampling at informative locations based on minimizing the entropy [62]. Guestrin *et al.* introduce the optimization criterion called mutual information [47]. Mutual information finds sensor placements that provide the most information about unsensed locations. They prove that the problem of picking optimal sensor location is NP-complete and provide a constant factor approximation algorithm. Leonard *et al.* develop controllers to create optimal ellipsoidal trajectories for mobile underwater sensors based on minimizing the posterior error assuming a Gaussian process [67]. Yilmas *et al.* plan a path for an AUV that will decrease measurement uncertainty using a linear programming approach [108]. Rigby develops an algorithm that uses Monte-Carlo simulations to pick a path for an AUV that minimizes the trace of the posterior covariance matrix assuming a Gaussian process [84].

Singh *et al.* present an offline algorithm for planning paths of multiple robots using mutual information [92]. In general this problem is NP-hard in both the offline and online versions. Krause *et al.* develop an algorithm to simultaneously choose where to place sensors and determine their optimal power schedule [64]. They show empirically that optimizing both of these together works much better than first optimizing placement and then optimizing scheduling.

Our algorithm differs from these works on sensor placement and path planning in a number of ways. Our algorithm is decentralized and runs in real-time on our underwater sensor network, whereas much of the related work computes placements and trajectories before the deployment. One of the key components of our algorithm that allows us to run in real-time is a choice of an objective function that is easier to compute on a sensor network platform with limited memory and processing capabilities. Most current approaches are based on the minimization of entropy, posterior error, or the use of mutual information. All of these formulations require the computation of the inverse or determinate of the full covariance matrix for the system. These computations exceed the memory and processing capabilities of most sensor network systems. Our system only relies on having the covariance and thus we are able to implement it on our sensor network platform. We show in Chapter 5, that our algorithm also tends to minimize the entropy criteria, while requiring far less computational complexity. Our algorithm also runs continuously and can adapt to changing conditions. Most of the previous work precomputes positions before deployment.

Further, our algorithm uses a decentralized gradient-descent based controller. Cortes *et al.* use a decentralized gradient controller to perform Voronoi tessellations for a known event distribution [29]. Details on these types of algorithms can be found in the book by Bullo *et al.* [21]. Schwager *et al.* extends these controllers to learn the underlying sensing function through consensus [89] and to monitoring environments using downward facing cameras on quad-rotor helicopters [88]. Our work draws inspiration from these techniques, but differs in the problem specification: our underwater sensors are only able to adjust their depth and are extremely constrained by communication.

Obtaining a covariance function for an underwater system is a problem that has been previously studied. Leonard *et al.* use a multivariate Gaussian function, similar to ours, to estimate the covariance in their system [67]. Lynch *et al.* note the historic use of multivariate Gaussian functions to model underwater systems and use stochastically-forced differential equations to analytically determine better covariance models for ocean environments [69]. They use the covariance models they develop as

input into objective analysis models. Objective analysis is statistical estimation under Gauss-Markov conditions. Feng *et al.* develop a nonparametric model to estimate a covariance matrix with a particular focus on financial systems [40]. In our system we use a multivariate Gaussian function as a first estimate of the covariance for systems where detailed information is unavailable. For systems with sensed or modeled data, we numerically compute a covariance function based on the actual data. These functions can be updated online as the system runs to improve their accuracy.

2.4 Prior Work in Underwater Localization

Navigation and position estimation for traditional AUVs is typically achieved with a combination of dead-reckoning systems and external localization systems. The dead-reckoning systems are typically composed of a Doppler velocity logger (for dead reckoning) and laser ring gyros (for precise orientation). These sensors are far too large and expensive for the scale of our robot, AMOUR.[2] For dead-reckoning AMOUR has an inexpensive inertial measurement unit (IMU) that gives good orientation information but poor information on translational motion. Instead, our system relies on an external localization system provided by a network of AQUANODES. The theory behind this algorithm was developed in [31]. In this thesis, we consider how to move the theory to a localization system that can be implemented and used underwater in the presence of limited bandwidth and extensive uncertainty.

The most common method for external localization underwater is to deploy a long base line (LBL) systems. This consists of a small number of static buoys deployed manually in the water. These buoys typically contain sonar pingers. When they receive a ping at a certain frequency they respond with a ping. Ranges can be obtained by measuring the round trip time of the ping. The response pings of the different buoys are on different frequencies so that simultaneous range measurements can be obtained, making localization easier.

LBL systems differ from our underwater sensor network in a number of ways. The LBL buoys typically do not have any other means of communication with the mobile

[2] Our robot costs under $10k, while inexpensive versions of these sensors cost $50k.

node that is being localized. Because of this and since all of the buoys respond to pings, the number of buoys that can be deployed in a single area is limited to the bandwidth of the acoustic communications channel. Typically at most four buoys are deployed. This limits the effective area of localization to the range of an individual LBL buoy (typically a few kilometers). This small area may be fine for most current deployments, but it is not sufficient to provide a GPS-type system over large areas of the ocean.

The algorithms used for localization underwater generally fall in one of three categories: (1) geometric methods, (2) Markov or Monte Carlo methods, and (3) Kalman filters. Our systems is most related to geometric methods, although the on-robot implementation also uses a Markovian approach. Existing localization techniques and the unique challenges of the underwater environment, such as the difficulties in obtaining ranges due to the variable speed of sound in water and multipath effects, are surveyed by Chandrasekhar *et al.* [26].

Hahn presents a straight forward geometric least-squares approach that takes ranges received from multiple static nodes to compute a position for an AUV [49]. The method does not handle ranges that are received over time and does not propagate information back to refine previous location estimates. Olson *et al.* use a geometrically inspired approach and dead-reckoning information to successfully filter out measurement noise before using an extended Kalman filter (EKF) to solve the simultaneous localization and mapping (SLAM) problem [75]. Our solution does not rely on dead-reckoning information and only solves for robot location. Galstyan *et al.* use geometric constraints to localize a network of static nodes by moving a mobile node through the network [43]. This problem is in some senses an inverse problem from ours as we are trying to locate the mobile node instead of the network. Newman and Leonard use a non-linear optimization algorithm to solve for the vehicle trajectory as well as the LBL node locations [74]. Their system assumes that the vehicle is traveling in a straight line at a constant velocity with Gaussian noise in the acceleration.

Kantor and Singh present the theory behind Markov and Monte Carlo methods for localizing moving robots based on range information [61]. Their Markov, grid-based approach is similar to our implementation on AMOUR, but, they do not propagate information back to refine previous location estimates. Gutmann observes that Markov approaches often yield better results than EKF approaches [48]. The main problem is that once an EKF gets into a bad state, it tends to stay there. Gutmann combines a Markov approach and an EKF to obtain better localization results for localizing a RoboCup robot than either approach alone. He uses dead-reckoning, range, and angle information to update the Markov grid.

Another approach to localization is to directly use a Kalman filter. One of the challenges with Kalman filters is in the initialization of the filter. The filter might diverge if a good estimate of the location of the mobile node is not known at the start. It is also possible that measurements that fall outside the measurement model will cause the filter to diverge and end up in a bad state. Kurth performs experiments with different manually specified initial locations and concludes that poorly initialized filters never recover [65]. Vaganay *et al.* initialize their location in their range-only Kalman filter by fitting a line to the first N trilaterated locations [99]. They assume that they are able to obtain synchronous range measurements and that their initial motion is a straight line. Djugash *et al.* initialize their filter using a GPS reading [33]. Our algorithm does not require explicit initialization and gracefully recovers if the algorithm diverges due to a large set of invalid measurements.

In addition to localizing single vehicles moving through a network, there has been research on localizing groups or swarms of robots. Kottege and Zimmer [63] present a localization system for swarms of robots underwater. In their system each robot is equipped with a number of acoustic receivers. Using the acoustic receivers they are able to obtain range and angle between pairs of robots. This single reading gives them a relative location estimate between the pair which is sufficient for the swarm behavior in which they are interested. However, they do not discuss how this could be used to create a global coordinate system for the robots.

Bahr and Leonard [13] and Vaganay *et al.* [100] look at a cooperative localization system between a team vehicles with different capabilities (e.g. some with expensive navigation equipment and others without). The vehicles with poor localization capabilities obtain ranges and location information from vehicles with good localization information. Vaganay *et al.* show that this type of system maintains accurate relative localization between the vehicles even after the absolute localization starts to drift due to inaccuracies in the more capable navigation systems [100]. Bahr and Leonard focus on minimizing the amount of communication that is needed while maximizing the information gain [13]. This is important in underwater systems where communication bandwidth is limited.

The following chapters present our system and algorithms in detail. They also discuss how they fill in the gaps present in current systems.

Chapter 3

Hardware and Software System

In this chapter we examine the details of the hardware and software that we developed to enable the implementation of our algorithm and the experiments described in this thesis. The core AquaNode hardware and software was developed primarily for this thesis. The acoustic and optical communication systems, as well as the underwater robot, Amour, were developed jointly. We start by examining the hardware and software design of our underwater sensor network nodes which we call AquaNodes. We then present the optical, acoustic and radio communication systems. Next, we briefly describe the underwater robot, Amour. Finally, we present the results of analysis and experiments that characterize the components of the system.

3.1 AquaNode Design and Implementation

Figure 3-1 shows a picture of the AquaNode. We developed the hardware, electronics and the software for this system in our lab. The AquaNodes are designed to be a flexible underwater sensing and communication system. Each AquaNode has a pressure sensor (for depth) and temperature sensor. It also has an underwater connector which allows for the connection of up to 4 other external analog or digital sensors. Communication is possible through acoustic, optical and radio systems (bluetooth and 900MHz radio).

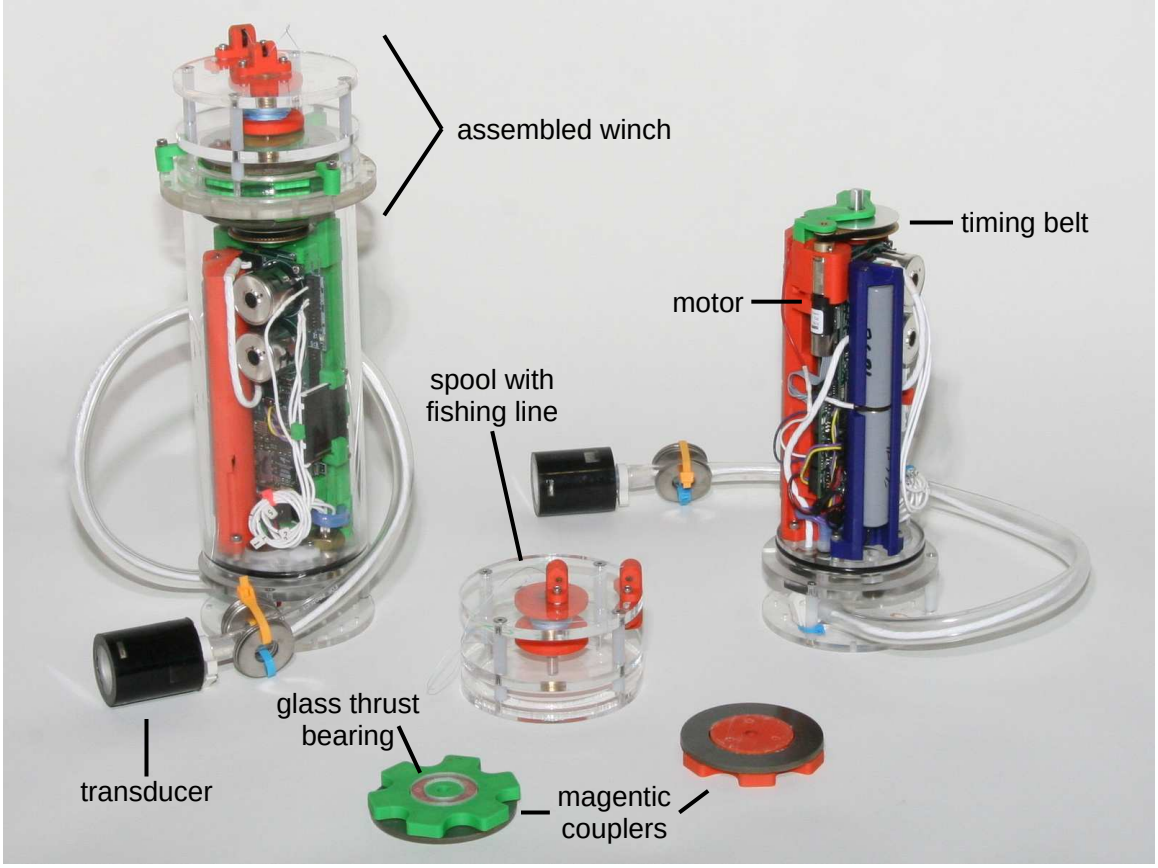


Figure 3-1: Picture of the depth adjustment hardware on a sensor node and the individual components.

Typically, AquaNodes are moored to an anchor and float in the water mid-column. With the addition of the novel winch-based depth adjustment system, described in Section 3.1.3, the AquaNodes are able to dynamically adjust their depth. This opens a variety of new capabilities, including the decentralized depth adjustment algorithm for improved sensing described in Chapter 4.

Figure 3-2 shows a block diagram of the AquaNode. The AquaNode consists of three main circuit boards. At the core of the AquaNode is the base board, which is a generic sensor board platform we developed in our lab for use in multiple projects. Attached to the base board is the midlayer board which contains power management as well as sensors specific to the underwater system. Additionally, the underwater sensor nodes contain radio, acoustic, and optical modems. The radio does not work underwater, but it can be used at the surface. The acoustic modem is a long-range,

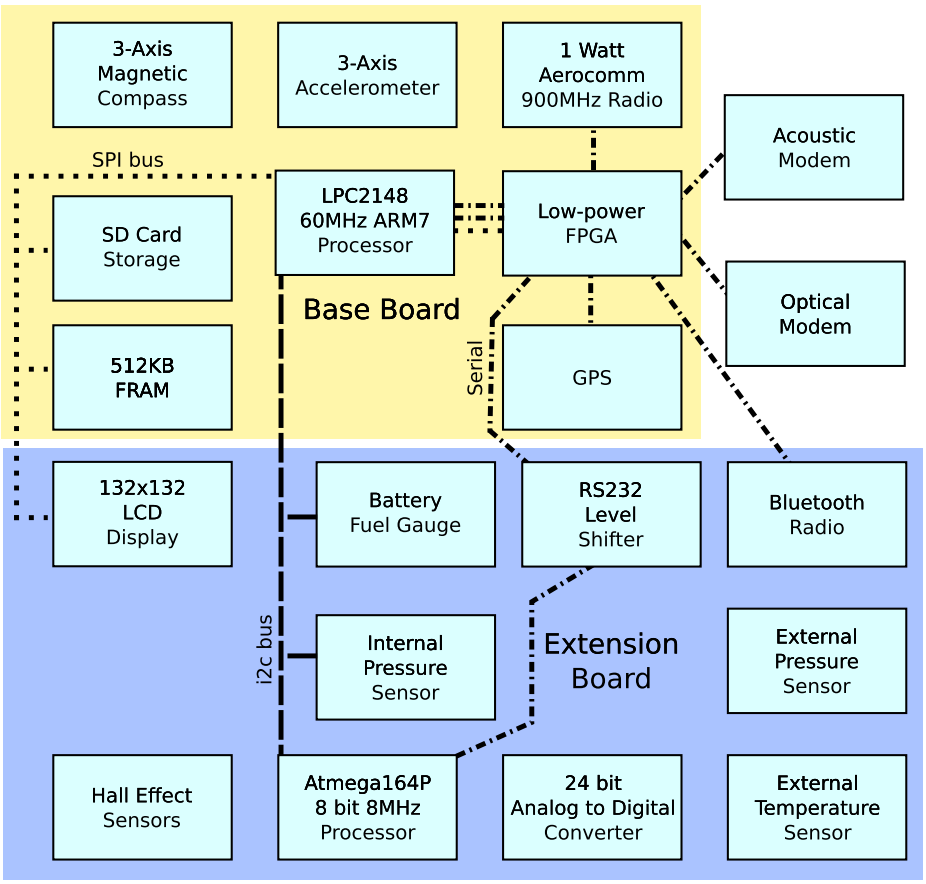


Figure 3-2: Block diagram of the AquaNode

low-datarate system used for underwater communication, and the optical modem is a high-speed, short-range, point-to-point communication system used for downloading large amounts of data. The rest of this section describes the individual components of the AquaNode in more detail.

3.1.1 Base Board

The base board is the core of the AquaNode. This runs a custom built operating system designed to be flexible and powerful for a variety of sensing-based tasks. We use the base board in a number of different projects beyond the underwater sensor network discussed in this thesis. These projects include virtual fencing for cows and other range animals [35, 87], river level monitoring for flood detection [14], adaptive illumination for color correct underwater photography [104], and a variety of other smaller projects.

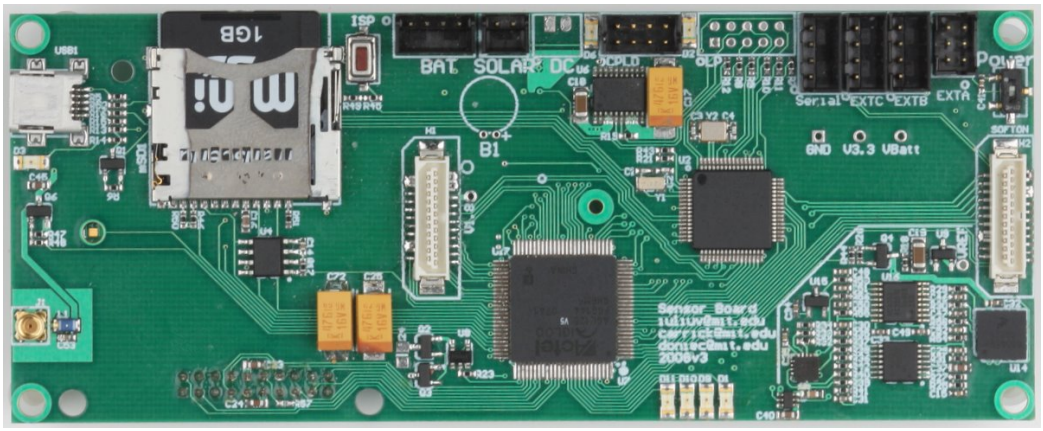


Figure 3-3: Picture of the base board.

Hardware

Figure 3-3 shows a picture of the base board and Figure 3-2 shows a block diagram of the hardware architecture. The operating system runs on an NXP LPC2148 ARM7TDMI processor clocked at up to 60MHz. This processor has 40kB of ram and 512kB of on-chip flash. It has two UART serial ports, two SPI ports, two I^2C ports, real-time clock, 10-bit analog to digital converter ports, and general purpose I/O pins.

For sensing we have a 3-axis accelerometer and a 3-axis magnetic compass, which we use to determine the 3D pose of the board. We also have a GPS on the board to determine position when GPS is available.

For data storage we have a 512kB FRAM, which is a high speed non-volatile RAM with no limit on the number of writes. Additionally, we have a miniSD card slot which supports SD cards of size up to 2GB with a custom-implemented Fat16 filesystem. The filesystem has a logging interface that makes logging sensor or other values extremely easy. The user just has to open a named log file and then uses fprintf-like syntax to write to the log file. Log files are stored in directories named by the date and automatically rotated every day so that all log files from a particular day are stored in the same location.

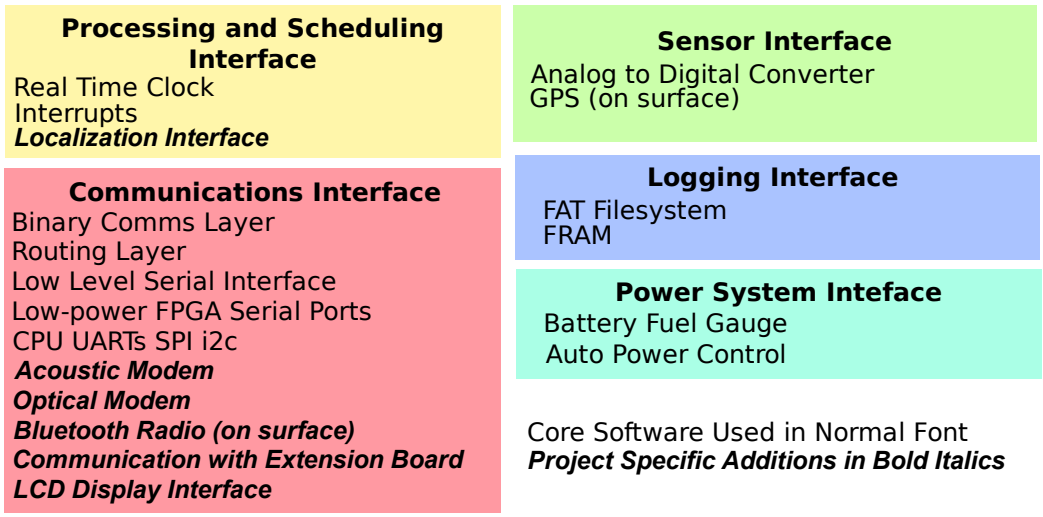


Figure 3-4: The core software of our operating systems and extensions added for the AquaNodes.

For communication we have a 1 watt Aerocomm AC4790 900 MHz radio [2]. We also have a small, low-power FPGA on the base board that primarily acts as a serial buffer and multiplexer that supplies 4 or more additional serial devices (the ARM processor only has 2 serial ports). The board also has a USB slave port, allowing direct USB connections to computers.

Additionally, the board has expansion connectors that allows the connection of project-specific expansion boards. For the AquaNodes we have attached the mid-layer board described in Section 3.1.2.

Software

The operating system on the base board is a cooperative multi-tasking schedule based system. We chose to develop our own operating systems, as opposed to using existing systems such as TinyOS, so that we could optimize the OS for our system while maintaining flexibility. TinyOS was designed to run on systems with very limited memory and processing capabilities. Our system has significantly more memory and processing ability and our operating system is designed to take advantage of these.

Figure 3-4 shows the various components of the operating system and extensions added onto the core system for use by the AquaNodes. Over 80% of all of the

software used by the AquaNodes is core code from the base board operating system. This section describes the details of the base board core software system.

Scheduler

The base board runs a scheduler based operating system. Processes are scheduled to run at various intervals with millisecond granularity. To schedule a process the function `id = scheduleAdd(func,name,interval)` is called. `func` is a function pointer to the function that will be called at approximately `interval` milliseconds. The string `name` is used by the system to display information about the process and is also used to manipulate the process. Processes can also be manipulated by referencing the `id` returned, or by using the function pointer `func`. For instance, events can be deleted by calling `scheduleDeleteFunc(func)`, `scheduleDeleteId(id)`, or `scheduleDeleteStr(name)`. There are similar functions that allow the interval at which a function runs to be updated.

In addition to running events repeatedly, it is possible to schedule events to run a single time with the function `id = scheduleRunOnce(func,name,when)`, where `when` indicates the number of milliseconds from now at which to run the event. It is also possible for an event to add or delete other events, as well as delete itself. This allows maximum flexibility in the configuration and running of events.

The scheduler itself is a fairly basic system that runs all the events that should run at the time in sequence. The events are not preempted so they must yield control within a reasonable amount of time. We decided to use this approach as we have a limited amount of RAM available, which would limit the number of concurrent processes that could run in a preemptive system. To prevent one long-running event from keeping other events from running we define a window over which events can run. If one event runs longer than it should, preventing other events from running at their designated time, the other events will be run first, before the long event is run again. This prevents complete starvation of other events.

Communication

The operating system has a message routing system that allows messages to be routed transparently between boards over the radio or other links. This allows users to

interact with (and reprogram) all of the boards while only being physically connected to a single board. The routing system is application-customizable so that manual or dynamic routing protocols can easily be developed.

At a low level, the operating system supports communication over a variety of interfaces including i2c, spi, uart, and RS485. Abstraction is provided for each of these to enable quick addition or reconfiguration of devices. For instance, a byte is sent over a uart by calling `success = uartSendByte(port,byte)`. The `port` specifies the port to send over and a value is returned to indicate if it was successfully sent or queued to send. The command `countSent = uartSendBytes(port,bytes,count)` is used to send multiple bytes. These commands abstract away the lower level queueing and routing required to send bytes. For instance, only 2 of the uarts are physical ports on the processor. The 4 other ports are located on the fpga and are accessed using spi. Similar command allow receiving bytes and using the other types of interfaces.

On top of the low-level communication system is a binary communication system. The binary communication works on any of the ports on any of the interfaces. The packet structure supports a routing layer that enables routing of messages between nodes and different ports. Each packet contains a header with a start byte, hop count, message id, source node id, and destination node id. This is followed by the data and a checksum.

The hop count and source and destination node ids are used by the routing layer to determine what to do when a packet is received. If the destination is the current node id or if it is a broadcast id the packet is passed to a handler for the specific message id. Packet handlers are added dynamically by calling the command `binaryCommAddHandler(messageId,function)`. This specifies that `function` should be called for every packet destined to this node with the specified `messageId`.

If a packet is destined for another node or broadcast then a routing table is accessed to determine if this message should be forwarded. The interface for the routing code is generic and any routing algorithm can be implemented on top of it. At a low level, routes are added by calling `routeAdd(destNodeId,rxPort,txPort)`. This indicates that any message received on the port `rxPort` with a destination of

`destNodeId` should be forwarded over the port `txPort`. A `rxPort` of -1 is specified to indicate how messages originating at this node should be routed. Addresses larger than 0xF0 are reserved for special uses such as indicating a broadcast message or an extended addressing range (to allow sending to more than 255 unique addresses). The `rxPort` and `txPort` can also be the same (for instance to act as a radio repeater).

Ports that are used for receiving and sending messages are easily added by defining a set of functions. The required functions are:

- `getNumBytesAvailable()`: Gets the number of bytes available on this port.
- `peekByte(n)`: Peeks at the n^{th} byte, but does not remove it.
- `readBytes(dest,count)`: Reads count bytes into the buffer dest and removes them from the port buffer.
- `readNextByte()`: Reads the next byte from the port and removes it from the buffer.
- `sendBytes(bytes,count)`: Sends count bytes over the port.
- `setDestination(port)`: Sets the destination node of the packet, useful for addressed systems such as some radios, spi, or i2c.

These are the only functions that need to be defined to use a port as a binary communications port. Most of these functions mirror low-level functions. However, defining them in this generic format allows the use of more complicated communication protocols with the routing and binary communication system. For instance, it is possible to add a port that uses spi to communicate. Calling the function `binaryCommAddPort(portId,getNumBytesAvailable...)` creates a new binary communications port defined by the above functions.

The communication system provides a number of different layers of abstraction to aid in development and deployment. The higher layers enable transparent routing of messages to nodes. At a low level the abstraction enables the addition and reconfiguration of communication systems and ports without having to change higher-level code.

Logging

The platform contains a number of systems to enable storing and logging of data. At a high level there is a logging interface that allows logging of data to named files using a printf-like syntax. This is implemented on top of a FAT16 filesystem on the SD card that allows the log files to be easily read using a SD card reader on a computer. In addition, data can be stored on the non-volatile FRAM. This is typically used for storing configuration data.

We implemented a FAT16 filesystem for the SD card slot on the sensor nodes. FAT16 allows access to SD cards of up to 2 gigabytes. Using FAT16 has the advantage that it is easily readable by most all computers. The filesystem drivers use a layered approach with separate layers and interfaces for the low-level hardware, drive, partitions, and filesystem. This enables future expansion of the filesystem driver to support other filesystems such as FAT32 (up to 32G), exFAT (up to 2TB), or other filesystems that are specifically designed for mobile devices. Filesystem-independent interfaces allow files and directories to be created, deleted, read, written, and moved.

A logging interface is available on top of the filesystem interface. The logging interface enables quick and easy creation of log files. The command `id = logNewLog(name)` creates a new log file with the specified `name`. The `id` returned is a handle to reference this log file when writing to the file using the command `logWrite(id,str,params,...)`. The string, `str`, and parameters, `params` follow the printf-type syntax. This allows logging of data in any format. By convention, the log files are comma delimited and any lines starting with `#` are considered comments. Using the function `logWriteBytes(id,bytes,count)` enables directly writing `count` raw `bytes` to the log file. This is useful, for instance, when directly logging the data received from a peripheral device as the bytes received can be directly logged.

The log files are created in the directory named by the year, month, day with the format: `/YYYY/MM/DD/`. The log files are rotated on a daily basis. This ensures that the files are a manageable size, contain some minimum date information (if none is stored in the file), and are easily located. The time of the day the rotation occurs can be set so that the rotation occurs at a convenient time. For instance, if the

experiment is measuring air temperature at night, an offset of 8 hours can be set so the log files will rotate in the morning. This will ensure that the periods of interest are all located in one contiguous log file. This is merely for convenience as the logging systems transparently rotates the files without interruption.

Power Control

The base sensor network system provides a number of high and low-level power control features. Each device can be turned on or off by calling a device specific function. For instance, the GPS can be turned on or off by calling `gpsPower(on)`. This allows the applications to reduce power usage by disabling peripherals.

Another method to conserve power in the system is to change the clock speed of the processor. The clock speed is set by calling `pllSet(clockSpeedHz)`. This function sets the clock speed of the processor to the specified clock speed if it is available. The current implementation allows clock speed settings between 12MHz and 60MHz in 12MHz steps. The processor also supports running at slower clock speeds for further power savings if desired.

Finally, the processor can be placed in a very low power sleep mode by calling `powerDown(time)`. The `time` specifies the maximum number of seconds the system should sleep. It will automatically wakeup after this time expires or if an external event triggers a wakeup (external wakeups are configurable).

In addition to manually powering down the processor, the power management system is also integrated with the event scheduler. Each event has a flag that can be set to specify if the event should wakeup the processor. The scheduler computes the time until the next event needs to run. If the period until this event is long enough, the processor will power down if it has been configured to do so by calling `powerFlagChange(shouldAutoPowerDown)`. The automatic power down can be configured using the functions:

- `powerSetMinSleepTime(sec)`: Sets the minimum number of seconds to sleep for. There must be at least this many seconds before the next event is scheduled to run for the system to sleep.
- `powerSetWakeupPost(sec)`: Sets the number of seconds after the last required event is run before sleeping. This allows other non-required events to run.

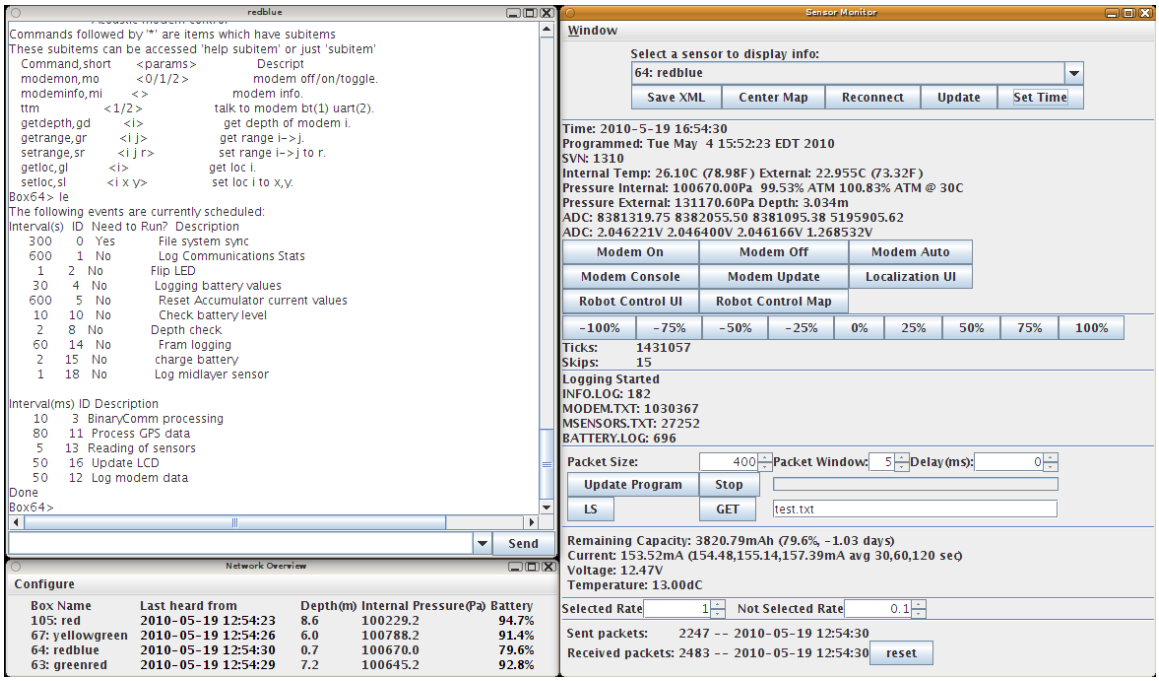


Figure 3-5: The Java user interface.

- `powerSetWakeupPre(sec)`: Sets the number of seconds to wakeup before the next required event is run. This allows initialization time and time for other non-required events to run.

User Interface

The system provides two different user interfaces. The first is a console-like test interface. The second is a Java graphical user interface. These interfaces allow users to interact with the board, change scheduled events, configure parameters, and re-program the board.

The Java user interface is shown in Figure 3-5, with the console interface in the left portion of the figure. The user is able to configure most aspects of the system using this interface. For instance running events are listed using the command `listevents` or `le`. Events are added using the command `addevent id` or `ae id` where `id` is a numeric id for the desired event. The `id` is determined by calling `addevent` or `ae` with no arguments.

Console commands are added in the code by calling `consoleAddCommand(function,name,shortname,paramDescript,helpMsg)`. The `function` is a pointer to the function that should be called when the console

command is executed by the user typing the `name` or `shortname` into the console. The `paramDescript` provides a description of any parameters and `helpMsg` is a short help message given when the user types `help`. Commands can also be grouped together for ease of use. For instance, all commands related to the acoustic are grouped together and can be accessed by typing `modem`. The console is accessed by directly connecting to a sensor board over a serial port, bluetooth, or radio. Alternatively, it can be accessed via the Java user interface.

The Java user interface (UI) also provides the same console interface and in addition it has an easily configurable graphical interface (shown in Figure 3-5). The user can connect to the UI using a direct serial connection, bluetooth, radio, or any other binary communication port. In addition, as the messaging between the sensor node and UI uses the binary communication packets, the UI can connect to many sensor nodes, perhaps over multiple communication hops. This allows the UI to control multiple sensor nodes over a variety of communication links. For instance, the UI can connect to a local sensor node over RS232, a nearby node using bluetooth, and many remote nodes over radio using the bluetooth node as a gateway. These are all connected at the same time. The configuration of how to connect to nodes in the UI is setup in an XML file.

The UI has multiple panels that enable easy customization based on the needed functionality and the project. For instance, there is a debug panel that collects communication statistics. This is not typically needed, so it can be added or removed from the UI by (un)checking a checkbox in the configuration window. This allows the same UI to be used for multiple projects. The default layout for a particular node is configured using an XML file. This dynamic configuration enables mixed use of the UI. For instance, a network containing land-based and underwater-based sensors can all be controlled by the same UI, with the UI showing the proper controls for the different types of nodes.

Reprogramming

The sensor network system is easily reprogrammed using the Java user interface or by updating a file on the SD card. The sensor nodes have a bootloader that runs

each time the board boots. The bootloader looks on the SD card for two files. The first is `main.bin` and the second is `main.crc`. The first file contains an image of the code that should be loaded onto the board. The crc files contains a 32 bit crc of the bin file. This is used to verify the integrity of the bin file. Each time the board boots the code in the bin file is compared to the currently loaded program. This takes under five seconds. If they differ the new program is loaded, otherwise the main operating system is booted.

A safety system is built in to prevent the bricking[1] of a sensor node in an inaccessible location. If the board quickly reboots more than 5 times in a row, the bootloader will load the file `mainbk.crc`. This allows confident reprogramming of the board even when it is inaccessible.

The file on the SD card can be updated by manually placing the file on the SD card. Alternatively, the Java user interface enables the user to send a new image file to the node. It is possible to do this with a direct connection or over other communication links such as the radio. Directly connected, it takes under two minutes to reprogram the node and only requires pressing one button in the UI. Over radio it can take over 15 minutes depending on the link quality. The UI allows multiple boards to be reprogrammed at the same time.

The UI also enables reprogramming of other peripheral devices. For instance, the acoustic modem can be reprogrammed in a similar manner by sending the program file to the acoustic modem from the user interface.

3.1.2 Midlayer Sensor and Power Management Board

The midlayer board is attached to the extension port of the base board. Figure 3-6 shows a picture of the midlayer board. The midlayer board has its own low power 8-bit processor, Atmel AtMega164P, which runs at 8MHz. This processor has 1kB of RAM, 16kB of program flash, and 512B of EEPROM.

[1]Bricking is a term to describe a sensor node that has been improperly programmed, resulting in a node that cannot be accessed or reprogrammed.

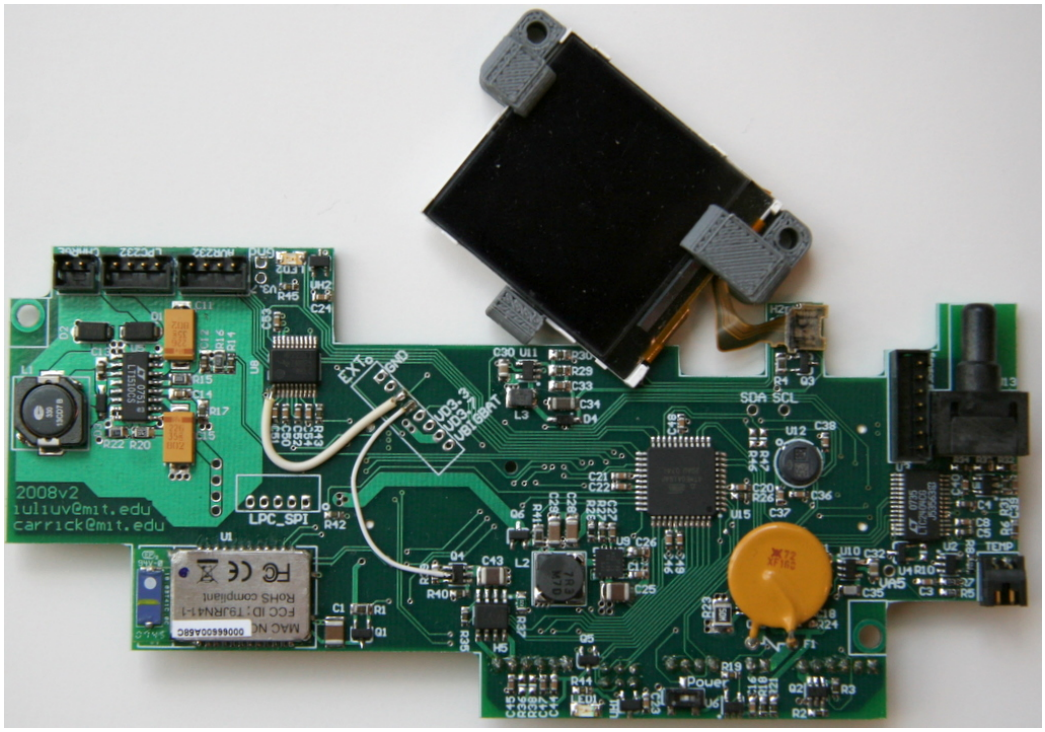


Figure 3-6: Picture of the midlayer board and the display.

The midlayer also has a battery monitoring and charging circuit, RS232 serial level shifter, bluetooth module, 24-bit analog to digital converter, motor controller and encoder, internal and external pressure sensor, and a 132x132 color LCD display. Additionally, it has a 1.5A H-bridge with software quadrature encoder for motor control. Some of these devices are directly connected to the Atmel processor, while others are connected to the base board. The midlayer and base board communicate via the I^2C bus.

The midlayer board controls the power to the base board. This allows it to completely shut down the base board when full sensing and computation power are not needed. The AquaNodes have lithium-ion batteries which have 60Whr of energy. In its lowest power mode (about 8mW) this is sufficient for about a year of standby time. In full sensing mode the AquaNode uses about 150mW which allows for two weeks of full sensing (reading recorded at least once a second). With frequent acoustic communication it uses about 1W allowing for an operation time of over two days. By varying the duty cycle of the sensing and the acoustic communication the desired deployment time can be achieved.

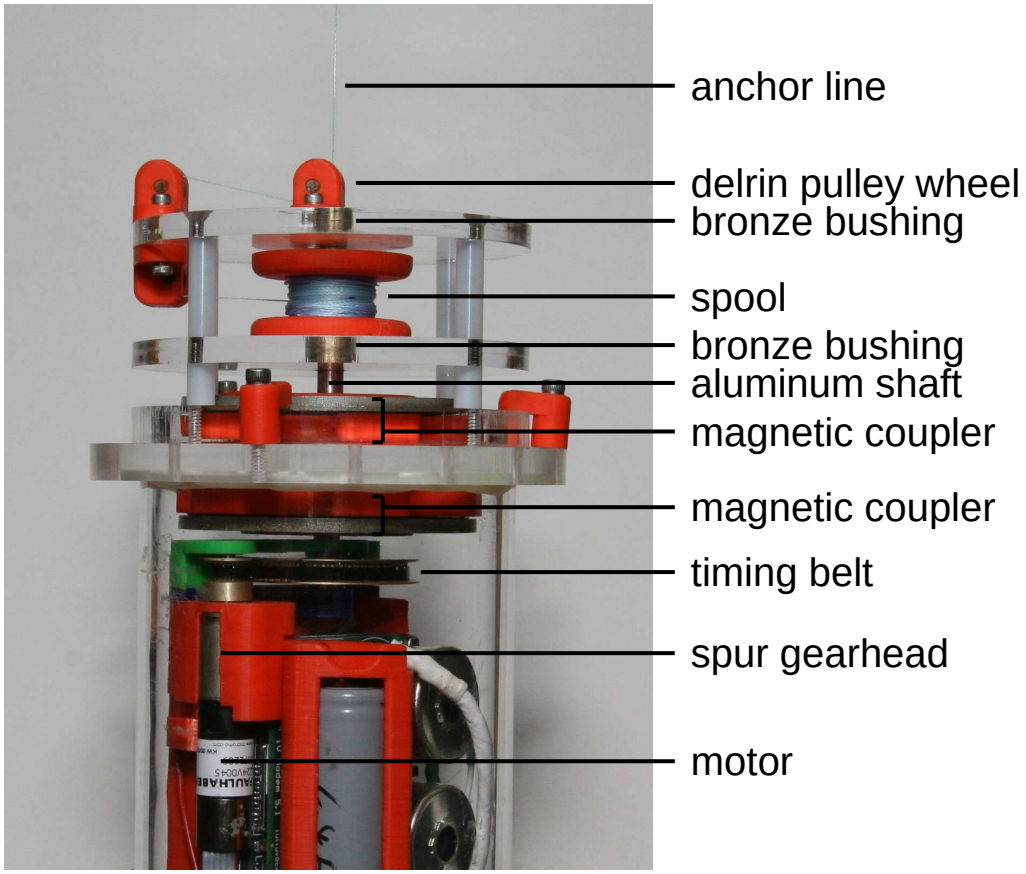


Figure 3-7: Details of the depth adjustment system.

3.1.3 Depth Adjustment System Design

The AquaNode is anchored and the depth adjustment system allows the length of anchor line to be altered to adjust the depth in the water. The AquaNode is cylindrically shaped with a diameter of 8.9cm and a length of 25.4cm without the depth adjustment mechanism and 30.5cm with the system attached. It weighs 1.8kg and is 200g buoyant with the depth adjustment system attached. Typically, the node is anchored using a 2 to 5kg weight, depending on water conditions. The depth adjustment system allows the AquaNodes to adjust their depth in water at a speed of up to 0.5m/s and use approximately 1.6W when in motion.

The depth adjustment system is a winch-based module that can be added to the core AquaNode system to enable depth adjustment in water of up to 50m deep. Figure 3-1 shows the depth adjustment system attached to an AquaNode as well as

a disassembled node. The winch-based depth adjustment system is normally oriented down in the water. In the image the node is inverted for visibility.

Figure 3-7 depicts the details of the depth adjustment system. Internal to the node is a motor, gearhead, and timing belt drive. A magnetic coupler transmits the drive power through the AquaNode case to turn a spool of anchor line. The motor is tightly packed inside the sensor node housing alongside the batteries.

The depth adjustment system allows the AquaNodes to change their depth in water with a speed of 2.4m/min and uses approximately 0.6W when in motion. The winch is driven by a 1.5A motor controller with a software quadrature decoder running on the AquaNode processor. This connects to the motor that drives the winch. The motor is a 1.3W output power 1224-12V Faulhaber with a spur gearhead having a 20.6 to 1 reduction. The motor and gearbox assembly is 51.6mm long and 12mm wide. We connect the gearbox output to a timing belt drive that further reduces the output by 6 to 1, providing a total reduction of 123.6 to 1.

The timing belt drive connects to a custom designed magnetic coupler. The magnetic coupler transmits drive power from the inside of the housing to the outside without needing to penetrate the housing with a shaft. This has a number of advantages. First, there is no chance of leaking. Second, this allows the external components of the winch to be easily removed. Finally, the magnetic coupler is compliant to misalignments of the two sides of the coupler.

The internal and external magnetic couplers are identical and consist of four parts. We designed a holder that contains places for six magnets. We orient the magnets in the holder with poles alternating so that the magnetic field forms a closed loop when connected to the other coupler. In order to concentrate the magnetic field, a steel ring sits on top of the magnets. On the bottom of the holder we place a custom built glass thrust bearing. This gives the couplers very low-friction, ensuring efficiency.

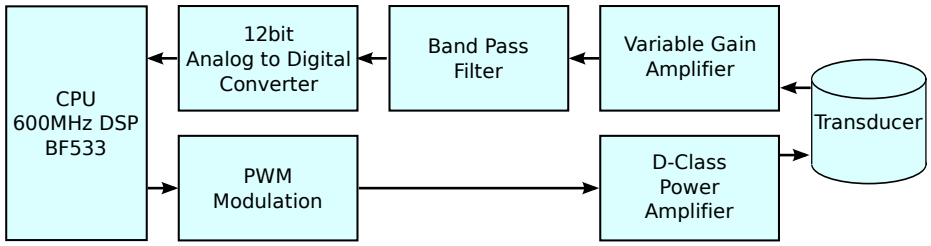


Figure 3-8: Block diagram of the acoustic modem

The external magnetic coupler is designed to be submersed in salt water, so resistance to corrosion is important. Both couplers use corrosion-resistant nickel plated neodymium magnets and we also coat the steel rings.

The external magnetic coupler attaches directly to the spool on which the anchor line is wound via an aluminum shaft. Bronze bushings support the shaft in order to allow it to spin with low-friction. Since the anchor line winds perpendicular to the shaft, three delrin pulley wheels guide and redirect the anchor line. These provide a low-friction method for properly aligning the anchor line on the spool. We use 30lb test fishing line as the anchor line on the spool. The spool holds over 50 meters of line.

3.2 Communication Systems

In this section we detail the acoustic, radio, and optical modems used in the underwater sensor nodes and robot.

3.2.1 Acoustic Modem

Figure 3-8 shows a block diagram of the acoustic modem and Figure 3-9 shows a picture of the hardware. The hardware and signal processing in the acoustic modem used in the AquaNodes was developed in our lab primarily by Iuliu Vasilescu [101]. Higher layer interfaces and protocols were developed for this thesis. We choose to develop an acoustic modem due to the high cost and lack of flexibility of existing systems. Commercial modems cost over $3k, whereas the total cost of our node is

Figure 3-9: Picture of the acoustic modem

$1k. In addition, other modems are bulky and would have increased the overall size of the AquaNodes. Finally, at the time we designed the acoustic modem commercial modems did not provide access to low level protocols we needed enable efficient multi-node communication and ranging capabilities.

At the core is an Analog Devices Blackfin BF533 fixed point DSP processor running at 600MHz. The processor generates a pulse-width modulated (PWM) signal that goes through a 10W class D power amplifier operating with nearly 90% efficiency. A PWM signal is used to allow different power output signals. The power amplifier drives a piezo-ceramic cylindrical transducer that we developed. We measured a maximum range of about 400 meters and typical working range on the order of 100 meters.

Signals are received through the same transducer used for sending. The signal passes through a variable gain amplifier and is then band pass filtered. The signal is finally digitized using a 12bit analog to digital converter.

The acoustic modem uses a frequency-shift keying (FSK) modulation with a 30KHz carrier frequency. This means that a zero bit is, for instance, represented as a pulse of frequency of 27KHz, while a one bit is represented by a frequency of 33KHz. The pulse length for each bit in the transmission is 1 millisecond followed by a 2ms pause. The total 3ms time per bit is known as the symbol size. The large pause is needed to reduce inter-symbol interference. This gives us a physical layer transmission speed of approximately 300 bits per second.

The MAC layer is a self-synchronizing time division multiple access (TDMA) scheme.[2] Each acoustic modem is given a time slot during which it is the master and controls communication during that slot. The slots are each 4 seconds long split into a master and slave portion. During the master portion of the slot the master node can transmit a 16 byte packet. A slave, specified by the master during its transmission, can respond during the second half of the slot with a 16 byte packet.

Typically, if there are N acoustic modems, there are N communication slots. By default node i will own slot i. This means that each node will be master every $4N$ seconds. For typical deployments of 10 nodes each sensor will be master every 40 seconds. The acoustic modems also have the ability to give and take slots remotely over the acoustic channel. Thus, a single node may have zero, one or many slots.

The time slots are synchronized automatically without the need for a global clock. When a modem hears another modem it automatically adjusts its clock to be in the slot of the transmitting modem. Sufficient guard times are present in the slots such that time of flight of the signal can be neglected in this slot synchronization. We have tested the automatic slot synchronization algorithm extensively in our field experiments.

We use a standardized 16 byte packet for transmission. This includes an 11 byte payload, 2 byte CRC, 1 byte source ID, 1 byte destination ID, and 1 byte for the slot number. Additionally, at the start of each packet is a 2.2ms linear frequency sweep that is used by receivers to detect and synchronize on the beginning of each packet. The rest of the time in each transmission slot is used for guard times to account for the propagation time of the acoustic waves.

3.2.2 Radio Modem

The sensor node also contains an off the shelf Aerocomm AC4790 radio. This radio has 1W transmit power and operates on the 900MHz band. The physical layer baud rate is 57600b/s. The claimed range of the radio is over 32km, however, in our experiments

[2]TDMA is inefficient and generally not used in air-based communication systems. However, underwater, large propagation delays make carrier-sense based systems perform poorly. Thus, TDMA is the common approach to underwater acoustic MAC design. See the survey by Partan *et al.*for details [77].

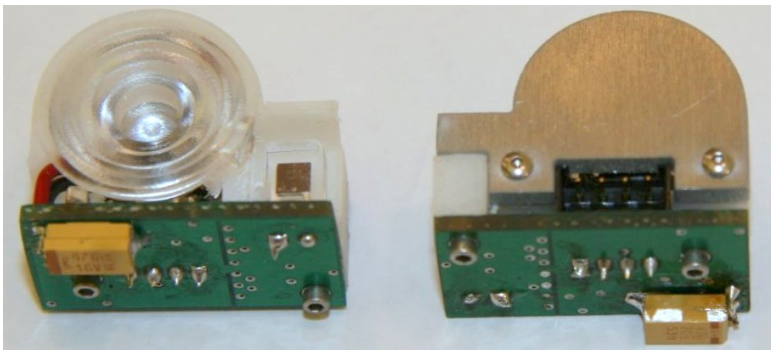


Figure 3-10: Picture of the optical board.

the radio achieved at most 3km and typical operation range was 100m. The radio has its own internal packet structure. Messages can be broadcast or directed to specific receivers. In the directed mode, the radio will retransmit up to a fixed number of times until it receives an ack for that packet from the receiver.

In our system, we typically use broadcast mode so other nodes can listen in on the packets others are transmitting. In this mode, the radio transmits each packet multiple times (set to 4 in our case). The radio simulates a full-duplex link by using fixed-length time slots for transmission. An individual radio transmits in at most half of the time slots. This reserves time for responses.

3.2.3 Optical Modem

The optical modem was designed in our lab primarily by Marek Doniec and Iuliu Vasilescu [36]. Figure 3-10 shows a picture of the optical modem. The optical modem is connected to the FPGA on the midlayer board and is powered directly from the batteries. The FPGA is used as it runs at a frequency of 32MHz and can do the timing of the LED modulation at that speed.

The optical modem is a point-to-point protocol. It has a range of up to about 3 meters in a 90° cone. It operates at variable speeds up to 4 megabits per second. It uses a pulse-position modulation (PPM) scheme, which is typical for optical communications systems. The optical modem uses a green (532nm wavelength) LED that is not absorbed as much as other wavelengths underwater.

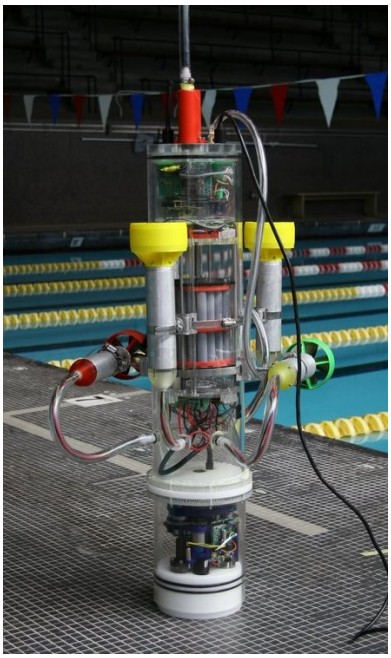


Figure 3-11: Picture of AMOUR by the pool

The optical modem has a serial port-type interface on the base board. This allows transparent use of the optical modem, it looks just like the bluetooth and physical serial connections to applications on the base board. This means it is easy to use our preexisting code for transferring files and reprogramming the base board over the optical modem.

3.3 Robot: AMOUR

Our underwater robot is called AMOUR, which stands for Autonomous Modular Optical Underwater Robot. We developed AMOUR in our lab. Typically, an AQUANODE is attached to the robot to add communication, sensing, and navigation capabilities to the base robotic platform. Figure 3-11 shows a picture of the robot. The robot is controlled by radio when it is at the surface and a tether or acoustic communication when underwater. In addition, the robot can be programmed to perform autonomous missions.

AMOUR has 5 thrusters which gives it enough degrees of freedom to operate horizontally or vertically. It has a mass of 25.5kg and a length of 0.86 meters. It has 750Wh of lithium-ion batteries. The horizontal operation is more streamlined allowing for more efficient long range travel. The vertical orientation is more maneuverable and is used to stay over a node for optical communication or docking for retrieval.

In order to pick up, carry, and drop off sensor nodes we have developed a dynamic buoyancy and balance system. These systems can adjust for a payload of over 1kg attached to the base of the robot in under 30 seconds. The balance is adjusted by moving the battery pack up and down in the robot. The buoyancy is adjusted by moving a piston which changes the internal volume of the robot. These systems save energy as the thrusters need not be used to compensate for the balance and buoyancy changes in the robot. For details, see [32, 102].

3.4 System Analysis and Experiments

In this section we analyze the performance and present experiments testing some of the components of the sensor network system. We start by presenting the energy usage of the components of the system. We then discuss experimental results that test the depth adjustment system. Next, we present experiments to characterize the three communication systems. Finally, we present an analysis of the amount of energy and time it takes to transmit data from the sensor network to land using each of the communication systems.

3.4.1 Energy Usage

Table 3.1 characterizes the power usage of the subcomponents of the AQUANODES. The acoustic and optical modems dominate when transmitting, however, the optical requires very little power when receiving, whereas the acoustic modem requires a significant amount of power even when waiting to receive a message. With all devices disabled and the midlayer board in standby mode, the system uses less than 1mA.

Component	Current (mA)
Base Board Sleep Mode	2
CPU Low Usage	16
CPU Max Usage	59
FPGA	<1
Base Sensors	6
GPS no fix	44
GPS fix	35–40
900 MHz Radio Receive Only	20
900 MHz Radio Transmit 1 Hz	73
Underwater Midlayer Standby	<1
Underwater Midlayer Active	15
Acoustic Receive	110
Acoustic Transmit	200
Optical Receive	15
Optical Transmit	100–500

Table 3.1: Subsystem power usage.

Adding the main board in with a low processing load yields a current usage of just over 16mA. Turning on sensors on the main and midlayer board draws an additional 21mA. This is significantly less than the radio, optical, and acoustic modem. This indicates that minimizing communication is extremely important to maximizing the lifetime of the sensor network.

3.4.2 Depth Adjustment System Performance

We performed experiments to characterize the depth adjustment mechanism in air, in a pool, and in a river. In addition, we performed a preliminary experiment to illustrate how acoustic communication is affected by changing the depth of the underwater sensor nodes.

Trials In Air

We performed experiments in air to characterize the performance of the depth adjustment system under varying loads. These experiments account for over 12 hours of near continuous motion of the depth adjustment system with greater than typical loads (200g) and serve as a stress test for the system.

We began by characterizing the depth adjustment system's lifting capabilities. We calculated the theoretical stall torque of the system. The motor provides 3.6mNm of torque and the spur gearhead has an efficiency of 86%. With an average spool diameter of 20mm, this results in a computed lifting force of 38.2N or a weight of approximately 3.9kg. We experimentally verified this by attaching the depth adjustment system to a spring scale. The system could support up to 3.4kg before stalling. This results in a timing belt and magnetic coupler system efficiency of 87%.

Next, we setup a compound pulley system that enables the winding and unwinding of over 20m of line. We adjusted the weight on the end of the pulley to vary the load on the depth adjustment system. Table 3.2 shows the current needed from the 3-cell Li-Ion battery (nominal voltage of 12V) to move the test rig up and down. The table also lists the average speed as well as the total amount of time the depth adjustment system could operate with the 60WHrs of energy available on-board the AquaNodes.

Force(N)	Current Down (mA)	Current Up (mA)	Speed Down ($\frac{m}{min}$)	Speed Up ($\frac{m}{min}$)	Time (Hours)
1.5	68.36	89.12	2.90	2.75	64.63
2.5	66.70	103.29	2.88	2.66	61.69
3.0	70.16	113.85	2.97	2.72	57.60
4.0	76.80	132.48	2.91	2.62	51.42
4.4	86.07	152.76	2.82	2.50	45.41
5.4	83.30	160.01	2.82	2.36	45.64
6.9	81.78	171.95	2.84	2.31	45.11

Table 3.2: Current usage, speed, and total amount of time the winch can move given different forces (in Newtons).

Examining the table provides insights into system operation. The speed increases and the energy decreases when moving down due to gravitational effects. The down speed stays nearly constant; however, the up speed decreases as the force increases. We expect this as the downward motion force tries to increase the motor speed, reaching a maximum. When moving upward the force acts against the motor, reducing the speed as the force increases. Additionally, even with high loads and continuous depth adjustment, the system can operate for nearly 2 days.

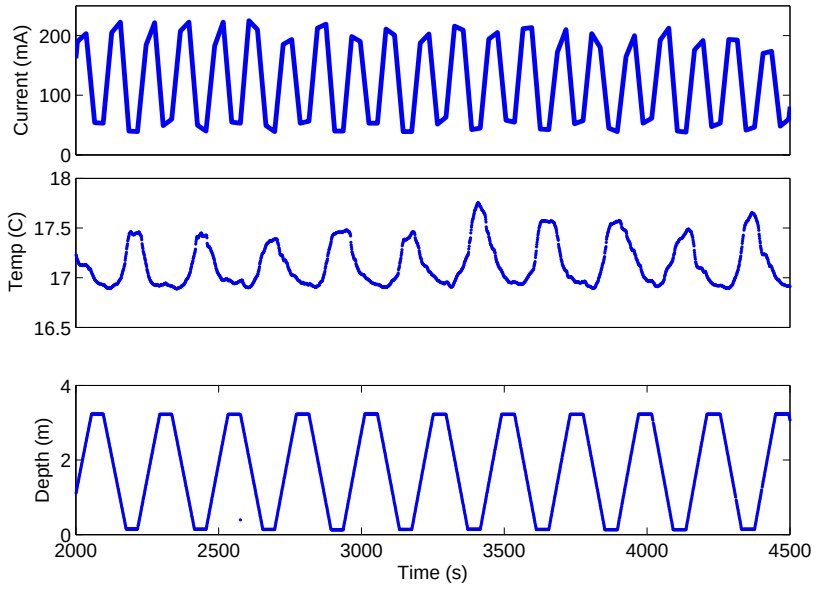


Figure 3-12: Current, temperature, and depth for one AquaNode deployed in the Charles River in Cambridge, MA.

Sensing In Water

We next tested the depth adjustment system in a pool and in the Charles River in Cambridge. We performed 7 multi-hour experiments. These tests characterized the performance of the system in water and illustrated how the depth adjustment system captures temperature variations.

We deployed 3 nodes with the depth adjustment system in the Charles River for 2 hours. Node depths ranged from 2 to 3.5m over the course of the experiment. The nodes traversed the water column every 2 minutes. Every second, the sensor nodes recorded temperature, depth, and battery current values.

Figure 3-12 depicts 45 minutes of the results from the experiment for one node. The node used approximately 50mA when idle and about 100mA when in motion. It moved at a speed of 2.40m/min up and 2.44m/min down.

The temperature of the water varied by over 0.5°C from the surface of the water to the bottom. The response of the temperature sensor shows that it takes some

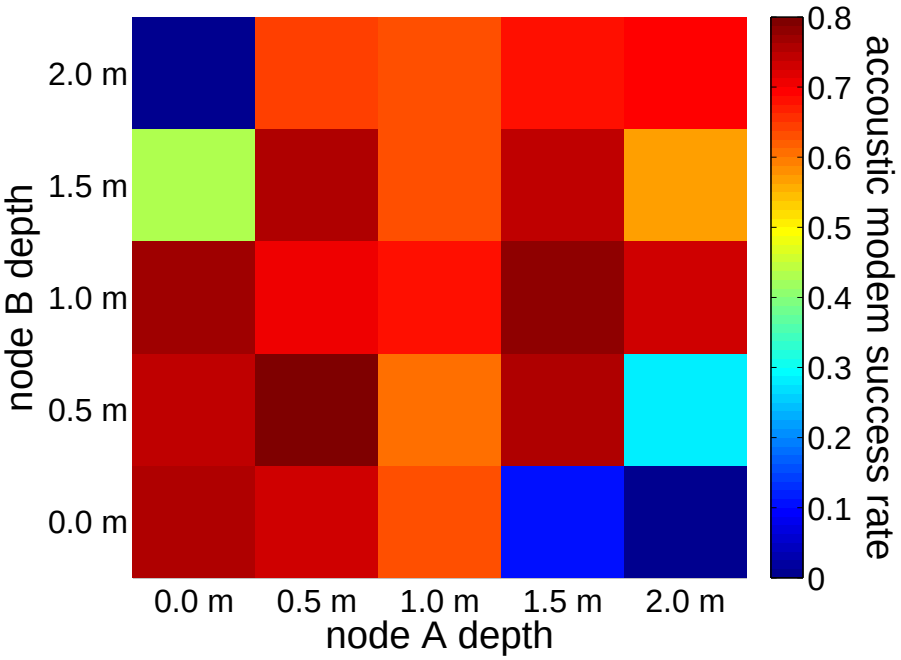


Figure 3-13: Acoustic communication success rate between two nodes spaced 2m apart at varying depths.

time to adjust from one temperature to another indicated by the non-constant values when the node stopped moving. This is most likely caused by the heat capacitance of the temperature sensor. This indicates that at full speed the node moves too fast to accurately measure the temperature. To obtain accurate water column readings, the node must move more slowly or pause for longer periods at each depth.

Improving Communication

The acoustic communication channel is a challenging and fickle medium. The physical characteristics of the water and ocean floor greatly impact the success rate of acoustic packet reception. For example, thermoclines (temperature layers found in water) can reflect acoustic signals, preventing communication between layers [90]. Adjusting the depth of the nodes in the network to optimize reception has the potential to greatly increase the network throughput.

We performed preliminary experiments to illustrate the impact of adjusting node depth on acoustic communication. We tested this in the Charles River. Due to boat

traffic, we placed the nodes on the inner side of a pier, which has walls on three sides. This location is more akin to a pool than an open river. Shallow, closed-in environments challenge acoustic communication as the walls reflect the acoustic signals, causing a high level of interference [77, 82, 97, 107].

We placed 2 AquaNodes approximately 2 meters apart in water that was about 2.5m deep. The 2 nodes each individually moved in 0.5m steps from 0m to 2m depth. At every combination of depths each node transmitted approximately 25 packets. The nodes were spaced closely to see the impact of changing the depth in the acoustics (as the propagation of the signal is cylindrical, not spherical). In the future we plan to do experiments with larger spacing in deeper water to see the impact of thermal and bottom features on the acoustic communication.

Figure 3-13 shows the results of the packet success rate for this experiment. The success rates are asymmetric. For instance, at depths for (A,B) of (1.5,1.0) the nodes obtain better communication performance than at (1.0,1.5). On average, the acoustic modems had the greatest success rate when they were both at similar depths, whereas when the nodes were at the extremes very few packets were successful. The lack of communication is most likely caused by the non-spherical signal propagation of our cylindrical acoustic transducer. Our transducers transmit well horizontally in the water, but poor vertically. For larger inter-node ranges, this does not impact the system. However, when the nodes are closely spaced, this reduces the transmission successes for nodes at very different depths.

For more complex acoustic channels, for instance with thermoclines, we expect these discrepancies and variations to be even more pronounced. In addition, the channel quality may vary over time. As such, adjusting the depth of the nodes has the potential to greatly improve the acoustic communication in the network.

Observations and Potential Improvements

In the tests we observed some other characteristics of the system. First, due to manufacturing tolerances, some of the depth adjustment mechanisms have more friction than others. This means the motors must work harder to overcome the friction.

The motors are sufficiently strong to do this, however, this wastes energy. In the experiments we observed some winches that used as much as 100mA more to move.

The second observation is the effect of waves and currents on the depth adjustment system. These can cause the sensor node to move such that the line is no longer taught. This does not cause problems if the winch is reeling in the line, but if it is letting out line it is possible that this slack can cause the winding of the line to be reversed. For instance, if the sensor node is moving to the surface and a wave causes slack in the line it is possible the winding direction will reverse. This will cause the depth adjustment system to start moving down in the water instead of continuing up.

The problem of the winding direction reversing can be addressed in a number of ways. First, adding buoyancy or lowering the depth adjustment system speed reduces the likelihood of this occurring. The winch speed can be adjusted based on the waves in the water (perhaps detected using the on-board accelerometer). Second, the depth sensor can be used to detect this problem. Reversing the winch when this occurs alleviates this problem. Finally, a pretensioning system, as found in some commercially available winches, can be added to address this problem. In practice, all three of these approaches should be implemented to eliminate this problem.

3.4.3 Communication System Performance

Device	**Physical Layer Rate**	**Real Rate**	**Maximum Range**	**Typical Range**	**Success Rate**
Optical Modem	1Mbit/s	800Kbit/s	4m	3m	90%
Bluetooth	1Mbit/s	92.1kb/s	50m	5m	100%
Serial Cable	115200b/s	92.1kb/s	300m	1m	100%
900MHz Radio	57600b/s	7.2kb/s	3km	100m	25-50%
Acoustic Modem	300b/s	22b/s	400m	100m	56%

Table 3.3: Summary of the bitrates of the various communications we use. Real rate is the throughput assuming 100% transmission success rate. This takes into account the (estimated) overhead which is significant for some of our devices. Success rate is the packet success rate at the typical range.

Table 3.3 summarizes the results of experiments with 5 of the different communication methods available on the underwater sensor nodes. In this table the physical

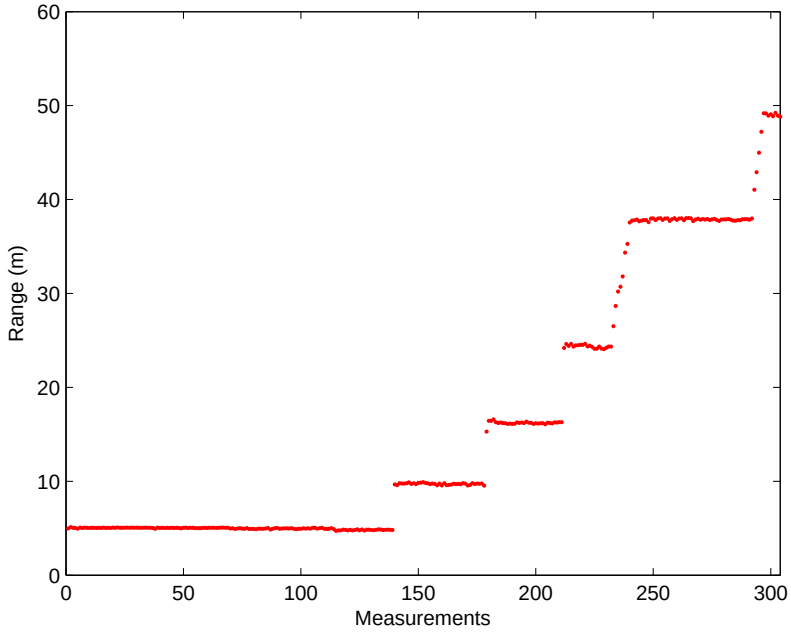


Figure 3-14: Ranges between a pair of acoustic modems in 2 to 4m water depth.

layer data rate is shown as well as the real data rate. The real data rate is the measured rate. The measured rate takes into account the overhead each system. For instance, the 900MHz radio broadcasts each packet in 6 times and shares the channel to simulate full-duplex communication. The acoustic modem introduces numerous guard times to prevent reflections, thus greatly reducing the actual throughput.

Also shown in the table is the maximum obtained range of the system as well as the maximum range we typically use. At this range the typical packet success rate is listed. Direct serial connections and bluetooth links have near 100% success rate. The former is a direct connection with little noise. The latter guarantees packet transmission at a low level. The Aerocomm radio has the lowest success rate at a range of 100m (far below the claimed maximum range). This may be due to channel interference and the fact that the radio does not adjust its physical baud rate to compensate for interference.

Acoustic Modem Underwater

Figure 3-14 shows the ranges computed between a pair of sensor nodes using the acoustic modem. One node was stationary and the second was moved in a step-wise fashion. Ground-truth measurements were obtained for the first four measurements by using a long measuring tape. The accuracy of the ground truth measurements is not high, however, it does show that the ranges correspond well. Table 3.4 shows the average measurement obtained from the acoustic modem and the ground truth for the first four measurements. The two values are very close together and are within the ground truth measurement error.

Acoustic Modem Distance (m)	**Ground Truth Distance (m)**	**Difference/Error (m)**
5.01	4.75	0.26
9.72	9.67	0.05
16.19	16.15	0.04
24.35	24.70	-0.35
37.88	N/A	N/A
49.03	N/A	N/A

Table 3.4: Range measurements in meters between a pair of acoustic modems.

Figures 3-15 and 3-16 show the results of an experiment where four static nodes were deployed in Lake Otsego, NY. The depth of the water was approximately 10 meters deep and relatively clear and cool. The nodes self-localized and then the robot drove through the network collecting ranging data. A range to a node was obtained every one to four seconds.

Figure 3-15 plots range data to the four nodes over a 20 minute selection of the hour long experiment. As can be seen from this plot, ranges are fairly continuous and there are very few outliers. Outliers are typically caused by receiving the multipath of a message instead of the direct path message. This causes an increase in the range measurement as the message took longer to arrive than typical. Additionally, there are a few measurements that are lower than expected. These can be explained by multipath messages from the previous time slot, which arrive in the current time slot. This is one of the many challenges associated with underwater communication and

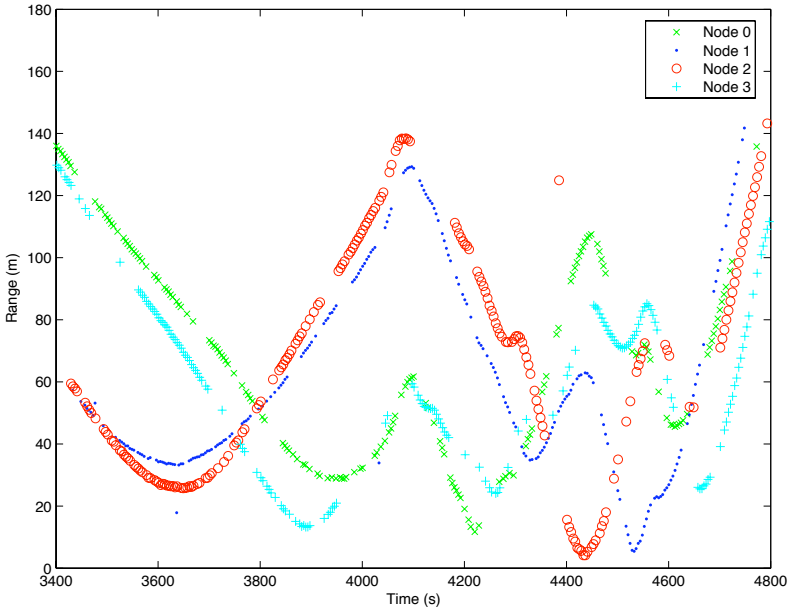


Figure 3-15: Ranges received by the robot while moving around in a network with four static nodes.

ranging (since we use the round-trip time to compute ranges). We avoid this problem in most cases by having a sufficiently long enough guard time, which allows most of the reflections to die down. These outliers are easily eliminated using a simple smoothness filter.

This data also shows that we successfully obtain ranges up to the maximum $140m$ range that we traveled. We also have successfully obtained ranges at up to $400m$, however, at this larger range the communication is less reliable.

Figure 3-16 illustrates the round-trip message success rate for this same set of data. Over this experiment we found that about 59% of messages were successful. This is typical for most of our experimental sites except for the Charles River where we find a much lower message success rate.

Notice that there are periods where the robot is unable to obtain ranges for 10s of seconds from a particular node. Figure 3-15 shows that there is very little correlation between the distances at which these communication failures occur. For instance, the data for Node 2, shows that there is a period when no ranges are received when it

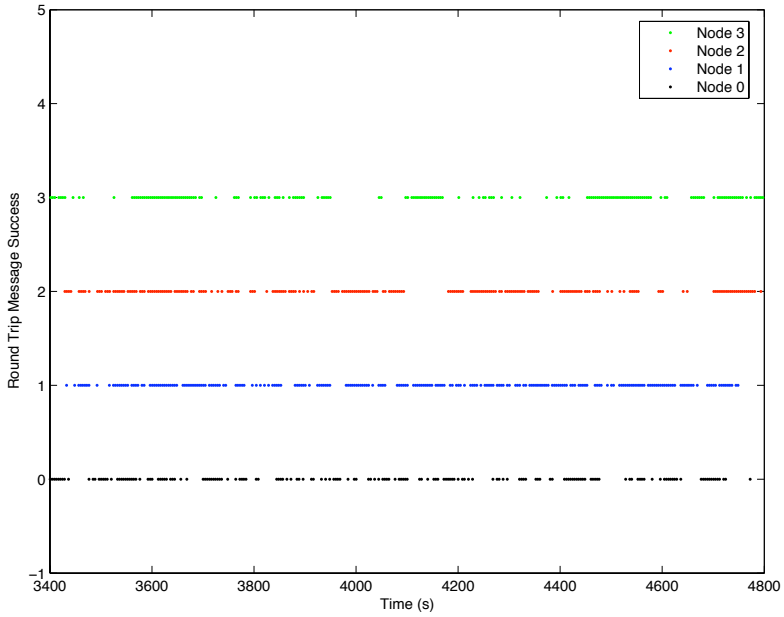


Figure 3-16: The successful round-trip messages from the robot to four different nodes.

is over $100m$ away, but there is a similar communication gap at $20m$, later in the experiment. Other times the robot is able to obtain most all messages to Node 2 at over $100m$. This indicates that the communications success rate is not merely a function of range, rather it is highly depending on the particular configuration of the nodes and the water conditions.

Radio Modem

We collected data to characterize the performance of the Aerocomm 900MHz radio. Figure 3-17 shows the results of these tests. This data was collected as part of the cow virtual fencing project [35]. Each cow had one of the base sensor network boards and was allowed to roam over a large field. The radio antenna was attached on top of the heads of the cows. Every two minutes each cow broadcast its own GPS position over the radio. The sensors were deployed many days collecting data.

Figure 3-17 shows percent of the time pairs of cows were at particular distances from each other (dashed line). Most of the time, the cows were grouped together

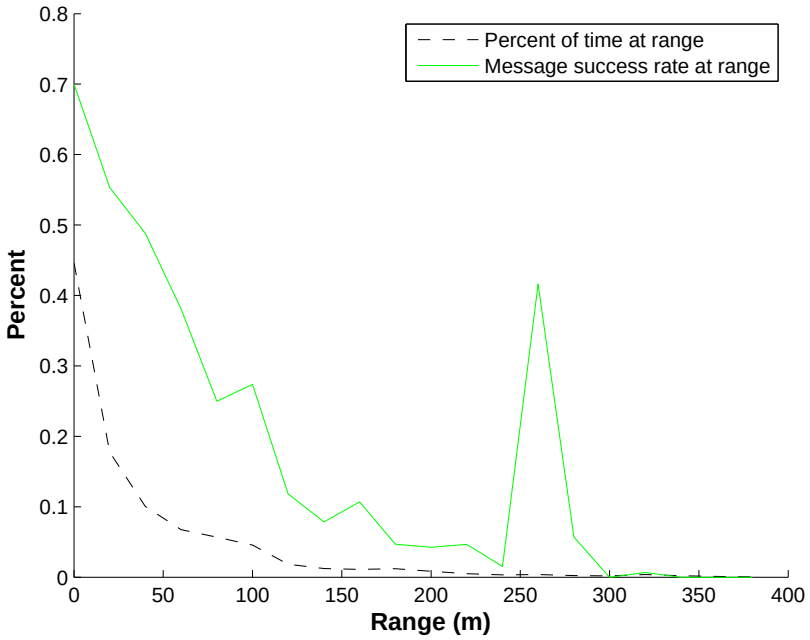


Figure 3-17: The success rate of radio packets at different ranges. Also shown is the amount of time spent at each range.

within 100 meters of each other. However, on occasion, cows were separated by over 300m. In addition, the figure shows the message success rate based on distance. When extremely close, the success rate was near 70%. Once the distance increased to over 150m, however, the success rate dropped below 10%. The only exception is a peek at slightly over 250m, which was most likely caused by the low number of samples at this distance influencing the statistics.

This experiment shows that the performance of the radios is poor even for relatively close ranges over flat land. We also tested the radios between the Longfellow and Harvard bridge in Cambridge. These bridges are slightly over 1km apart. We obtained just over 50% of the packets at this distance with line-of-sight over water.

Optical Modem

We performed experiments to characterize the performance of the optical modem [36]. Figure 3-18 shows the results experiments performed in the Singapore Harbor, which

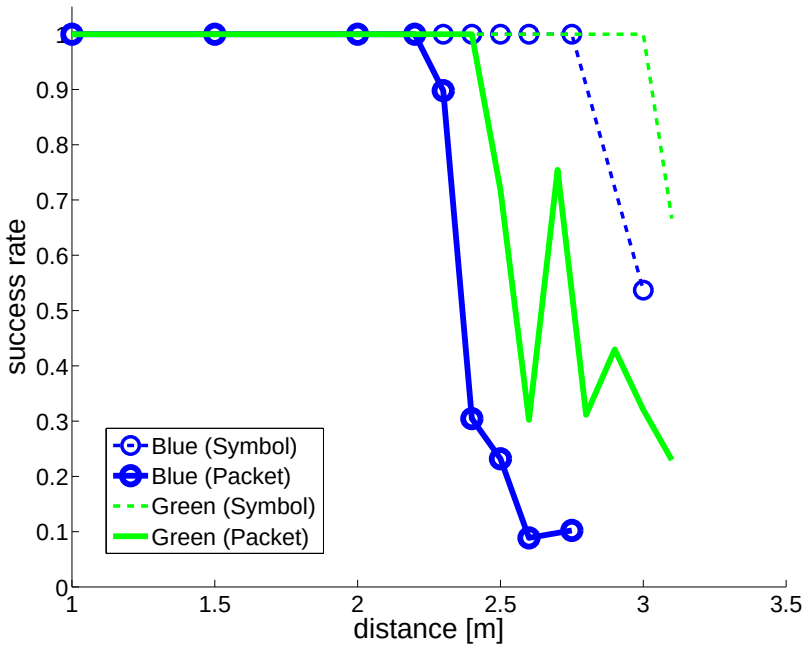


Figure 3-18: Optical modem range experiments performed in Singapore Harbor.

had less than 3m of visibility. The optical modems were placed at a depth of 4m in the water to reduce the effect of ambient light on the experiment. The modem transmitted 128 byte packets at a 1.2Mbit/sec. We tested green and blue LEDs.

Figure 3-18 shows both the full packet and symbol success rate versus distance. The symbol success rate roughly corresponds to receiving individual bits, while the packet success rate indicates receiving a full 128 byte packet correctly. The optical modem with the blue LEDs outperformed the modem with the green LEDs by approximately .25m. For both, the success rate at 2.25m was 100% for both the packet and symbols. At 2.5m the blue LED rate was about 10% while for green it was over 60%. After this the packet success rate drops significantly, while the symbol success rate remains at near 100% until 2.75m and 3.0m for blue and green respectively.

This experiment and others show that the optical modem has a very strong cutoff distance after which it performs poorly. Before this cutoff it performs extremely well.

3.4.4 Communication Energy and Time

The amount of energy used by underwater sensor nodes is critical to the longevity of the system. It is extremely difficult to recharge devices underwater and it is not feasible to regularly replace batteries or devices. The amount of energy used for computation and storage has decreased radically in the past years. Unfortunately, the amount of energy used for communication has not dropped. In our system, and most underwater as well as terrestrial sensor networks, the power requirements of communication far outweighs the power used by all other systems on the sensor node. Thus, a key consideration of our system is the amount of energy used to transmit each bit.

In this section we examine the amount of energy and time used by the three different communication systems to transmit data back to shore. The first system is the acoustic communication system, which uses sound waves to communicate. The second is the radio modem, which works at the water surface. To get to the surface of the water we make use of our depth adjustment system. The third system is the short-range optical communication system that we use in conjunction with our underwater robot. The robot drives to the AquaNode, downloads the data optically and then returns with the data.

Acoustic

The acoustic modem has a maximum transmission power of about 10 watts. However, it typically operates using a lower power mode that uses about 5 watts. Recall that the modem uses a decentralized TDMA algorithm. In each 4 second TDMA slot we are able to send and receive a 16 byte packet with 11 bits of payload. Thus we have a throughput of 5.5 bytes per second. This translates to an acoustic power per bit, P_a, of:

$$P_a = 5\,\mathrm{W}/(5.5 * 8\,\mathrm{bits/sec}) = 113.6\,\mathrm{mJ/bit}. \tag{3.1}$$

Depth Adjustment and Radio

Radio waves do not propagate underwater, however, by using the depth adjustment system the AquaNodes can go to the surface and use the radio to communicate. We use a 1 watt Aerocomm AC4790 900 MHz radio [2].

The radio has a theoretical range of 32km, although in our experiments we have only achieved slightly over 3km. The physical layer has a fixed datarate of 76800 bits per second. However, in broadcast mode the radio transmits each packet 6 times to increase the probability of being received. Additionally, to simulate full-duplex operation following each packet transmission it waits for a period equal to the time it would take to receive another packet to allow other radios to transmit. This means the true datarate for this radio is closer to

$$76800/6/2\,\text{bits/sec} = 6400\,\text{bits/sec}. \tag{3.2}$$

We can then calculate the power per bit using the radio, P_r to be:

$$P_r = 1\,\text{W}/6400\,\text{bits/sec} = 0.16\,\text{mJ/bit}. \tag{3.3}$$

However, to send using the radio it must first go to the surface using the depth adjustment system. The depth adjustment system uses about 0.6 watts and moves at 2.4m/min. Thus, we need a power per meter, P_w of:

$$P_w = 0.6\,\text{W}/0.04\,\text{m/sec} = 15000\,\text{mJ/m}. \tag{3.4}$$

The total power, P_{rw} to transmit k bits from a depth of d meters using the radio and depth adjustment system (assuming we return to the same location after) will be:

$$P_{rw} = 2dP_w + kP_r = 2d * 15000\,\text{mJ} + k * 0.16\,\text{mJ}. \tag{3.5}$$

Robot and Optical

The third method of communication in our system is to have the robot drive out to the sensor node and download the data optically.

The optical communication system is a high speed point-to-point short range communication system. The range is about 2 meters in water and it is somewhat directional. The transmission rate is adjustable, but typically we use a 1 megabit baud rate while using 7 watts. The power per bit for the optical system, P_o is:

$$P_o = 7\,\mathrm{W}/1000000\,\mathrm{bits/sec} = 0.007\,\mathrm{mJ/bit}. \tag{3.6}$$

This is many orders of magnitude less energy per bit than the acoustic and radio. However, the robot requires a lot of energy to travel to the sensor node. At typical cruising speed the robot uses 300 watts and travels at 0.5 meters per second. So the energy used per meter traveled, P_m is

$$P_m = 300\,\mathrm{W}/0.5\,\mathrm{m/sec} = 600,000\,\mathrm{mJ/m}. \tag{3.7}$$

The total power, P_{om}, to travel a distance D meters to a sensor node, transfer k bits and then return is:

$$P_{om} = 2DP_m + kP_o = 2D * 600,000\,\mathrm{mJ} + k * 0.007\,\mathrm{mJ}. \tag{3.8}$$

Energy and Time Comparison

Figure 3-19 plots the total amount of power needed to send various amounts of data. This plot is based on equations 3.1, 3.5 and 3.8. For this example we are assuming that we want to transmit data 500 meters and that the sensor node is at a depth of 10 meters.

Figure 3-19 is on a log-log scale so that the large ranges of power usage depending on the number of bytes sent is visible. The power used by the acoustic system increases at a fast and constant rate as each additional bit sent requires the same amount of

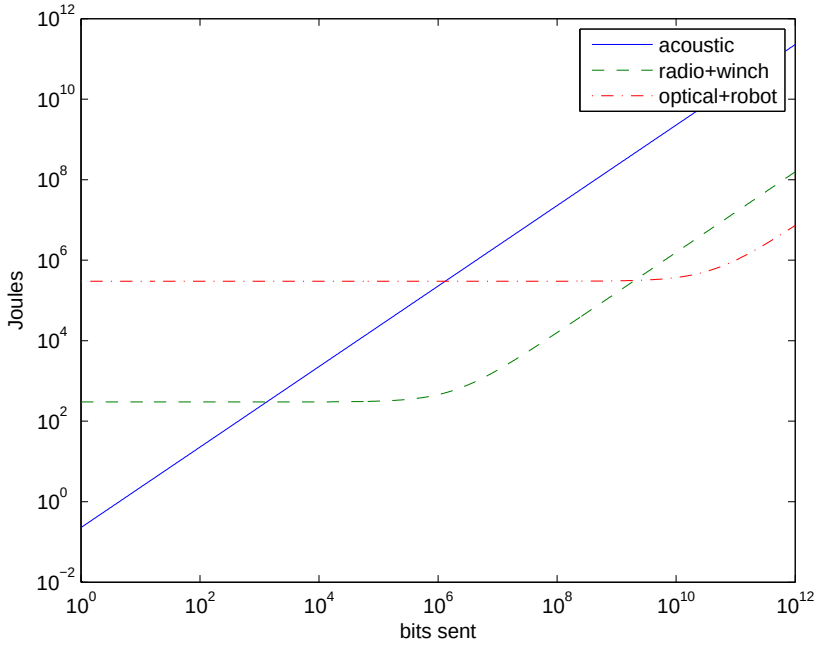


Figure 3-19: Power per transmission size on log-log plot for a message sent 500 meters from a depth of 10 meters using each of the three communications systems

additional energy. For the radio and optical systems there is a large initial amount of required energy, but then very little additional energy per bit. This initial offset is the amount of energy required to drive the robot to and from the sensor for the optical download and the energy to bring the sensor node to and from the surface to use the radio.

The power usage plot shows that there are clear decision points as to where to should switch between the three communication systems. If sending less than 1328 bits, the acoustic system should be used. To send between 1328 bits and 1.9 gigabits it is more power efficient to first go to the surface with the depth adjustment system, transmit with the radio and then return to the previous depth. Finally, sending more than 1.9 gigabits it requires less power to send the robot to the AquaNode and download the data optically.

Figure 3-20 similarly shows the amount of time it takes to send data. Interestingly, the three communication systems have the same ordering as in the power analysis.

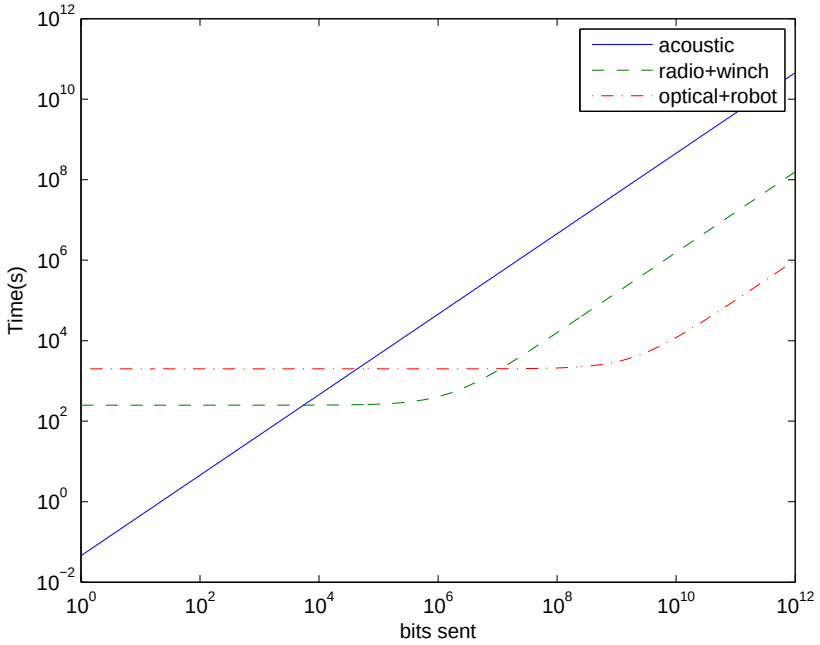


Figure 3-20: Time to send bits on log-log plot for a message sent 500 meters from a depth of 10 meters using each of the three communications systems

However, the switching points are somewhat different. For this plot it is fastest to send acoustically for less than 5594 bits. Faster to send after going to the surface with the radio for between 5594 bits and 11.5 megabits. And faster to use the robot and optical system for any larger data transmissions.

These two plots give two of the possible considerations when trying to decide which communication system is best to use. However, there are other considerations. For instance, it is easier to recharge the robot than it is to recharge the sensor node. So it may be better to send the robot out to the sensor nodes to collect data even if it is a relatively small amount of data. Additionally, if the robot is downloading data from many sensor nodes the initial constant cost of going out to the sensor nodes can be amortized over all the sensor nodes. Using the depth adjustment system to go to the surface has the drawback that the sensors are no longer at the desired depth. Thus, it may be advantageous to use the acoustic system for larger amounts of data if it is important for the sensor to remain at a particular depth.

Chapter 4

Decentralized Depth Adjustment: Algorithm

4.1 Introduction

The vastness of the ocean prohibits placing sensors at every location of interest in order to monitor underwater phenomena. As such, models are required that take some sensed locations as input and derive the value of the field at other locations. Modeling the surface characteristics is challenging, and to fully understand the depths scientists must develop new ocean models that cover the full 3D system.

The AquaNode's depth adjustment system provides the capability to measure the ocean at a variety of locations and depths over time. This system adds to the wealth of knowledge that is starting to be collected by underwater gliders, profiling floats, AUVs, ROVs, and towed instruments. The recent development of optical and acoustic sensors, such as laser optical plankton counters (LOPC), acoustic Doppler current profilers (ADCP), and laser in-situ scattering and transmissometry (LISST) aids these mobile sensing platforms [59].

The rapid accumulation of data and an increased understanding of ocean processes has led to the advancement of ocean modeling and the development of new modeling techniques, such as massive parallel programming and data assimilation [38]. These models require accurate measurements of the spatial and temporal characteristics of

ocean phenomena such as subsurface chlorophyll maxima, coastal fronts and jets, river plumes, and meso-scale eddies. Measuring these phenomena, however, still remains a challenge due to the highly dynamic nature of the processes and the slow response time of most autonomous instruments. The lack of precise measurements and characterizations of these phenomena poses a hurdle to advancing the accuracy of ocean models. In the dynamic, shallow, coastal waters this is especially true.

The AquaNodes are low-cost, flexible systems that are well-suited for studying coastal processes. In this chapter we develop and analyze a decentralized, adaptive algorithm for positioning an underwater sensor network. Our AquaNodes dynamically adjust their depths through a new decentralized gradient-descent based algorithm with guaranteed properties. Through neighbor communication this algorithm collaboratively optimizes their depths for placement from sensing in support of computing maximally detailed volumetric models. We prove the controller algorithm converges to a local minimum. In Chapter 5 we verify the algorithm with simulations and experiments on our hardware platform.

The decentralized controller positions the nodes such that they are in good locations to collect data to model the values of the system over the whole region, not just the particular points where there are sensors. This allows the reconstruction of the whole sensory field, not just sensing at the locations of the sensors. The sensor nodes use a covariance function that describes the relationship between the possible positions of the sensor nodes and the whole region of interest. As a first pass, we model the covariance as a multivariate Gaussian, as is often used in objective analysis in underwater environments [69]. In Chapter 5 we extend this and compute a numeric covariance for the Neponset river using a physics-based hydrodynamic model. The algorithm assumes a fixed covariance model, however, it can be iterated with different covariance models to capture dynamic phenomena.

The controller uses the covariance in a decentralized gradient descent algorithm. This algorithm requires very little communication, allowing each node to only send its own depth information, as well as providing fault tolerance in instances where communication packets are lost. Both of these are important in underwater sensor

networks, which can only communicate acoustically—a low bandwidth (300b/s) and limited reliability (<50% packet success) communication method. Our algorithm has limited memory and computation requirements, allowing it to run in real-time on our power efficient underwater sensor network.

In Chapter 3 we detailed the underwater sensor network, AquaNodes, and the winch-based depth adjustment system. In this Chapter we start by introducing and analyzing our decentralized depth controller algorithm in Section 4.2. Next, we prove it converges in Section 4.2.7. We then present the implemented algorithm in Section 4.2.8. Finally, we extend the algorithm to cases where the covariance is changing to allow sensing dynamic phenomena in Section 4.3. Chapter 5 verifies the algorithm in simulation and experiments on the AquaNode platform.

4.2 Decentralized Control Algorithm

In this section we formulate the problem, develop a general decentralized controller, introduce a Gaussian covariance function, define the controller in terms of the covariance function, and prove the convergence of the controller.

4.2.1 Problem Formulation

Given N sensors at locations $p_1 \cdots p_N$ we want to optimize their positions for providing the most information about the change in the values of all other positions $q \in Q$, where Q is the set of all points in our region of interest. We are especially interested in the case where the motion of the sensors, p_i, is constrained to some path $R(i)$. In the case of our underwater sensor network the nodes are constrained to move in 1D along z with fixed x, y.

The best positions to place the sensors are positions that tell us the most about other locations. Consider the case with one sensor at position p_1, and one point q_1 of interest. Intuitively, we want to place p_1 at the location along its path that is closest to q_1. At this position any changes in the sensory value at q_1 are highly correlated to observed changes we measure at p_1. More generally, this correlation is captured by covariance, so the sensor should be placed at the point of maximum covariance

with the point of interest. Or more formally, position p_1 such that the $Cov(p_1, q_1)$ is maximized.

More generally, we want to maximize the covariance between the point of interest q_1 and all sensed points p_i by moving all p_i to maximize:

$$\arg\max_{p_i} \sum_{i=1}^{N} Cov\left(p_i, q_1\right) \tag{4.1}$$

This is for the case of one point q_1, if we have M points of interest in the region Q, we can add an additional sum over the points of interest:

$$\arg\max_{p_i} \sum_{j=1}^{M} \sum_{i=1}^{N} Cov(p_i, q_j) \tag{4.2}$$

This objective function, however, has the problem that some areas may be covered well, while others are not covered. Figure 4-1 shows the case with three sensors, p_1, p_2 and p_3, covering two points, q_1 and q_2. For this example we assume that p_1 and p_3 are fixed and look at the effect of moving p_2. Both configurations in Figure 4-1 yield an objective function value of $.5 + .5 + .5 = 1.5$. This contradicts the intuition that the configuration on the right is better.

To prevent the problems associated with Equation 4.2 and illustrated in Figure 4-1 we need to ensure that the objective function penalizes regions that are already covered by other nodes. We achieve this by modifying the objective function to minimize:

$$\arg\min_{p_i} \sum_{j=1}^{M} \left(\sum_{i=1}^{N} Cov(p_i, q_j) \right)^{-1} \tag{4.3}$$

Instead of maximizing the double sum of the covariance, this objective function minimizes the sum of the inverse of the sum of covariance. This reduces the increase in the sensing quality achieved when additional nodes move to cover an already covered region.

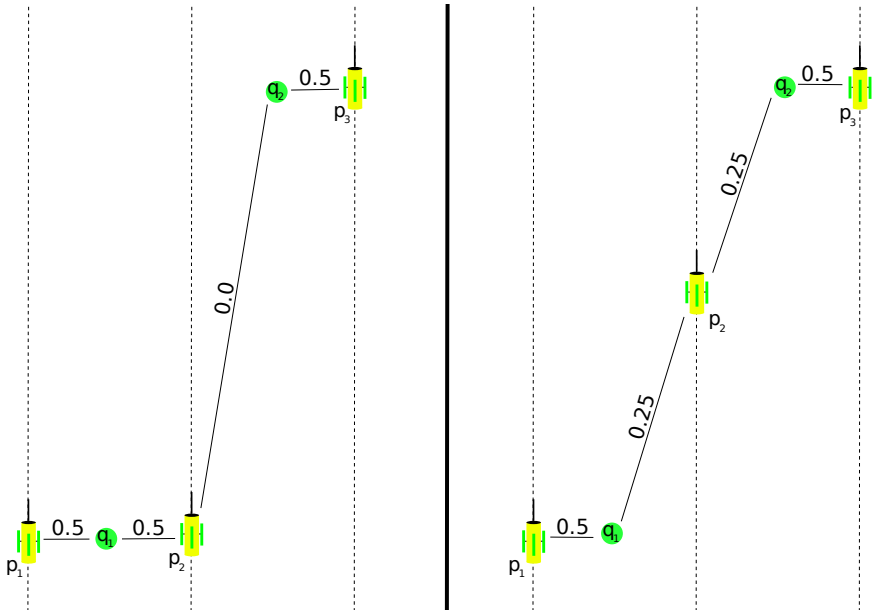


Figure 4-1: Dashed lines are the motion constraints on the AquaNode motion, green circles are the points of sensor interest. Solid lines and the labels indicate the covariance between the point of interest and the indicated sensor.

This changes the example at left in Figure 4-1 to give an objective value of:

$$\frac{1}{.5+.5}+\frac{1}{.5}=3 \tag{4.4}$$

It changes the right side in Figure 4-1 to:

$$\frac{1}{.5+.25}+\frac{1}{.5+.25}=2\frac{2}{3} \tag{4.5}$$

Our new minimization of the objective function will select the rightmost configuration in Figure 4-1, which is intuitively better.

To extend this to optimize placement for sensing every point in the region, we modify the objective function to integrate over all points q in the region Q of interest:

$$\int_Q \left(\sum_{i=1}^{N} Cov(p_i,q)\right)^{-1} dq \tag{4.6}$$

4.2.2 Assumptions

The decentralized depth control algorithm we develop in this chapter makes some assumptions about the system. These are:

- The nodes know their locations;
- The nodes can communicate with all other nodes (shown in simulation that only neighborhood communication is needed);
- The nodes can adjust their depths;
- The nodes know the covariance function.

The nodes are able to determine their locations either by going to the surface and obtaining a GPS fix or using a static localization algorithm based on range data. Both of these are discussed in Section 6.2.2. The nodes communicate with each other using the acoustic modem. In practice, all nodes will not be able to talk with each other, however, we show in simulation in Section 5.4.2 that the nodes only need to communicate within a local neighborhood in the network. The algorithm assumes that the nodes are able to adjust their depths. We have implemented a depth adjustment system on the AquaNodes that enables this.

The most difficult assumption made by the algorithm is that the covariance function between node locations and points of interest is known. Section 5.2 discusses how a model of the covariance can be derived using physical models of the system. An initial model can be chosen and then updated online based on recent measurements. If nothing is known about the system, the nodes can be deployed for a period of time to collect data that can be used to determine the covariance of the system.

4.2.3 Objective Function

The objective function, $g(q, p_1, ..., p_N)$, is the cost of sensing at point q given sensors placed at positions $p_1, ..., p_N$. For N sensors, we define the sensing cost at a point q as:

$$g(q, p_1, ..., p_N) = \left(\sum_{i=1}^{N} f(p_i, q) \right)^{-1} \tag{4.7}$$

This is the inner part of Equation 4.6 when $f(p_i, q) = Cov(p_i, q)$.

Integrating the objective function over the region of interest gives the total cost function. We call this function $\mathcal{H}(p_1, ..., p_N)$ and formally define it as:

$$\mathcal{H}(p_1, ..., p_N) = \int_Q g(q, p_1, ..., p_N)\ dq + \sum_{i=1}^{N} \phi(p_i) \tag{4.8}$$

where Q is the region of interest. The sum over the function $\phi(p_i)$ is a term added to prevent sensors from trying to move outside of the water column. We need this restriction on the node's movement to prove convergence of the controller for this cost function. Specifically, we define $\phi(p_i)$ as:

$$\phi(p_i) = \left(\frac{z_i - (d_i/2)}{d_i/2} \right)^{\beta} \tag{4.9}$$

where d_i is the depth at the location p_i and β is an even positive value. The $\phi(p_i)$ component causes the cost function to be very large if a sensor is placed outside of the water column.

4.2.4 General Decentralized Controller

Given the objective function in Equation 4.8, we wish to derive a decentralized control algorithm that will move all nodes to optimal locations making use of local information only. We derive a gradient descent controller which is localized, efficient, and provably convergent.

Our goal is to minimize $\mathcal{H}(p_1, ..., p_N)$, henceforth referred to as $\mathcal{H}$. To do this we

start by taking the gradient of $\mathcal{H}$ with respect to each of the z_is:

$$\begin{aligned}
\frac{\partial \mathcal{H}}{\partial z_i} &= \frac{\partial}{\partial z_i} \int_Q g(q, p_1, ..., p_N)\ dq + \frac{\partial}{\partial z_i} \sum_{j=1}^{N} \phi(p_j) \\
&= \int_Q \frac{\partial}{\partial z_i} \left(\sum_{j=1}^{N} f(p_j, q) \right)^{-1} dq + \frac{\partial}{\partial z_i} \phi(p_i) \\
&= \int_Q - \left(\sum_{j=1}^{N} f(p_j, q) \right)^{-2} \frac{\partial}{\partial z_i} \sum_{j=1}^{N} f(p_j, q)\ dq + \frac{\partial}{\partial z_i} \phi(p_i) \\
&= \int_Q - \left(\sum_{j=1}^{N} f(p_j, q) \right)^{-2} \frac{\partial}{\partial z_i} f(p_i, q)\ dq + \frac{\partial}{\partial z_i} \phi(p_i) \qquad (4.10) \\
&= \int_Q -g(q, p_1, ..., p_N)^2 \frac{\partial}{\partial z_i} f(p_i, q)\ dq + \frac{\partial}{\partial z_i} \phi(p_i) \qquad (4.11)
\end{aligned}$$

Next, we take the partial derivative of $\phi(p_i)$ and find:

$$\frac{\partial}{\partial z_i} \phi(p_i) = \frac{\partial}{\partial z_i} \left(\frac{z_i - (d_i/2)}{d_i/2} \right)^{\beta} = \beta \left(\frac{z_i - (d_i/2)}{d_i/2} \right)^{\beta - 1} \qquad (4.12)$$

Thus, the partial derivative $\frac{\partial \mathcal{H}}{\partial z_i}$ is:

$$\int_Q -g(q, p_1, ..., p_N)^2 \frac{\partial}{\partial z_i} f(p_i, q)\ dq + \beta \left(\frac{z_i - (d_i/2)}{d_i/2} \right)^{\beta - 1} \qquad (4.13)$$

To minimize $\mathcal{H}$ we move each sensor in the direction of the negative gradient. Let $\dot{p_i}$ be the control input to sensor i. Then the control input for each sensor is:

$$\dot{p_i} = -k \frac{\partial \mathcal{H}}{\partial z_i} \qquad (4.14)$$

where k is some scalar constant. Moving each node in the negative direction of the gradient minimizes the objective function. This controller provides a general controller usable for any sensing function, $f(p_i, q)$. To use this controller we next present a practical function for $f(p_i, q)$.

4.2.5 Gaussian Sensing Function

We use the covariance between points p_i and q as the sensing function:

$$f(p_i, q) = Cov(p_i, q) \tag{4.15}$$

In an ideal case we would know exactly the covariance between the i^{th} sensor and each point of interest, q. As this is not possible, we have chosen to use a multivariate Gaussian as a first-approach approximation of the sensing quality function. Using a Gaussian to estimate the covariance between points in underwater systems is common in objective analysis [69]. In Section 5.2 we show how to numerically estimate the covariance given real or modeled data.

We define the Gaussian to have different variances for depth (σ_d^2) and for surface distance (σ_s^2). This captures the idea that quantities of interest (e.g. algae blooms) in the oceans or rivers tend to be stratified in layers with higher concentrations at certain depths. Thus, the sensor reading at a position p_i and depth d is likely to be similar to the reading at position q if it is also at depth d. However, sensor readings are less likely to be correlated between two points at different depths. Thus, the covariance function is a three-dimensional Gaussian, which has one variance based on the surface distance and another based on the difference in the depth between the two points.

Let $f(p_i, q) = Cov(p_i, q)$ be the sensing function where the sensor is located at point $p_i = [x_i, y_i, z_i]$ and the point of interest is $q = [x_q, y_q, z_q]$. Define σ_d^2 to be the variance in the direction of depth and σ_s^2 to be the variance in the sensing quality based on the surface distance. We then write the sensing function as:

$$f(p_i, q) = Cov(p_i, q) = Ae^{-\left(\frac{(x_i - x_q)^2 + (y_i - y_q)^2}{2\sigma_s^2} + \frac{(z_i - z_q)^2}{2\sigma_d^2}\right)} \tag{4.16}$$

where A is a constant related to the two variances, which can be set to 1 for simplicity. Note if $\sigma_s = \sigma_d$ then this function is merely a 3D Gaussian based on the Euclidean distance between the two points.

In Section 5.5 we use data from a physics-based hydrodynamic model to numerically estimate the covariance of chromophoric dissolved organic matter (CDOM) in a river system. In this system the covariance between points decreases slowly as distance increases along the river in a nearly linear fashion over a range of kilometers. The covariance in depth changes rapidly over a few meters as the area we are interested in exhibits stratification and mixing of CDOM along a large stretch of the river. We represent the covariance in this system using a Gaussian basis function, which comes within 99% of the computed covariance. This basis function allows the compact representation of the covariance and allows similar proofs of convergence of the system. For ease of notation, we focus on the simple single Gaussian representation for the rest of this section. These techniques can be applied to more complex representations.

4.2.6 Gaussian-Based Decentralized Controller

We take the partial derivative of the sensing function from Equation 4.16 to complete the gradient of our objective function shown in Equation 4.11. The gradient of the sensing function $\frac{\partial}{\partial z_i} f(p_i, q)$ is:

$$\begin{aligned}
\frac{\partial}{\partial z_i} f(p_i, q) &= \frac{\partial}{\partial z_i} Cov(p_i, q) \\
&= \frac{\partial}{\partial z_i} A e^{-\left(\frac{(x_i - x_q)^2 + (y_i - y_q)^2}{2\sigma_s^2} + \frac{(z_i - z_q)^2}{2\sigma_d^2}\right)} \\
&= A e^{-\left(\frac{(x_i - x_q)^2 + (y_i - y_q)^2}{2\sigma_s^2} + \frac{(z_i - z_q)^2}{2\sigma_d^2}\right)} \frac{\partial}{\partial z_i} \left(- \left(\frac{(x_i - x_q)^2 + (y_i - y_q)^2}{2\sigma_s^2} + \frac{(z_i - z_q)^2}{2\sigma_d^2} \right) \right) \\
&= A e^{-\left(\frac{(x_i - x_q)^2 + (y_i - y_q)^2}{2\sigma_s^2} + \frac{(z_i - z_q)^2}{2\sigma_d^2}\right)} \frac{-(z_i - z_q)}{\sigma_d^2} \\
&= -f(p_i, q) \frac{(z_i - z_q)}{\sigma_d^2} \qquad (4.17)
\end{aligned}$$

Substituting this into Equation 4.11, we get the objective function:

$$\begin{aligned}
\frac{\partial \mathcal{H}}{\partial z_i} &= -\int_Q \left(\sum_{j=1}^{N} f(p_j, q)\right)^{-2} \frac{\partial}{\partial z_i} f(p_i, q)^{-1}\, dq + \frac{\partial}{\partial z_i}\phi(p_i) \\
&= -\int_Q \left(\sum_{j=1}^{N} f(p_j, q)\right)^{-2} \left(-f(p_i, q)\frac{(z_i - z_q)}{\sigma_d^2}\right) dq + \frac{\partial}{\partial z_i}\phi(p_i) \\
&= \int_Q \left(\sum_{j=1}^{N} f(p_j, q)\right)^{-2} f(p_i, q)\frac{(z_i - z_q)}{\sigma_d^2} dq + \frac{\partial}{\partial z_i}\phi(p_i) \\
&= \int_Q g(q, p_1, ..., p_N)^2 f(p_i, q)\frac{(z_i - z_q)}{\sigma_d^2} dq + \frac{\partial}{\partial z_i}\phi(p_i) \qquad (4.18)
\end{aligned}$$

4.2.7 Controller Convergence

To prove that the gradient controller (equation 4.14) converges to a critical point of $\mathcal{H}$, we must show [21, 66, 86]:

1. $\mathcal{H}$ must be differentiable;
2. $\frac{\partial \mathcal{H}}{\partial z_i}$ must be locally Lipschitz;
3. $\mathcal{H}$ must have a lower bound;
4. $\mathcal{H}$ must be radially unbounded or the trajectories of the system must be bounded.

While this assures convergence to a critical point of $\mathcal{H}$, small perturbations to the system will cause the gradient controller to converge to a local minimum and not a local maximum or saddle point of the cost function [86]. In Chapter 5 we show that most local minimum are near the global minimum and that the algorithm performs well in practice.

Theorem 1. *The controller $-k\frac{\partial \mathcal{H}}{\partial z_i}$ converges to a critical point of $\mathcal{H}$. In other words as time, t, progresses the output of the controller will go to zero:*

$$\lim_{t \leftarrow \infty} -k\frac{\partial \mathcal{H}}{\partial z_i} = 0 \qquad (4.19)$$

Proof. We show that the objective function satisfies the conditions outlined above. In Section 4.2.6 we determined the gradient of $\mathcal{H}$, satisfying condition 1. $\frac{\partial \mathcal{H}}{\partial z_i}$ has a locally bounded slope, meaning it is locally Lipschitz and satisfies condition 2.

To show that $\mathcal{H}$ is bounded below, to satisfy condition 3, consider the composition of the objective function:

$$\mathcal{H}(p_1, ..., p_N) = \int_Q g(q, p_1, ..., p_N)\ dq + \sum_{i=1}^{N} \phi(p_i) \tag{4.20}$$

The $\sum \phi(p_i)$ term is the sum of a number raised to an even power and is clearly bounded below by zero. Expanding the notation in the integral term we can see:

$$\int_Q g(q, p_1, ..., p_N)\ dq = \int_Q \left(\sum_{j=1}^{N} f(p_j, q) \right)^{-1} dq \tag{4.21}$$

and $f(p_j, q)$ is a Gaussian which is always positive. The integral and sum of positive terms is also positive. Thus, both terms and therefore $\mathcal{H}$ are bounded below by zero, satisfying condition 3.

Unfortunately, $\mathcal{H}$ is not radially unbounded. However, the trajectories of the system are bounded, satisfying condition 4. To see this note that the trajectories of the system are along the z axis. Equation 4.9 defines the $\sum \phi(p_i)$ term. By choosing a sufficiently large β the cost of moving outside of the column overcomes any potential sensing advantage gained by moving outside as the integral term of $\mathcal{H}$ is bounded below.

Thus, we have satisfied all the conditions for controller convergence, proving that the controller $-k\frac{\partial \mathcal{H}}{\partial z_i}$ converges. □

4.2.8 Algorithm Implementation

Algorithm 1 shows the implementation of the decentralized depth controller (Equation 4.10) in pseudo-code. The procedure receives as input the depths of all other nodes in communication range. The procedure requires two functions `F(p_i,x,y,z)` and `FDz(p_i,x,y,z)`. These functions take the sensor location, p_i, and the point,

Algorithm 1 Decentralized Depth Controller Update Step

```
1:  procedure UPDATEDEPTH(p_1 ··· p_N)
2:      integral ← 0
3:      for x = x_min to x_max do
4:          for y = y_min to y_max do
5:              for z = z_min to z_max do
6:                  sum ← 0
7:                  for i = 1 to N do
8:                      sum+ = F(p_i,x,y,z)
9:                  end for
10:                 integral+ = (−1/sum^2) * FDz(p_i,x,y,z)
11:             end for
12:         end for
13:     end for
14:     delta = K * integral
15:     if delta > maxspeed then
16:         delta = maxspeed
17:     end if
18:     if delta < −maxspeed then
19:         delta = −maxspeed
20:     end if
21:     if abs(delta) < mindelta then
22:         return true
23:     end if
24:     changeDepth(delta)
25:     return false
26: end procedure
```

$[x, y, z]$, that we want to cover. The first function, `F(p_i,x,y,z)`, computes the covariance between the sensor location and the point of interest. The second function, `FDz(p_i,x,y,z)`, computes the gradient of the covariance function with respect to z at the same pair of points.

After the procedure computes the numeric integral, it computes the change in the desired depth. This change is bounded by the maximum speed the node can travel. The algorithm then checks if the desired change is less then some threshold. If it is, the algorithm returns `true` to indicate that the algorithm has converged. If it has not converged, the procedure changes the node depth and returns `false`. Note that for this implementation we ignore the impact of $\phi(p_i)$ on the controller. We found that this had little impact on the results.

4.3 Control for Dynamic Phenomena

The decentralized controller positions the sensor nodes to optimize sensing based on a single covariance model. However, many fields of interest have covariances that depend on other sensed or external factors. For instance, in the case of CDOM (see Section 5.2) the covariance depends on the tide as it changes throughout the day. In this section, we modify Algorithm 1 to handle systems with changing covariance.

4.3.1 Periodic Update of Covariance

The decentralized depth control algorithm converges to a local minimum configuration. In our experimental work(see Section 5.5), we show that convergence occurs within 20 minutes for the constant covariance functions tested. In practice, once the algorithm converges, the positions should be maintained for a period of time to collect data. However, after a sufficient amount of time has passed the covariance should be updated and the algorithm rerun to update the positions of the nodes. The choice of how long to stay in one location impacts the energy usage, and therefore the longevity of the system. Staying less time enables finer grained tracking of dynamic phenomena.

Algorithm 2 shows the high-level algorithm that performs the depth control and manages and updates the covariance function based on sensed or known parameters. The algorithm continuously calls `updateDepth(`$p_1 \cdots p_N$`)`, in line 14, until the function returns indicating that it has converged.

Once the decentralized controller converges the algorithm waits, in line 7, until a period of time has elapsed (dependent on the desired deployment time) before updating the covariance function to a new function based on the current time and sensor readings. The function
`updateCovarianceFunction(time,sensors)` may update the covariance based on a lookup table, model calculation, measurements, or other methods.

After the covariance function has been updated, the variable *converged* is set to false. This will cause the controller to run during the next iteration of the loop.

Algorithm 2 Dynamic Decentralized Depth Controller

```
1:  procedure DEPTHCONTROL
2:      converged ← false
3:      while true do
4:          if converged then
5:              if converged long enough then
6:                  updateCovarianceFunction(time,sensors)
7:                                              ▷ Update covariance function F
8:                  converged ← false
9:                  if shouldScanWaterColumn then
10:                     scanWaterColumn()
11:                 end if
12:             end if
13:         else
14:             converged ← updateDepth(p_1 ··· p_N)
15:         end if
16:     end while
17: end procedure
```

Before this, in line 9, a variable *shouldScanWaterColumn* is checked. This externally set variable indicates if the sensor should perform a full water column scan before continuing onto its next sensing location.

Performing a full column scan (moving from sea floor to surface) is useful to obtain a wider range of sensed locations and adds variability to the locations the nodes will next converge to. The variability is due to the fact that if the covariance function only changes by a small amount, it is likely that the sensors will only move by a small amount. This is desirable from an energy perspective, however, more variation in the positions may be of interest to the scientists. In addition, this will lower the chances that the nodes get stuck in a poor local minimum of the objective function for the whole experiment.

4.3.2 Continuous Update of Covariance

Periodic updates of the covariance function is useful, however, this only positions the sensor nodes at the optimal locations for a small period of time as the optimal locations changes as the covariance changes. In this section we discuss the impact of a continually variable covariance function on the algorithm and convergence.

Consider, for example, the effect of tide on the optimal positions for the sensors when measuring salinity at the mouth of a river that empties into the ocean. At low tide, the depth may be just 2m deep, whereas at high tide it could be 3 or 4m. The positions of the sensors should be updated continuously to account for the changing depth of the water column.

Algorithm 1 is easily updated to reflect dynamically changing covariance by updating the functions `F` and `FDz`. These functions must be updated to use the most up-to-date covariance function, instead of a static covariance function. In Section 4.2.7 we proved the decentralized gradient controller converges to a local minimum. However, this proof assumes that the covariance function is constant in time and does not change. Thus, the update to the algorithm to allow for the continuous update of the covariance may effect the convergence of the algorithm.

Since, dynamics in underwater systems tend to be slow compared to the motion of the sensor nodes. As such, as long as the covariance is changing slow enough, it can be considered static with respect to the dynamics of the sensor nodes. With this assumption, the convergence proof in Section 4.2.7 will hold. Alternatively, we can run the updates in stages and rely on static convergence at the end of each phase.

Another way to analyze Algorithm 2 is to consider the effect of a small change in the covariance function on the final positions of the sensor nodes. If small changes in the covariance result in small changes in the final positions, then the controller does not need to move the nodes very far to account for the changes to the covariance and is thus likely stabilize and converge. In Section 5.4.3 we show in simulation that this is the case–small changes to the covariance function result in small changes in the positions of the nodes.

Continuously updating the covariance and adjusting the positions, however, is not an energy efficient method. In Section 5.4.6 we analyze the energy usage of the system and find that the deployment time is drastically reduced if the acoustic modem and depth adjustment system is run continuously. This analysis suggest that periodic updates of the covariance function is the best tradeoff between optimizing sensing locations and sensor network life expectancy.

Chapter 5

Decentralized Depth Adjustment: Simulations and Experiments

5.1 Introduction

In Chapter 4 we introduced the decentralized depth control algorithm that optimizes the depths of the sensor network nodes for sensing. In this chapter we explore the performance of the algorithm in simulation and on our AquaNode underwater sensor network platform (analyzed in detail in Chapter 3).

Scientists are starting to better understand ocean processes and are developing improved ocean and coastal models. One example is the modeling of Boston Harbor (shown in Figure 5-1). Currently, all of Massachusetts Bay is covered by only three NOAA surface buoys to measure tide, temperature, wind, etc. [4]. These sensors feed models of the bay that help scientists understand the dynamics of the Massachusetts Bay system. Adding sensors to the bay, especially nodes able to measure at varying depths, will greatly improve the modeling and understanding of this coastal water system. However, to fully understand the system we must also understand the impact of fresh water flows into the harbor. One such river is the Neponset River.

We are interested in detecting and measuring the tidal front in the Neponset River to feed a numerical model developed by researchers at UMass Boston [57]. Understanding the tidal front enables scientists to better determine the spread and

impact of organic and inorganic matter carried by the river. The model used is developed for Boston Harbor, based on the Estuarine, Coastal, Ocean Model (ECOM-si) with Mellor and Yamada 2.5 turbulent closure for the vertical mixing [19, 18, 91]. The model domain covers the entire Boston Harbor (over 500km^2) and a portion of the Massachusetts Bay with a grid resolution around 70m, as shown in Figure 5-1. The model is currently forced by surface winds and heat fluxes derived from measurements at NOAA buoy 44013 in western Massachusetts Bay, and freshwater discharges at the USGS gauges, and boundary forcing (tides, currents, temperature and salinity) derived from the model output of the Massachusetts Bay hydrodynamic model [57]. The model is able to capture the general dynamic processes including tidal cycle, seasonal development of stratification, and wind- and river-driven circulation, but adding information on the flow from the Neponset river will greatly enhance the model.

Our Oceanographer collaborators at UMass Boston are particularly interested in measuring Chromophoric dissolved organic matter (CDOM) in the Neponset river. CDOM is the optically active component of the total dissolved organic matter in the oceans. In estuaries, CDOM is mostly produced in fringing marshes and exported through freshwater discharges and hence it is closely tied to salinity with nearly linear salinity-CDOM mixing curves [44]. Additional sources (sinks) from mid-estuary production (removal) will lead to the mixing curve concave upward (downward). An understanding of CDOM dynamics in coastal waters and of its resulting distribution is important for remote sensing and for estimating light penetration in Boston Harbor and the ocean [22, 16, 17]. Improved understanding of CDOM dynamics requires sensor networks measuring the Neponset River.

To ensure that a reasonable dataset is collected we can use our numerical model to inform a decentralized control algorithm with depth adjustment. Using this information allows better placement for computing maximally detailed volumetric models. In this chapter we simulate positioning sensors along the Neponset river and adjusting their depths to optimize the sensing of CDOM as it is discharged into Boston harbor.

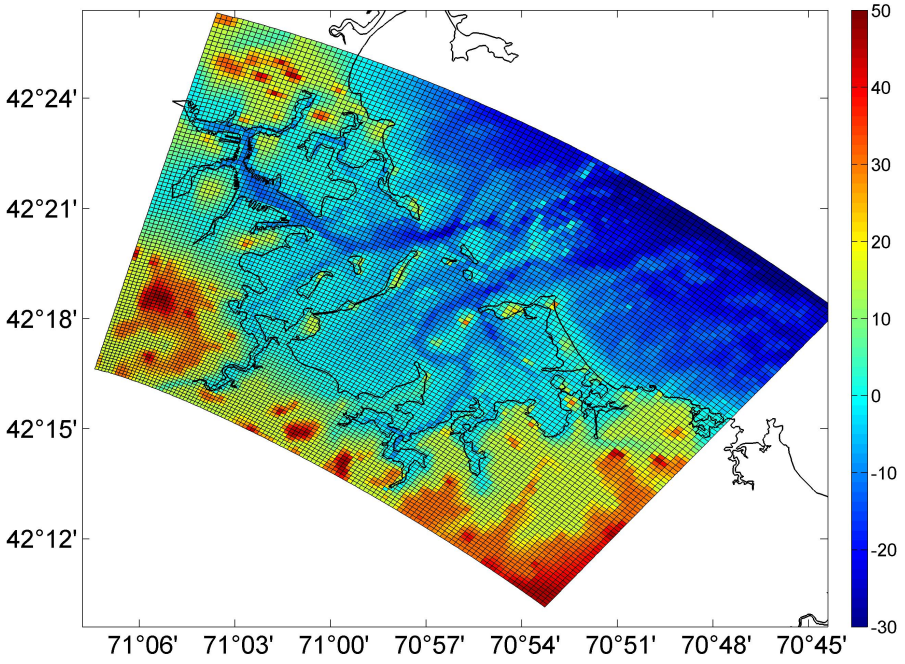
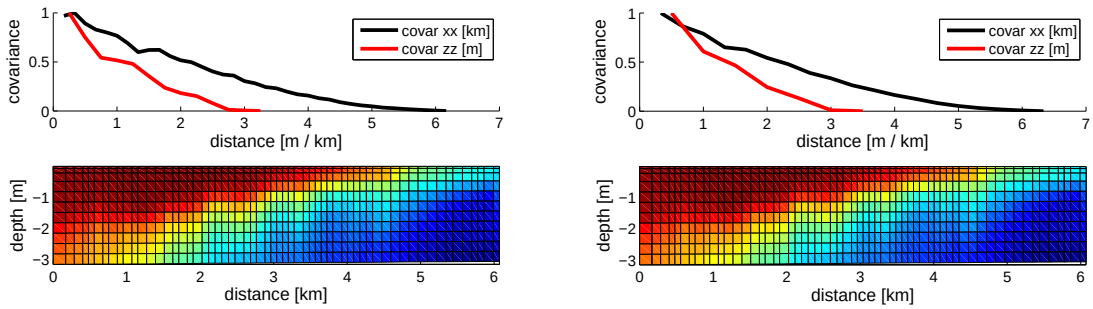


Figure 5-1: Boston Harbor modeling area bathymetry, courtesy Mingshun Jiang (UMass Boston).

To accurately position the sensor nodes in the Neponset river, we develop a covariance model based on a physics-driven hydrodynamic model and implement this on the AquaNode platform. The techniques used to develop the covariance model are general and can be applied to other systems with model or real data.

Section 5.2 discusses the numeric covariance model, while Section 5.3 discusses the practical considerations that must be taken into account before implementing the generic algorithm (discussed in Chapter 4) on a hardware system. We then analyze the algorithm parameters and setup, compare it to other techniques and metrics, and provide an energy analysis in Section 5.4.

Finally, in Section 5.5, we implement the algorithm on the AquaNode platform and perform experiments in the lab, the pool, and a river. We analyze the performance of the algorithm in these setups where communication is far from ideal and show that the algorithm performs well and converges. In these experiments we implemented the Gaussian covariance, model-based covariance, and tested the later in the under dynamic conditions using the periodic covariance update algorithm.



(a) (b)

Figure 5-2: Bottom: Model of the CDOM concentration in the Neponset river when tide caused a river depth of on average 2m (a) and 3m (b). Top: The corresponding numerically computed covariance.

5.2 Covariance Model

In Section 4.2 we developed the decentralized depth controller algorithm using a multivariate Gaussian as the covariance function. However, more generally, if we have real information about the system, we can numerically compute the covariance.

Figure 5-2 (bottom) shows the concentration of CDOM in the Neponset river, which feeds into Boston Harbor at two different tide levels. Figure 5-2 (top) shows the numerically computed covariance along the length of the river and along the depth. We numerically computed the covariance along the length of the river by examining the each pair of points at distance k apart and taking the sample covariance, which is $Cov(\text{pts } k \text{ apart}) = \sum(ab)/N - mean(a) * mean(b)$ for the N points a and b that are k distance apart. We compute the covariance along the depth of the river in a similar manner. We find a high correlation along the length of the river over kilometers and high variance over just a few meters in depth as shown in Figure 5-2.

We fit a Gaussian basis function to the numeric covariance curve. To do this we use Matlab's `newrb` function. Figure 5-3 shows the result of the basis function fit. The error in the fit depends on the number of Gaussians used. For this plot with 6 elements the error is 1.88%, whereas using 10 elements gives an error of 0.54%. Using a basis function gives us a compact representation of the covariance function that a sensor node can easily store and compute.

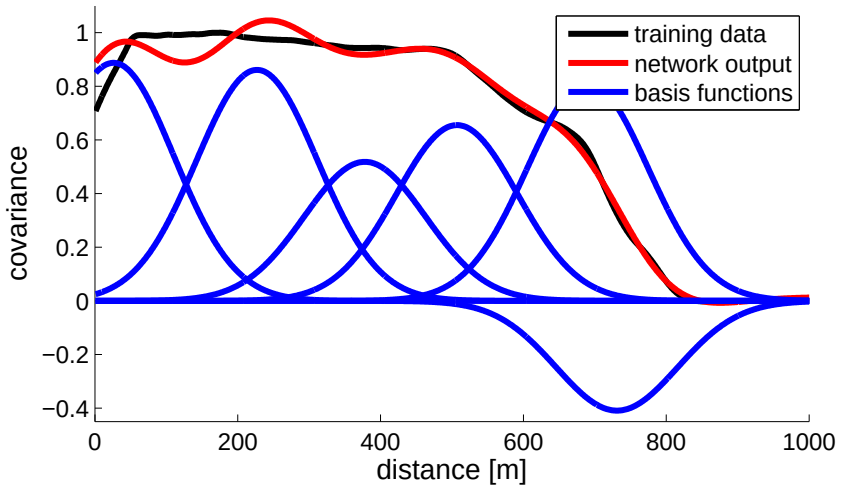


Figure 5-3: The Gaussian basis function elements for a fit with 6 basis functions.

An advantage of using a Gaussian basis function is that it allows online updates of the covariance function based on the sensed data. Nodes can share and learn the parameters of the basis function using a number of existing techniques for fitting basis functions to unknown data [76].

Figure 5-2 shows two different tide level in the river system. Different covariance functions, based on the tide, enable dynamic repositioning of the sensor network to adapt to changing conditions. In Section 5.5.4 we examine the results of experiments where the covariance function is periodically changed.

5.3 Practical Considerations

We must examine a number of practical considerations before implementing this controller in simulation and on the underwater sensor node hardware. These are:

- The controller continuously integrated over an area Q; practically the region must discretized for numeric integration;

- The acoustic communication has limited bandwidth and messages are transmitted infrequently;

- The controller assumes knowledge of the location and depth of every node; in practice a node only knows the location and depth of neighbors.

In this section we examine each of these issues and argue that the practical implications are minimal for our system. We back up these assertions with experimental results in the following sections.

5.3.1 Discretization and Run Time

The controller requires integrating the cost function over the area Q. Processors cannot integrate continuously so the regions are broken into discrete sections and these sections are summed. There are two factors which affect how the region is discretized. The first is the desired sensing accuracy and the second is computational complexity. If the discretization is too coarse the sensing accuracy may deteriorate as the algorithm will not differentiate between different configurations. Alternatively, if the discretization is too fine, the computational time may become too long.

More specifically, we can analyze the run time of the distributed controller in the discretized setting. Each node must compute its control input, which is the negative gradient of the objective function $\mathcal{H}$. This will be:

$$\frac{\partial \mathcal{H}}{\partial z_i} = -\sum_{x \in Q} \sum_{y \in Q} \sum_{z \in Q} \left(g(q, p_1, ...p_N)^2 f(p_i, q) \frac{z_i - z_q}{\sigma_d^2} \right)$$

The triple summation has a computational cost related to the volume of the area, while $g(q, p_1, ..., p_N)$ has a computational complexity proportional to the number of sensors in the system. Let $|X|$, $|Y|$ and $|Z|$ be the number of steps in the x, y and z direction respectively. And as before, let N be the number of sensor nodes in the system. Then the computational complexity is:

$$O(|X| \cdot |Y| \cdot |Z| \cdot N).$$

In practice, the number of steps in the numerical integration of the volume will dominate the runtime. Thus, we must carefully choose the resolution to balance

computational resources and accuracy. We explore the effect of changing the size of the grid in Section 5.4.2 and find that computationally feasible grid sizes yield good results.

5.3.2 Communication Bandwidth

Algorithm 1 assumes that the control input to the sensor nodes are occurring continuously and with completely up-to-date knowledge of the whole system. In practice, we can only compute the control input at fixed intervals due to the computational costs. These intervals can be fast relative to the dynamics of the network mitigating problems regarding time discretation, however, we face another more important problem.

The acoustic modems (described in Chapter 3) use a TDMA scheme to schedule communications and each message is limited to 11 bytes. The depth can easily be encoded within this byte limit, however, each node is only able to share its depth every $4N$ seconds as each TDMA slot is 4 seconds long. Thus, the algorithm must contend with old information when making the control decisions. One approach to address this issue is to only allow the nodes to move for a short time every $4N$ seconds. This would give all nodes a relatively consistent snapshot of the network configuration. However, some packets will be lost, implying the period between motions would have to be longer to get consistent configurations. This approach drastically slows down the operation of the controller.

We use a controller update time of about $4N$ seconds. Despite packet loss we have found in our experiments that it performs well. The reason for this is because the minima of our cost function tend to be "deep" and stable. We expect configurations where nodes alternate between an up and down configuration, so if a node starts going up it is likely that it's final configuration will be nearly fully up. Thus, in practice the controller can run at a reasonable rate and integrate new information as it becomes available.

5.3.3 Local Knowledge

The decentralized controller assumes that it has information about the depths of all nodes in the system. In practice each node only knows the depths of its neighbors. This is because the cost of forwarding depths over multiple hops in the network would quickly saturate the limited acoustic bandwidth. Fortunately, the effect of far away sensors is minimal as the covariance function decays rapidly with distance so the algorithm can ignore the effect of sensors that it cannot hear. Similarly, in Section 5.4.2 we show that the region, Q, of integration can be reduced. This is because our local actions have practically no effect on regions far away.

5.4 Simulation and Analysis

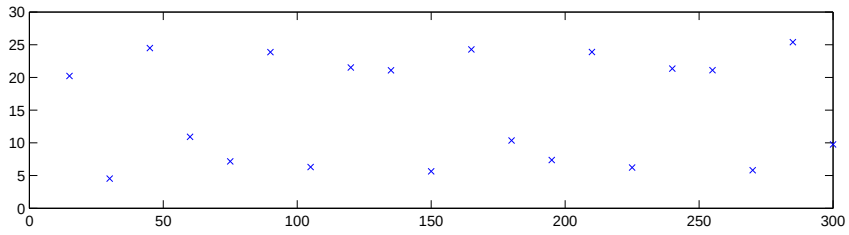
In this section we discuss the results of simulation experiments. Several practical considerations arise in implementing this controller on real hardware. In this section we discuss parameter sensitivity, positioning sensitivity, comparison to other methods, data reconstruction, and analyze the energy usage.

5.4.1 Simulation Setup

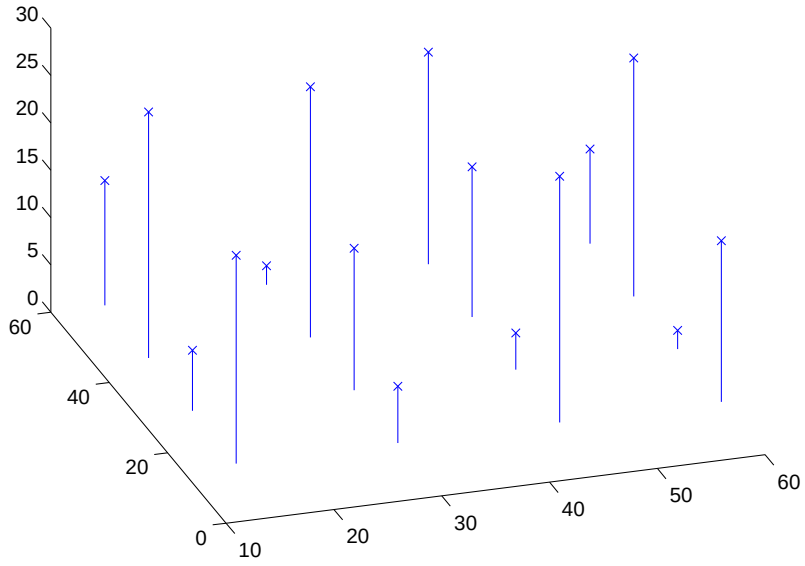
We implemented the decentralized depth controller in Matlab to test the performance of the algorithm.

In these experiments, unless otherwise noted, we use the base Gaussian covariance function described in Section 4.2.5, in later sections we examine the CDOM covariance discussed in Section 5.2. We use a "k" value of 0.001, capped with a maximum speed of $\pm 2\ m/s$, and having $\sigma_s = 10$ and $\sigma_d = 4$. Each node performs twenty iterations of the controller. The nodes are placed in a line spaced 15m apart from each other and are in "water" of 30m depth. A 1 meter grid is used to integrate over for all operations.

Figure 5-4(a) shows the results after running the decentralized controller on a network of 20 nodes. Each iteration of the algorithm took 8s per node with convergence occurring after 6 iterations.



(a)



(b)

Figure 5-4: The final positions after the distributed controller converges for a (a)2D and (b)3D setup.

Figure 5-4(b) shows the result of an experiment with 16 nodes arranged in a 4-by-4 grid with 15m spacing in 30m depth. Each iteration of the algorithm took 2min per node with convergence occurring after 7 iterations.

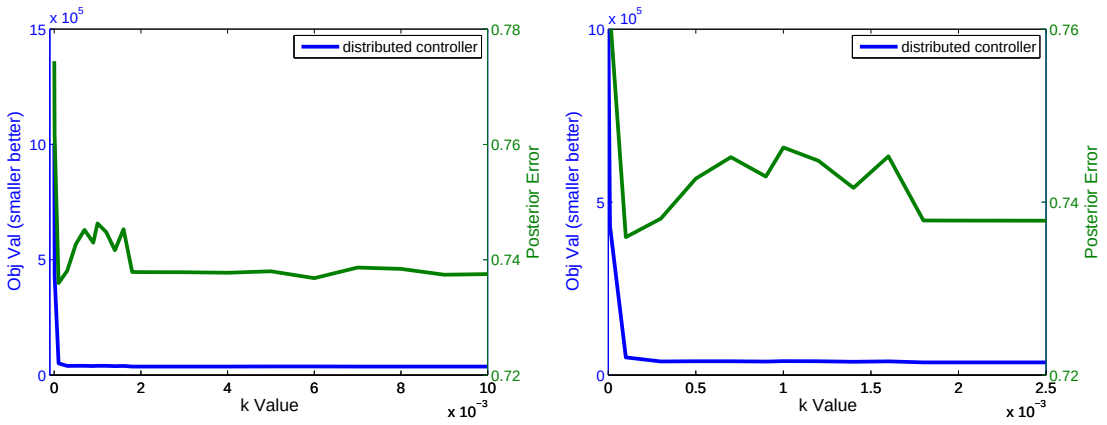


Figure 5-5: The objective value found when different "k" values are used in a system with 20 nodes. (a) Shows the full range of values explored. (b) Shows a zoomed in section of (a).

5.4.2 Parameter Sensitivity

We analyze k, neighborhood size, and grid size.

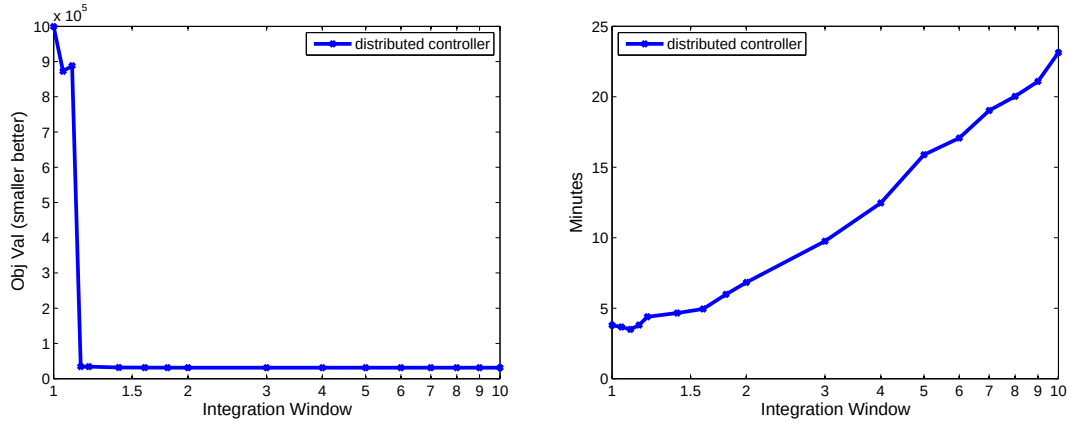
Changing k

Figure 5-5 show the affect on the objective value, $\mathcal{H}$, on changing the k value for a system with 20 nodes. Recall that each node moves according to:

$$-k\frac{\partial \mathcal{H}}{\partial z_i} \tag{5.1}$$

Increasing the value of k causes the nodes to move down the gradient of $\mathcal{H}$ more quickly. Values that are too large can lead to oscillations around the final configuration or lead to instabilities in the system. If the value of k is too small, the system may not move fast enough to converge within a reasonable number of steps.

Figure 5-5(a) shows the range of k's explored, while Figure 5-5(b) zooms in on k values less then 0.003. Using a k value of less than 0.0005 yields very poor results since the system does not converge within the 20 iterations of the algorithm that we allow. However, values larger than this perform well.



(a) (b)

Figure 5-6: The (a) objective value and (b) runtime for a 15 node network when changing the size of the neighborhood over which the integration occurs.

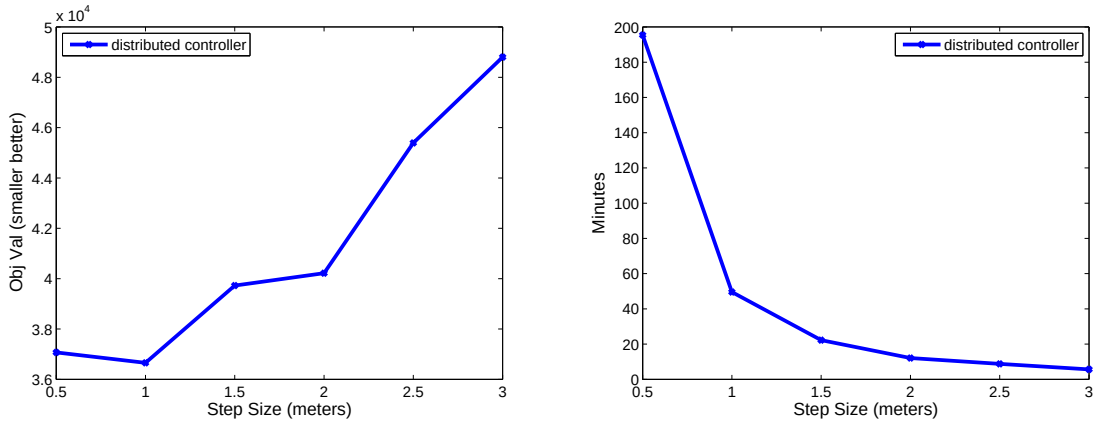
In Figure 5-5(b) we see that k values between 0.002 and 0.003 perform well. In general the exact value of k has an impact on the results, however, it tends to be minimal as long as a sufficiently large value of k is used.

One reason that large values of k yield good results is that the nodes are limited in how fast they can move. Recall that in our simulations we limit the maximum speed of the nodes to 2m/s. Thus, even with large values of k the system tends to converge and not oscillate.

Changing Neighborhood Size

We examine the effect of changing the neighborhood size. The neighborhood size is the size over which each node integrates when computing the numeric integral (Algorithm 1, line 10). The decentralized controller assumes that it has information about the depths of all nodes in the system. In practice we can only know the depths of our neighbor nodes due to poor acoustic communication. The covariance function decays rapidly with distance, reducing the affect of far away sensors, allowing nodes to ignore the sensors they cannot hear.

Figure 5-6 shows the results of changing the size of the neighborhood over which we integrate. For this simulation the nodes were placed 15m apart. The neighborhood



(a) (b)

Figure 5-7: Changing the grid size. (a) Objective value and (b) total search time as the step size changes.

size varied from ±15m to ±150m. As can be seen from Figure 5-6 (a), using a neighborhood of just 15m results in very poor performance. However, a slight increase in neighborhood size drastically increases performance. This indicates that near-neighbors have the largest impact. Thus, the neighborhood size should be chosen to include all one-hop neighbors. The actual number of neighbors that need to be heard will depend on the node spacing and how quickly the covariance decays. Figure 5-6 (b) shows that the total runtime required for the algorithm increases in a linear fashion as the neighborhood size increases. This experiment verifies the intuition that nodes only need to hear direct neighbors to have good performance.

Changing Grid Size

We examine the impact of changing the size of the grid over which we numerically integrate. In simulation we uniformly change the step size in the x and z axes. Using a large step size reduces the computations needed to perform the decentralized controller, however, if the step size is too large, important regions may be overlooked, causing a degradation in performance.

Analyzing Algorithm 1 we see a runtime of $O(|X|\cdot|Y|\cdot|Z|\cdot N)$, where $|X|$ represents the number of steps in the grid along the x-axis and N is the number of sensor nodes

in the system. As shown in Figure 5-7 (b) the runtime decreases in a quadratic fashion as the step size increases. The quadratic results from the 2D simulation; in the 3D case the runtime would decrease cubically as the step size increases.

Figure 5-7 (a) shows the impact of changing the grid size on the objective function. As step size increases the objective function does as well. Thus, a grid size of 1m seems reasonable as this is the minimum and results in a good runtime. If the spacing of the nodes was closer a finer grid may be needed and similarly if they are spaced further apart a larger grid could be used. In our experience having between ten and twenty steps between each pair of nodes yields a good balance between runtime and algorithm performance.

5.4.3 Positioning Sensitivity

We analyze start configuration, random placement error, and changing the covariance function.

Changing Start Configurations

We examine how close the decentralized controller comes to obtaining the global minimum of the system. To do this we ran a number of simulations starting the nodes at different depths and examining the results. Figure 5-8 shows the various starting and ending configurations, and (c) shows the final objective value and posterior error for these trials.

We tested a number of starting configurations. In all of these experiments the final configuration ended up with nodes roughly alternating. However, some local minimums occur that result in a worse objective function value. In particular, the configuration in Figure 5-9.9, which alternated two down and two up resulted in a similar final configuration. As can be seen in Figure 5-10 this configuration yielded the worst objective value and posterior error of all the trials.

We can contrast this with the configuration in Figure 5-9.6, which resulted in the best objective value. The remainder of the configurations fell somewhere in between

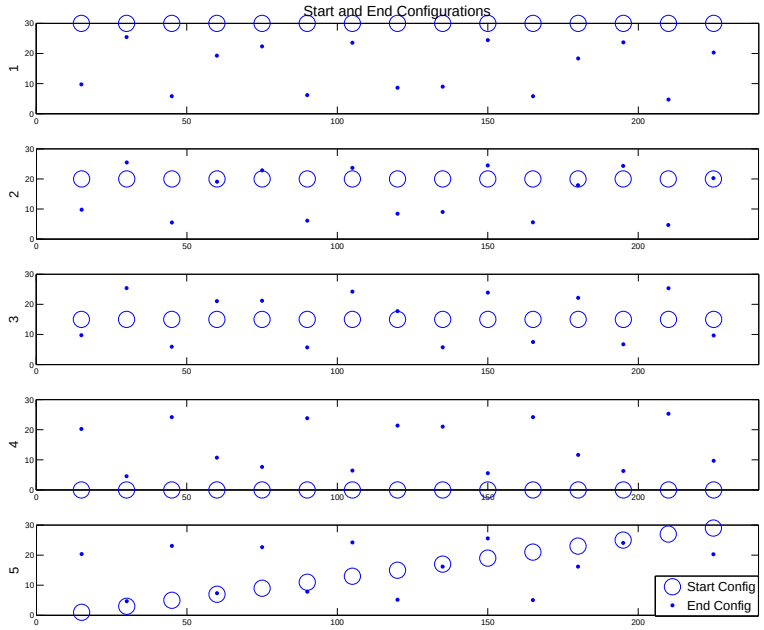


Figure 5-8: The results of the running the depth adjustment algorithm on various node start positions, configurations 1–5.

these two. Some configurations demonstrate that a local minimum can occur that is hard to overcome. Fortunately, while this occurs occasionally, it is not often and when it does occur the system still obtains fairly good results.

Random Placement Error

As it is impossible to perfectly place nodes in a real-world setup, we examine the effects of random variation in the x-axis placement of the nodes. The nodes were deployed starting in the ideal depth configuration. Figure 5-11(a) demonstrates the results of 100 trials with $\pm$6m random error added to the ideal node positions in the x-axis. We then run the depth adjustment algorithm on the mispositioned nodes, which improves their overall depth positioning for sensing.

Figure 5-11(b) outlines this for errors in x-axis placement ranging from 2m to 16m. Each point is the average of 100 random trials. Again the depth adjustment algorithm improves the overall position. This experiment shows that the decentralized depth

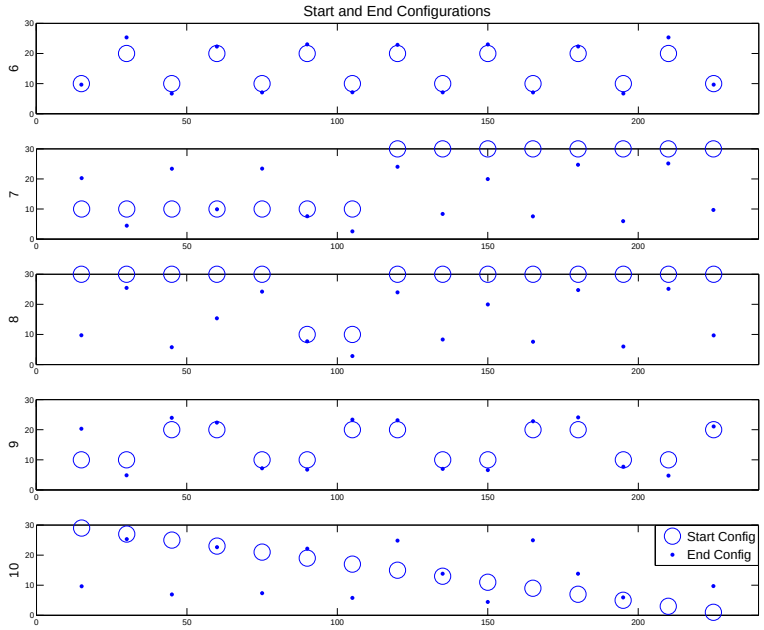


Figure 5-9: The results of the running the depth adjustment algorithm on various node start positions, configurations 6–10.

adjustment algorithm can improve the overall sensing even if the start depths (z-axis) are ideal and x-axis placement is not.

Changing Covariance

In this section we examine the effect of changing the covariance function on the final positions of the sensor nodes. Recall, from Section 4.2.5, the Gaussian sensing function:

$$f(p_i, q) = Ae^{-\left(\frac{(x_i-x_q)^2+(y_i-y_q)^2}{2\sigma_s^2}+\frac{(z_i-z_q)^2}{2\sigma_d^2}\right)} \tag{5.2}$$

Figure 5.1 compares the resultant positions of the sensor nodes when varying σ_s (the surface distance variance) and σ_d (the depth variance). First, the controller was run with $\sigma_d = 4$ and $\sigma_s = 10$ on a 20 node network in 30m of water. The sigmas were then updated to the values in the table and the algorithm was rerun. The resultant average absolute difference and percent difference in the final positions of the nodes compared to the $\sigma_d = 4$ and $\sigma_s = 10$ run is shown.

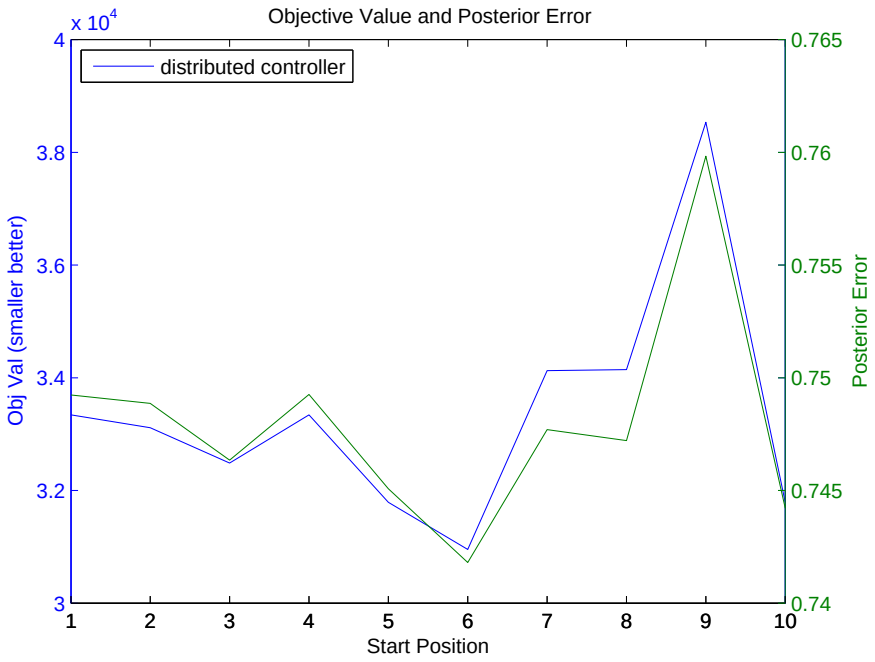
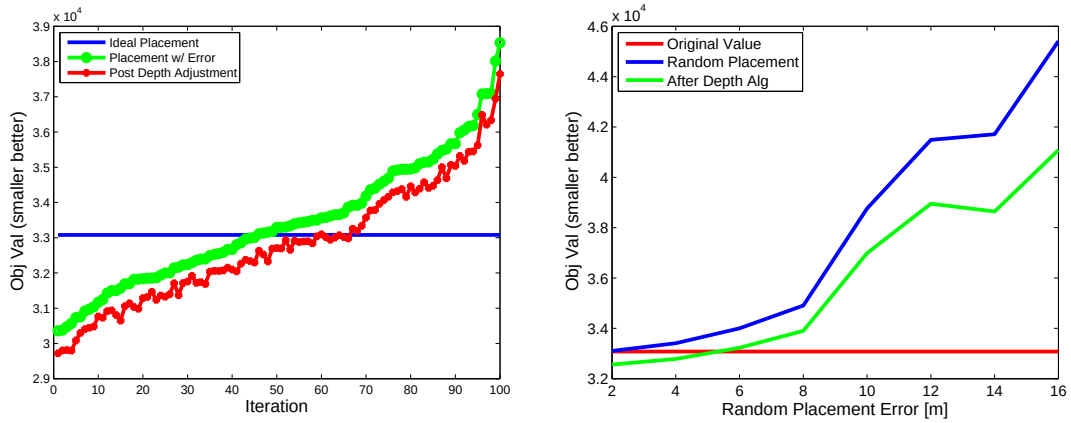


Figure 5-10: The objective value after running the depth adjustment algorithm on various node start positions shown in Figure 5-8 and 5-9. Also shown is another metric, the posterior error, which is discussed in Section 5.4.5.

This table shows that small changes in the covariance function results in small final position changes. This implies that the controller is stable and convergent when the covariance function is slowly changing. This enables the use of the controller under dynamic conditions as discussed in Section 4.3.

change in σ_s	change in σ_d	Mean Absolute Difference (m)	Percent Difference
1	1	0.1651	0.5%
0	1	0.3563	1.2%
1	0	0.4705	1.6%
-1	0	0.6580	2.2%
-2	0	0.9005	3.0%
-1	-1	2.6751	8.9%

Table 5.1: Mean absolute difference in node positions after changing the variance.



(a) (b)

Figure 5-11: Nodes deployed with random error in x-axis placement. (a) Plot of 100 runs with 6m error. (b) Average over many runs and positions.

5.4.4 Data Reconstruction

The ultimate goal of placing sensors in the water is reconstructing the complete data field, not just the points where sensors exist. The distributed depth adjustment algorithm places sensors in locations to maximize the utility of the sensed value for doing this type of reconstruction. In this section we show the results of simulations in reconstructing data fields given point measurements at sensor node locations.

Figure 5-12 shows the results of reconstructing a field given three manually configured sensor placements and the depth adjustment algorithm. At top in Figure 5-12 we show the actual field we attempt to recover. This field has similar covariance properties to the Gaussian covariance. The manually chosen configurations were: (1) sensors placed alternatively at the top and bottom of the water column; (2) sensors placed in the middle of the water column; (3) sensors placed a quarter of the way off the top and bottom. Finally, at the bottom is the positioning based on the depth adjustment algorithm. While visually similar to the manually chosen configurations, it slightly improves upon the positions to better cover the region.

To quantify this we use the sum of squared error metric, comparing the actual model data to the recovered using Matlab's 'v4' version of `griddata`. The actual Matlab code used was:

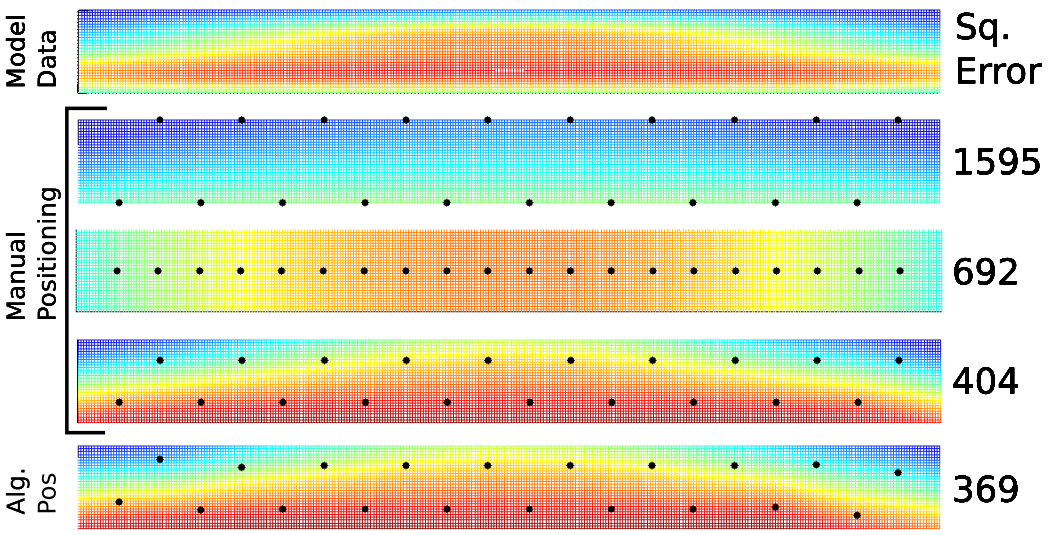


Figure 5-12: At top is the model data (d6) followed by three manual node positionings and, at bottom, the results of the decentralized depth controller. Black dots indicate node positions, at right is the sum of squared error for each set of recovered data.

```
x = nodeLocations(:,1);
y = nodeLocations(:,3);
z = getFuncVals((x-xmin)./(xmax-xmin),
                (y-zmin)./(zmax-zmin));

[XI,YI] = meshgrid(xmin:xstep:xmax,
                   zmin:zstep:zmax);
ZI = griddata(x,y,z,XI,YI,'v4');
ZIgt = getFuncVals((XI-xmin)./(xmax-xmin),
                   (YI-zmin)./(zmax-zmin));

sqerr = sum(sum((ZI-ZIgt).^2));
```

The code starts by obtaining the actual value of the sensory function at the node locations `x,y` and places them in `z`. A grid is then overlayed on the area and the field is reconstructed using the 'v4' interpolation engine of Matlab's `griddata`. This method is used as it reconstructs data outside of the convex hull of the sensor node locations which is desired in our setup. Finally, the results of the reconstructed

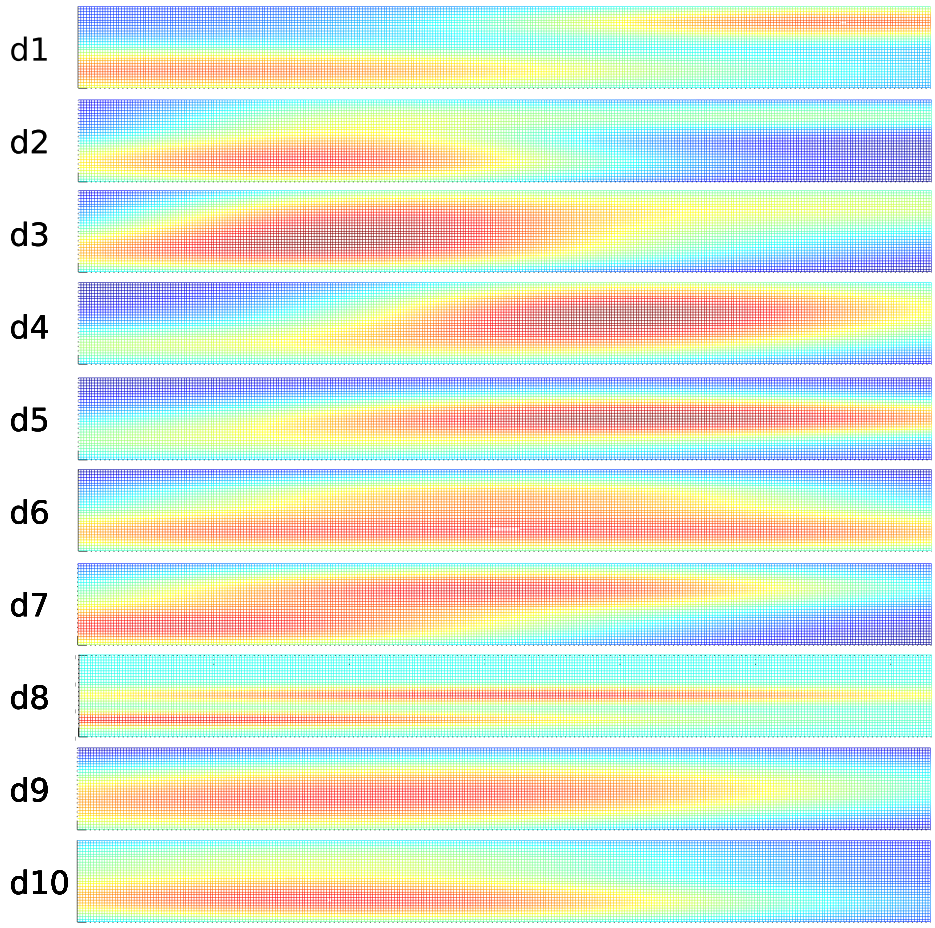


Figure 5-13: The 10 data fields.

field are compared to the ground truth from the known field. The sum of squared error values are shown in the right of Figure 5-12. The dynamic depth adjustment algorithm outperforms the three manually chosen configurations.

Figure 5-13 shows 10 data fields that were semi-randomly used to test the data reconstruction (field `d6` is the field used in Figure 5-12). Table 5.2 shows the sum of squared error statistics for the three manually picked configurations and the decentralized algorithm for all ten configurations. Note that the decentralized controller performs better then the other positions in all but three cases. In these cases (`d5`, `d8`, and `d9`) the midupdown configuration (sensors placed evenly one quarter of the way off the top and bottom) performed best. However, the decentralized controller performed nearly as well in all of these cases.

	d1	d2	d3	d4	d5	d6	d7	d8	d9	d10
updown	491	1030	1769	2959	5877	1595	1159	1259	1933	1741
center	855	696	799	1660	6672	692	506	3863	1236	609
midupdown	428	497	444	766	**1979**	404	384	**1021**	**479**	681
ours	**382**	**457**	**426**	**748**	2100	**369**	**346**	1078	484	**634**

Table 5.2: The sum of the squared error for each node configuration for recovering data from configurations d1-d10 in Figure 5-13. Bold indicates minimum value for that configuration.

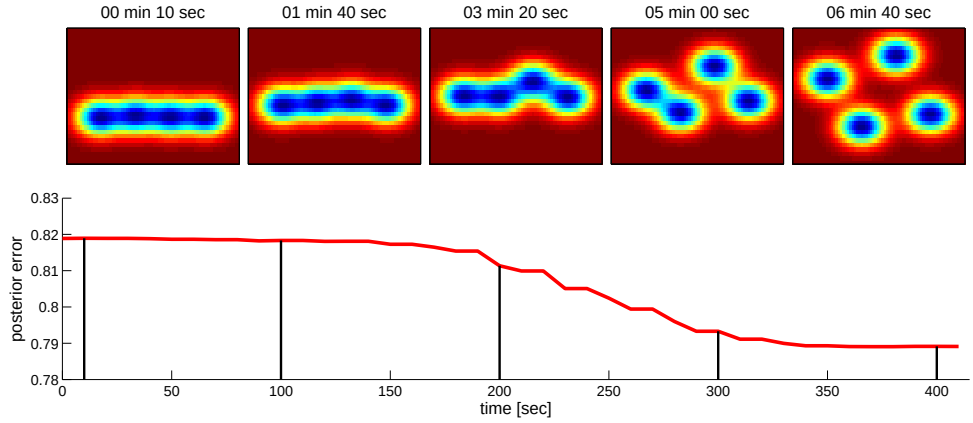


Figure 5-14: At bottom, the posterior error as the experiment progresses.

While midupdown and the decentralized control algorithm perform similarly in most cases there are a number of important distinctions. The midupdown configuration is a static configuration. If conditions change the nodes will not readjust themselves to improve sensing quality. In addition, if the inter-node spacing is not equal midupdown does not perform as well as a similarly spaced deployment using the decentralized controller. Finally, any error in the bottom placement of the nodes will likely result in a worse data reconstruction as discussed in Section 5.4.3.

5.4.5 Comparison to Other Methods

We compare the decentralized depth adjustment algorithm to posterior error methods and Matlab's `fminsearch`.

Posterior Error

A common metric for defining how an area is covered by sensors is to examine the posterior error of the system [47]. Calculating the posterior error requires that the system can be modeled as a Gaussian process. This is a fairly general model and valid in many setups. The posterior error of a point can be calculated as:

$$\sigma^2_{q|P} = Cov(q,q) - \Sigma_{q,P} \cdot \Sigma^{-1}_{P,P} \cdot \Sigma_{P,q} \tag{5.3}$$

The vector $\Sigma_{q,P}$ is the vector of covariances between q and the sensor node positions $P = \{p_1, ..., p_N\}$. The vector $\Sigma_{P,q}$ is $\Sigma_{q,P}$ transposed. The matrix $\Sigma_{P,P}$ is the covariance matrix for the sensor node positions. The values of $\Sigma_{P,P}$ are $\Sigma_{p_i,p_j} = Cov(p_i, p_j)$ for each entry (i,j).

This computation, however, requires an inversion of the full covariance matrix. This is impractical on real sensor network hardware that has limited computation and memory. As such, we cannot calculate the posterior error on the sensor nodes, but we can calculate it as a metric to evaluate our own objective function and to compare different sensing configurations. As shown in Figure 5-10, the posterior error and the objective function track each other, showing that our metric has similar properties to that of the posterior error metric. Figure 5-14 (bottom) shows a plot of a run of the decentralized depth controller and plots the normalized sum of the posterior error at all points. Figure 5-14 (top) shows snapshots of the plots of the posterior error as the algorithm progresses. The algorithm performs well under this metric as well as the objective function metric.

Distributed Controller Versus "fminsearch"

To analyze the performance of our distributed depth adjustment algorithm we compared it to Matlab's standard unconstrained non-linear minimizer, `fminsearch`. This minimizer adjusts the depths of each of the nodes until it minimizes the objective function. `fminsearch` is completely centralized and must know the positions of all nodes. In our simulations we find that `fminsearch` has similar results in terms of

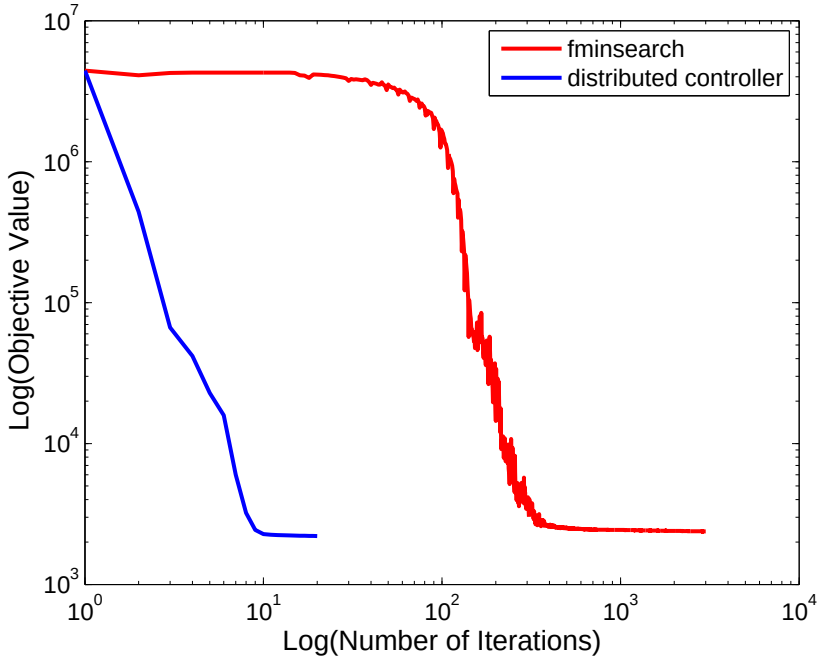
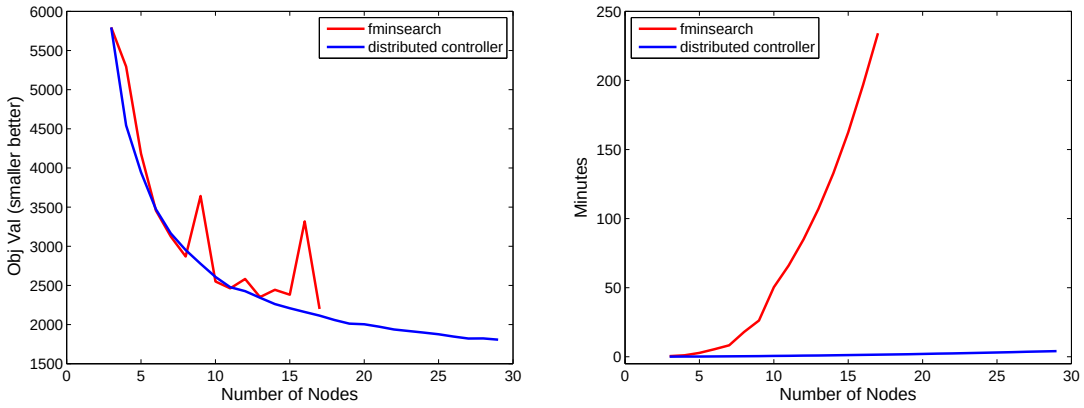


Figure 5-15: The objective value versus the number of iterations for the decentralized controller and fminsearch.

optimizing the objective function for a relatively small number of nodes. However, it has poor runtime and is thus limited to running on systems with a small number of nodes. Matlab's `fminsearch` is at a disadvantage as it does not have any information about the covariance function. Instead, it must rely on trial and error in positioning to optimize placement.

Figure 5-15 describes how the objective function decreases with each iteration of `fminsearch` and each iteration of our distributed controller plotted on a log-log plot. Both algorithms achieve similar objective function values (`fminsearch`: 35707, distributed controller: 33114). However, the distributed controller did so in under 10 iterations requiring 18min computation time, while `fminsearch` required nearly 200 iterations and 162min.

Figure 5-16(a) shows the final objective value achieved for both algorithms for configurations of 3 to 29 nodes. The distributed controller performed 20 iterations for all configurations. The results show that for under 17 nodes `fminsearch` and the distributed controller achieve similar results; above 17 nodes the runtime of `fminsearch`



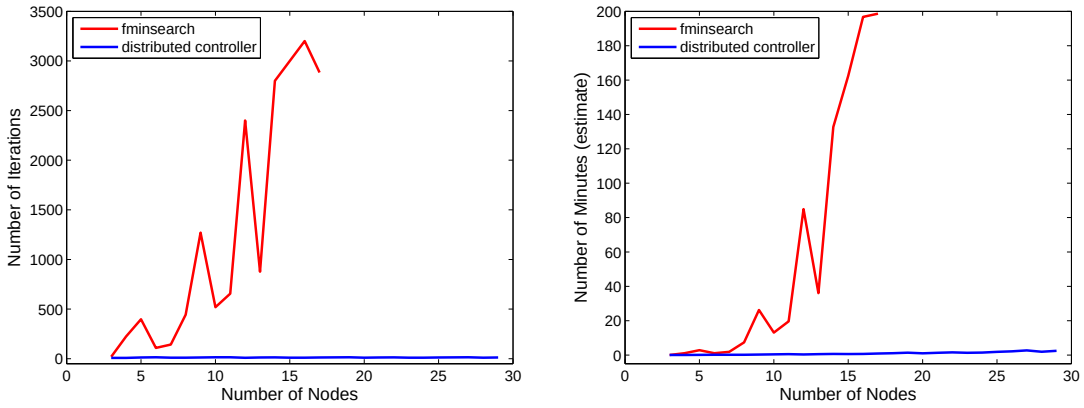
(a) (b)

Figure 5-16: The objective value (a) and total search time (b) for the decentralized controller and fminsearch.

was prohibitive. It is not possible to compute the absolute minimum of the objective function for a given setup; given that the distributed controller and `fminsearch` both find similar minimums, we expect that the value found is very near the global minimum.

Figure 5-16(b) shows the number of minutes required by `fminsearch` and the distributed depth adjustment algorithm. The runtime of the centralized non-linear solver, `fminsearch`, explodes as the number of nodes in the system increases. The runtime of our decentralized depth adjustment controller only has a slight linear increase as the number of nodes increases. Further limits on the size of the neighborhood searched, as discussed in Section 5.4.2, show that this linear increase can be bounded by only integrating over a local neighborhood. Thus, the decentralized depth adjustment algorithm is able to perform well in systems with a very large number of nodes.

Figure 5-17(a) depicts the number of iterations required by `fminsearch` and the decentralized controller necessary to achieve 1.05 times the minimum value of the objective function. Figure 5-17(b) displays the amount of time required to reach this minimum value. The distributed gradient controller is fixed at 20 iterations although averaging 11.6 iterations and a maximum of 15. `fminsearch` often requires large



(a) (b)

Figure 5-17: The number of iterations (a) and minutes (b) until fminsearch and the decentralized controller came within 1.05 times the minimum value found.

number of iterations, even though the objective function value does not decrease significantly with each subsequent iteration (see Figure 5-15). These plots illustrate how quickly each algorithm reaches a value near the ultimate minimum. Note that `fminsearch` only ran on systems with up to 17 nodes due to runtime constraints.

These experiments show that the distributed depth adjustment algorithm outperforms a standard nonlinear optimizer. The distributed depth controller provides a much more reasonable linear increase in computation time per node compared to `fminsearch`.

5.4.6 Dynamic Covariance Energy Analysis

In Section 4.3 we presented two different algorithms to adapt to changing covariances. The first, updates the covariance periodically–adjusting the depth till convergence, pausing for a period, updating the covariance and repeating. The second, continuously updates the covariance function and runs the depth controller. Here we analyze the effect on the energy budget, and therefore lifetime, of two different approaches.

There are three main components that use energy in the underwater sensor network system. These are:

- Core sensor and processing board, which uses about 0.5W

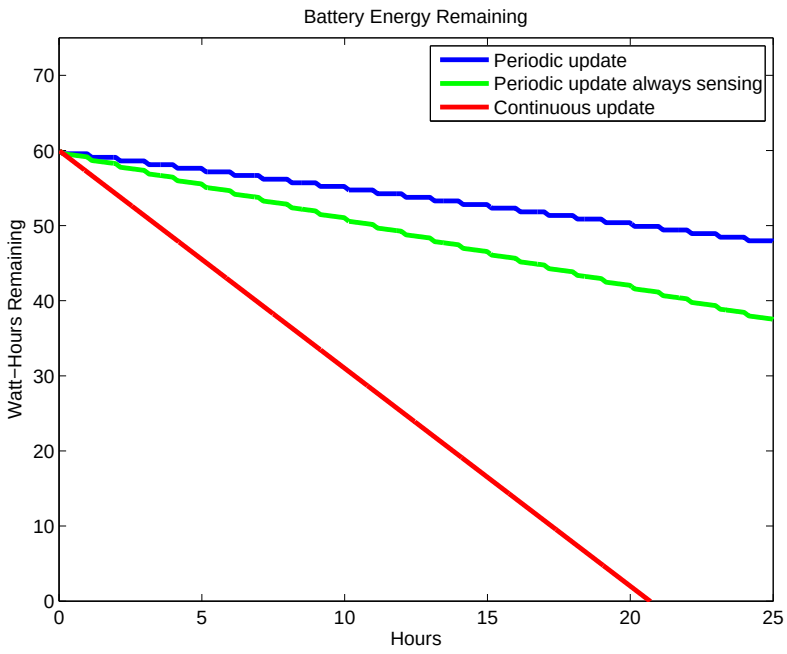


Figure 5-18: The energy used when running the algorithm continuously and once an hour for 10 minutes.

- Acoustic communication, which uses 0.8W when periodically transmitting
- Adjusting depth, which uses up to 1.6W

In order for the depth adjustment algorithm to run all three of these systems must be enabled. When the algorithm is not being run the acoustic communication and depth adjustment system can be disabled. Additionally, the core sensor board can be turned off if continuous sensing is not needed (periodic samples, e.g. every minute, can still be taken with little energy cost).

Figure 5-18 shows the energy remaining over time for three different configurations assuming the batteries start fully charged (60Wh). The lower line shows the charge remaining if the depth adjustment system runs continuously. In this case it lasts for 20.69 hours. If instead, the depth adjustment algorithm runs for 10 minutes every hour and otherwise the acoustic modem and depth adjustment system are shutdown the middle energy usage line shows that the batteries will last for 66.76 hours. Finally, the top line shows the result if the sensor and processing board is

also shutdown when the depth adjustment algorithm is not running. In this case, periodic measurements can still be taken (every minute), but continuous processing and sensing (every second) cannot be performed. This extends the life of the network to 124.54 hours or over 5 days. Further reduction in the frequency of the algorithm runs can further extend the life of the network.

5.5 Hardware Experiments

We performed experiments in the lab, pool, and river using four of the AquaNode underwater sensor nodes. For all experiments the sensor network ran the decentralized depth controller. We tested both the base covariance model discussed in Section 4.2.5 and the model-based covariance discussed in Section 5.2. In addition, we explore the impact of periodically changing the covariance function.

5.5.1 Experimental Setup

We designed the implementation to be flexible and easy to add or update models in the system. However, we also had to be cautious as the processor has limited memory and does not have a floating point unit. To increase speed, wherever possible integer math is used, only resorting to floating point operations where needed to maintain accuracy.

We implemented Algorithm 1 on the system. To maintain maximum flexibility the only function that needs to be updated to change the covariance model is the function `F(p_i,qx,qy,qz)` (line 8). This function takes a sensor located at position `p_i` and the point we are interested in sensing `qx,qy,qz` and returns the covariance between these two points.

For the implementation of `FDz(p_i,qx,qy,qz)` (line 10) we use the numerical gradient of `F` at that position. Using a numerical gradient has the advantage that we do not need to rederive a gradient for each new covariance function, which is challenging when using arbitrary models (that are potentially numeric) for the covariance. The disadvantage of using a numerical gradient is the potential for accuracy and stability problems. In particular, with numeric gradients, choosing the size of the step to use

can be challenging. If it is too small floating point precision issues can occur, if it is to large, inaccuracies can occur. However, the algorithm only knows the depth of each node to the millimeter and the function `F` takes the positions as integers with mm values. Thus, the algorithm uses an offset of $\pm$1mm to compute the gradient numerically. We find that this calculation typically corresponds to over eight decimal places of accuracy as compared to the analytic gradient.

For the base covariance model discussed in Section 4.2.5 we implemented the function `F` as:

```
exp(-(((px-qx)*(px-qx)+(py-qy)*(py-qy))
       /(2.0*SIGMA_SURF*SIGMA_SURF)
    +((pz-qz)*(pz-qz))/(2.0*SIGMA_DEPTH*SIGMA_DEPTH)));
```

with `SIGMA_DEPTH = 4.0` and `SIGMA_SURF = 10.0`. Some additional optimizations were made to limit duplicate computations. For the algorithm the node locations were scaled to be 15m apart along the x-axis, the neighborhood size was $\pm$20m along the x-axis, the virtual depth ranged in z from 0 to 30m, and a step size of 1m was used. These values correspond to values that performed well in the simulations.

For the covariance based on the model data (CDOM covariance) we implemented `F` based on the Gaussian basis function. Since we used Matlab's `newrb` function to compute the basis function, we used their documentation to determine the reconstruction of `F`:

```
val = 0.0;
for(i = 0; i < NUM_BASIS; i++){
  val += netLWX[i]
          *exp(-pow(((fabs(px-qx)-netIWX[i])*netb1X[i]),2));
  val += netLWZ[i]
          *exp(-pow(((fabs(px-qx)-netIWZ[i])*netb1Z[i]),2));
}
val += netb2X + netb2Z;
Yme = Yme + net.b{2};
```

The value `netLW` is the amplitude and `netIW` is the center of the Gaussian as reported by Matlab. The factor `netb1` is the inverse of the variance of the Gaussian. The actual values of these variables are dependent on the model data. Figure 5-3 shows the fit of the Gaussian basis function as compared to the actual data for the Neponset river CDOM covariance data. For this setup, the node locations were scaled to 500m spacing along the x-axis, the neighborhood size was $\pm$800m, the depth ranged from 0 to 3m, used a step size of 40m along the x-axis, and used a step size of 0.1m in depth.

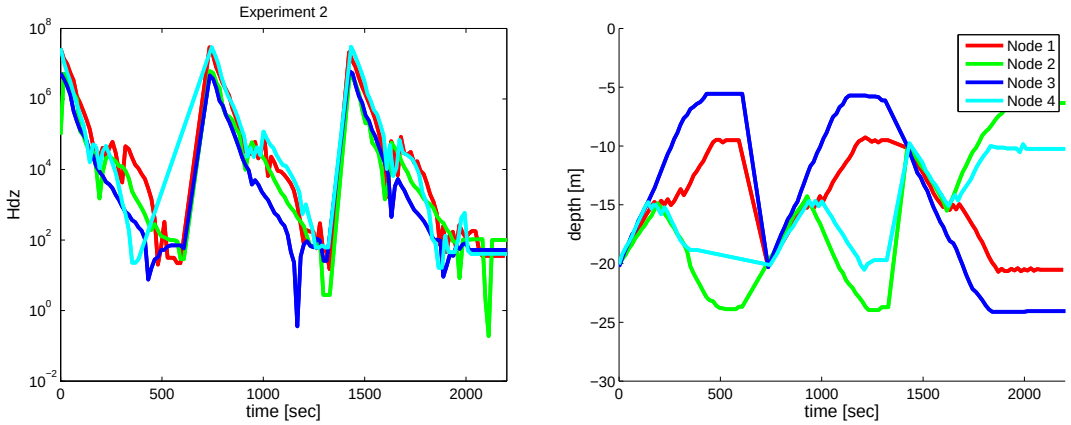
Given these two covariance models and implementations we performed experiments in the lab and in the pool. Both sets of experiments used four underwater sensor nodes. For the lab experiments we placed the transducers of the acoustic modem in a bucket of water and allowed the depth adjustment system to operate freely in air. The lab experiments required the same functionality as the pool experiment while providing improved acoustic communication. For the pool experiments we placed the four sensor nodes in a line in the deep end of a 3m deep pool.

As the pool and bucket did not allow the node spacing used in the setup, we manually set the positions of the nodes to have the proper x-axis spacing. We also scaled each of the node's estimated depth to map the range of depth used in the covariance model to a 1m depth range in the pool. This was to keep the sensor nodes near the middle of the column of water as the acoustic modems were not able to communicate with each other if they were outside of the range.

5.5.2 Results and Convergence

We successfully ran 25 iterations of both covariance models in the bucket and the pool. For the Gaussian we ran 4 trials in the pool. Each trial converged within 12 minutes with each iteration averaging 14s. For the CDOM covariance we ran 5 pool trials. Each trial converged within 20 minutes with each iteration averaging 35s.

Figures 5-19(a) and 5-20(a) depict the absolute value of the cost function, $\frac{\partial \mathcal{H}}{\partial z_i}$, for each node in the pool while using the Gaussian covariance. Initially, the gradient of the objective function was high; however, over the course of the experiment the value on each node decreased until it reached a stable state.



(a) (b)

Figure 5-19: The value of $\frac{\partial \mathcal{H}}{\partial z_i}$ (a) and the depths (b) over the course of a three experimental runs.

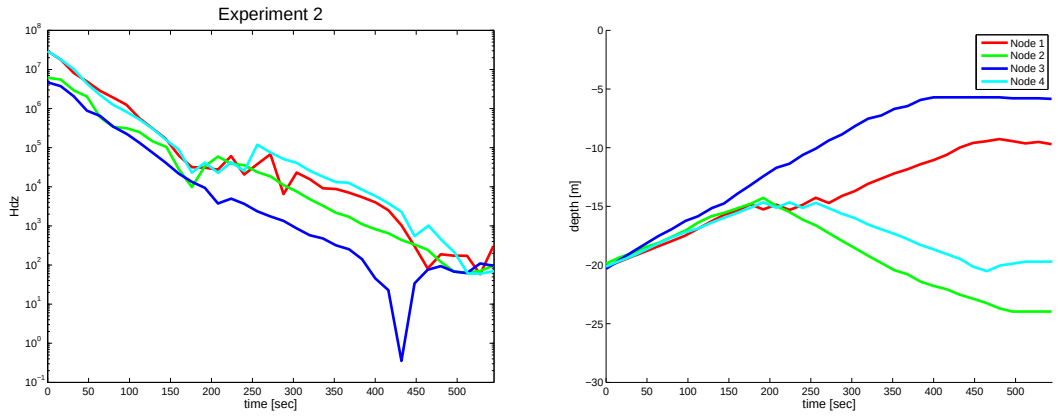
Figures 5-19(b) and 5-20(b) show the depths of each of the nodes over the course of an experiment. Initially, the nodes were started at 20m. All of the nodes approached the center of the water column after 200s. From here Nodes 1 and 3 continued up in the water while Nodes 2 and 3 returned to a lower depth. The total time to convergence in this experiment was approximately 8 minutes.

Table 5.3 shows the start and end configurations for 6 of the experiments we performed using the Gaussian covariance function. In most of the experiments the controller converged to a configuration where the nodes were oriented in a zig-zag configuration. An exception to this is trial Bucket 3 where the nodes initially started in a diagonal configuration. In this case, the nodes converged to a down-up-up-down configuration, a local minimum. Section 5.4.3 explores how different start configurations effect the final positioning of the nodes.

Similarly, Table 5.4 shows the start and end configurations for 6 of the pool and bucket experiments for the river model covariance function. The results here are similar to that of the previous set of experiments.

5.5.3 Communication Performance

The acoustic channel is a very low bandwidth, high noise environment. Despite being placed close together in the pool, the communications were similar to what we



(a) (b)

Figure 5-20: The value of $\frac{\partial \mathcal{H}}{\partial z_i}$ (a) and the depths (b) over the course of a single experiment.

typically find in river experiments in that all nodes hear single-hop neighbors and some nodes hear further nodes. This is due to the highly reflective and therefore challenging acoustic environment in the pool.

In our implementation, each node only used depth information from neighboring nodes whose last depth message was received within the past two minutes. Figure 5-21(a) shows the number of neighbors each node used in the calculation of the decentralized depth controller, $\frac{\partial \mathcal{H}}{\partial z_i}$. The nodes were in a line in numerical order. Node 1 typically only heard Node 2; Node 2 heard 1, 3, and 50% of the time heard 4; Node 3 heard 2, 4; and Node 4 heard 3 and 20% of the time heard 2.

Figure 5-21(b) shows the lag in Node 2's estimate of the depths of the other three nodes. As we use a TDMA communication algorithm, we expect to receive an update from each node every 16 seconds. However, due to packet loss the updates may arrive less frequently. Thus, the algorithm may be use somewhat old data to calculate the controller output, although never older than two minutes. The experiments in this section show that despite sometimes poor communication, the decentralized controller still converges and is robust.

	Node0	Node1	Node2	Node3
Bucket 1 Start	10.0m	10.0m	10.0m	10.0m
Bucket 1 Final	10.3m	24.1m	5.9m	19.7m
Bucket 2 Start	20.0m	20.0m	20.0m	20.0m
Bucket 2 End	19.8m	5.9m	23.8m	10.2m
Bucket 3 Start	3.7m	7.8m	12.2m	15.9m
Bucket 3 End	9.5m	22.9m	23.9m	9.6m
Pool 1 Start	10.2m	9.9m	10.1m	9.8m
Pool 1 End	20.6m	6.9m	24.1m	10.2m
Pool 2 Start	20.0m	20.1m	20.3m	20.1m
Pool 2 End	9.5m	23.9m	5.6m	18.8m
Pool 3 Start	20.2m	19.9m	20.3m	20.1m
Pool 3 End	9.6m	24.0m	5.8m	19.7m

Table 5.3: Selected start and end configurations for a number of bucket and pool experiments using the base covariance function.

5.5.4 Updating Covariance

We performed experiments to characterize the performance of the decentralized depth adjustment algorithm when the covariance function changes periodically. Figure 5-23(bottom)(a-e) shows the concentration of CDOM along the Neponset River based on the model described in Section 5.2. Each subfigure (a-e) plots this for (a) 3.25m, (b) 3.0m, (c) 2.75m, (d) 2.5m, and (e) 2.25m of average water depth. The changes in water level are due to tidal effects. Figure 5-23(top)(a-e) shows the numeric covariance for each of these plots normalized to fall between zero and one.

We deployed 4 nodes in the Charles river and simulated updating the covariance function to determine the effect of periodic covariance function updates. Figure 5-22 shows the results. This figure plots the value of the objective function that each node computes over time as well which covariance function from Figure 5-23 is currently in use (step function at top of Figure 5-22).

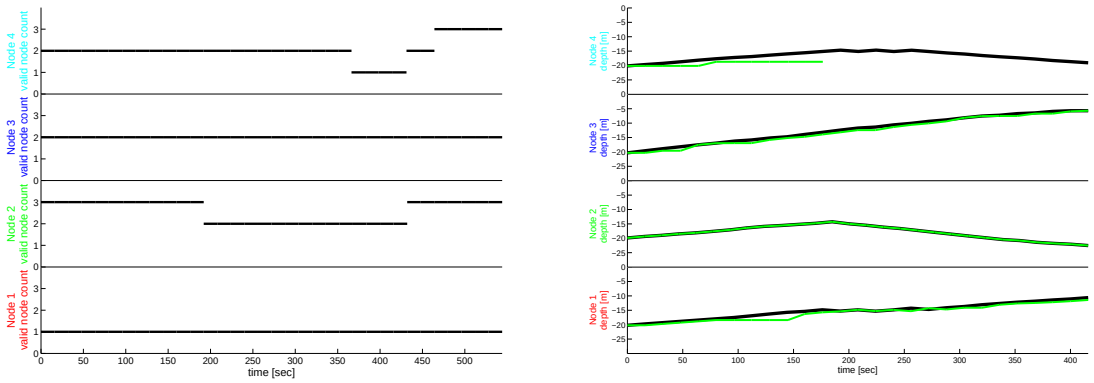
In this experiment, the nodes initially had very high objective functions. They then moved, which lowered the objective function. After the nodes stabilized we changed the covariance function from the 3.25m data to the 3.0m data. Interestingly, the objective function did not change significantly. Similarly, the transition from 3.0m to 2.75m objective function did not have much impact. When moving to

	Node0	Node1	Node2	Node3
Bucket 1 Start	1.0m	1.0m	1.0m	1.0m
Bucket 1 Final	1.6m	0.7m	0.6m	2.4m
Bucket 2 Start	1.0m	1.0m	1.1m	1.0m
Bucket 2 End	2.5m	1.5m	1.7m	0.5m
Bucket 3 Start	1.0m	2.0m	1.0m	2.0m
Bucket 3 End	0.8m	2.4m	0.6m	2.1m
Pool 1 Start	1.0m	1.0m	1.0m	1.0m
Pool 1 End	2.4m	1.5m	0.5m	2.4m
Pool 2 Start	2.0m	2.0m	2.0m	2.0m
Pool 2 End	0.7m	2.3m	0.7m	2.2m
Pool 3 Start	2.0m	2.1m	2.0m	2.0m
Pool 3 End	0.7m	2.8m	0.8m	2.8m

Table 5.4: Selected start and end configurations for a number of bucket and pool experiments using the river covariance function.

2.5m, however, the objective function increased greatly. This caused the decentralized controller to adjust the depths of the nodes to again reduce the objective value. A final change in the covariance function from 2.5m to 2.25m also resulted in a spike in the objective function, which the decentralized depth controller quickly minimized by changing the depths of the nodes.

This experiment verifies that the decentralized controller handles changes to the covariance function in a river environment. Interestingly, some changes to the covariance function result in very minor changes to the value of the objective function, however, others cause significant changes.



(a) (b)

Figure 5-21: The number of neighbors used in $\frac{\partial \mathcal{H}}{\partial z_i}$ calculation (a) and one node's estimate the other nodes' depths versus the actual depths over time.

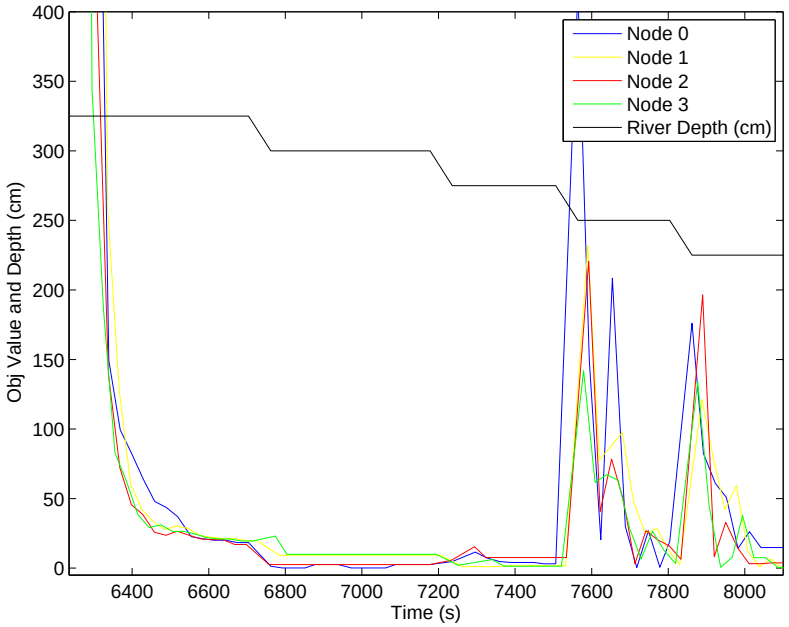


Figure 5-22: Objective value for 4 nodes when changing the depth-based covariance function.

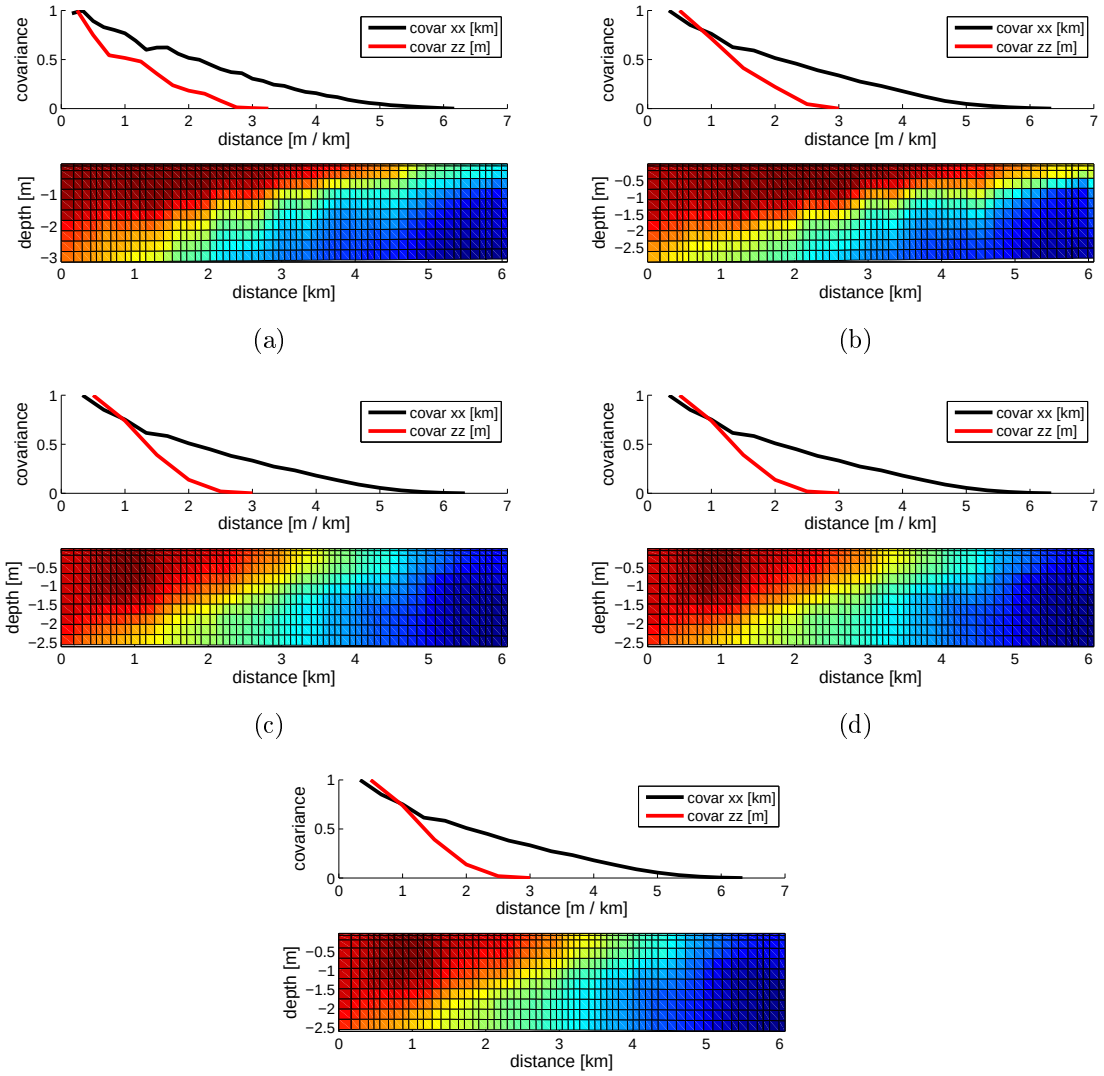


Figure 5-23: The different CDOM values (bottom) and covariance functions (top) for the Neponset River with tidal river level of (a) 3.25m, (b) 3.0m, (c) 2.75m, (d) 2.5m, and (e) 2.25m

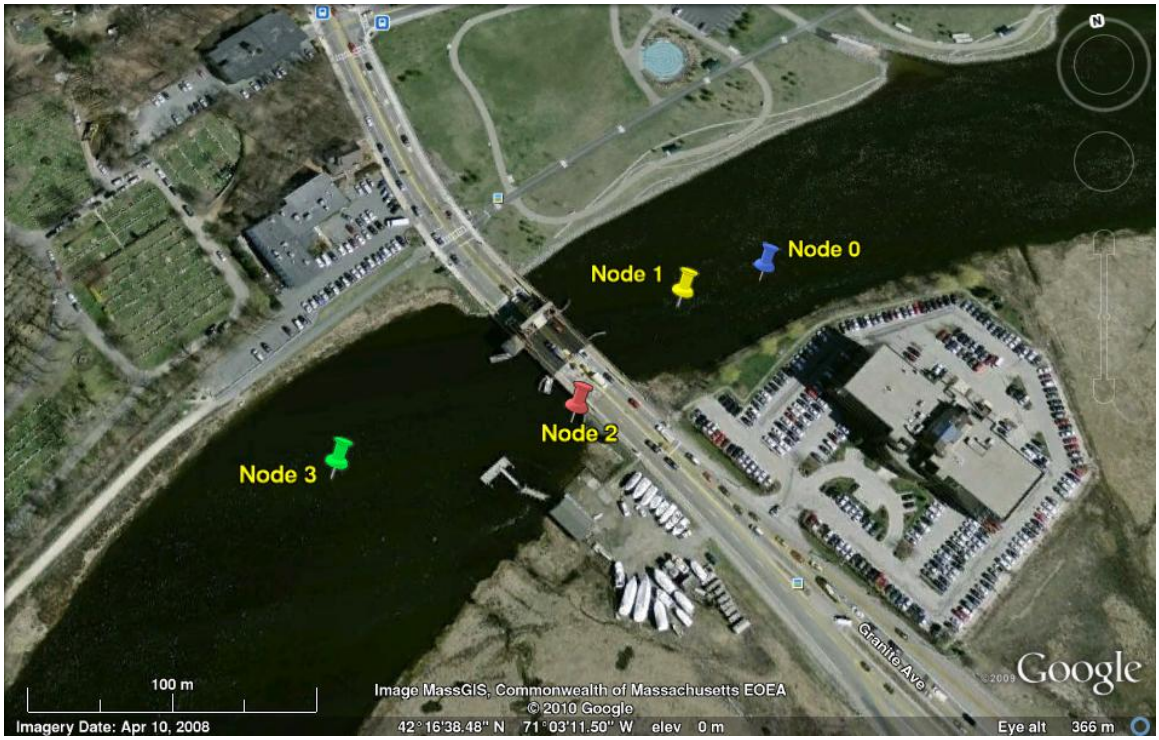


Figure 5-24: Picture of the Neponset River deployment.

5.5.5 Neponset River Experiment

In addition to performing experiments to verify the algorithm in controlled pool and river experiments, we also performed a scientific data collection experiment in the Neponset River, which is located just south of Boston, MA. Figure 5-24 shows a map of the deployment. We deployed four nodes around the Granite Avenue bridge, two on each side of the bridge.

Figure 5-25 shows a picture of an AquaNode in the water at the surface (left in image) and the marker buoy. The node was connected directly to an anchor at the bottom of the river. A length of chain was also attached to the anchor. A rope went from the chain to the marker buoy. The chain allowed us to offset the marker from the location of the anchor and sensor node, reducing the chance of the two lines entangling.

We started the experiment at high tide during which the depth at the deployment locations ranged from 3m to 6m. After 7 hours we recovered the nodes at low tide.

Figure 5-25: Picture of an AquaNode in the Neponset River.

The tidal change was nearly 3m. After 4.25 hours there was a large rain storm that caused significant runoff into the river.

The nodes were programed to perform column profiles of the water, going to the surface for 5 minutes and then returning to the bottom for 10 minutes. In addition, the nodes communicated and ran the dynamic depth adjustment algorithm. However, we did not control the depths of the nodes using the algorithm as we wanted a baseline set of data to use to obtain covariance information.

Sensing and Depth Adjustment

Each node had temperature, pressure, and CDOM sensors. The CDOM sensors were external to the nodes and we used two different types. The first was a miniature sensor with a diameter of about 2cm and a length of about 10cm. The other sensor was nearly the same diameter as the sensor node and half the height. We used two of each sensor. The small sensors were attached to nodes 0 and 2 and the large sensors were attached to nodes 1 and 3. The small sensors were attached directly to the AquaNodes, while the larger sensors were made neutrally buoyant and allowed to float in the water. Unfortunately, the larger sensors created too much drag in the water given the fairly significant water current in the river. This caused the over

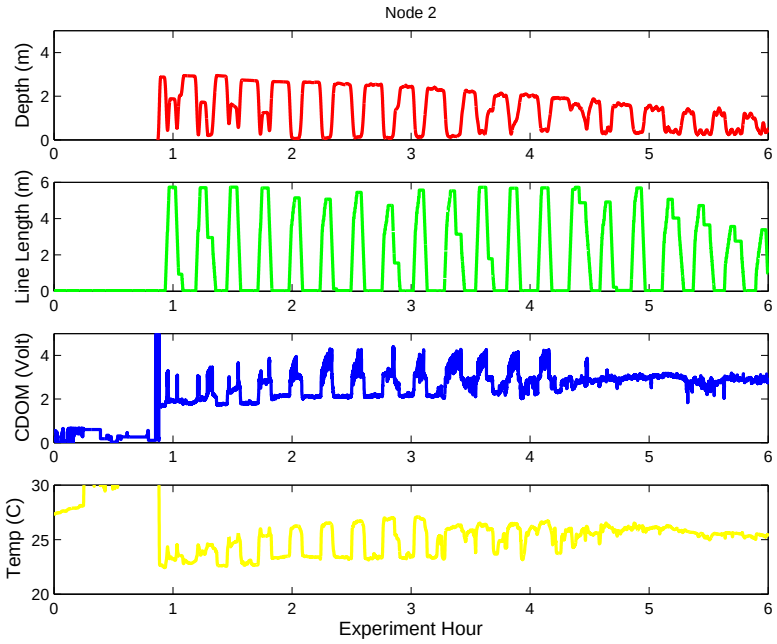


Figure 5-26: Data from Node 2 in the Neponset River.

current software protection to disable the winch. Thus, the two nodes with the large sensors did not perform the depth transects.

Figure 5-26 shows the data collected from Node 2 during the Neponset River experiment. The top (red) portion of the figure shows the depth as measured by the pressure sensor. This shows the depth transects as the node moved up and down. The first few show that it went up and then down briefly before going up and down again. This was most likely caused by poor initial winding on the spool causing the line to reverse winding. After a few times the spool corrected itself.

The second plot down in Figure 5-26 (green) shows the length of the line (based on motor encoder data) that was deployed in order to reach the surface. At the start of the experiment it took 6m of line in order to surface from a depth of approximately 3m. This node was limited to deploying 6m of line. This means that the water current was dragging the node at an angle of almost 45°. As the water depth decreased with the tidal change the amount of line needed stayed nearly the same until the last hour of the experiment. This is due to the fact that as the tide changed the river current

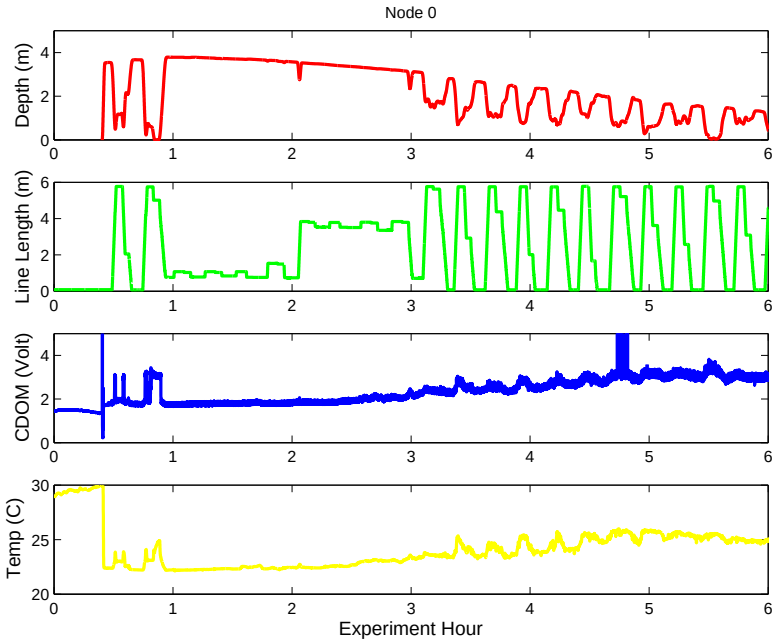


Figure 5-27: Data from Node 0 in the Neponset River.

increased, causing an even larger angle in the water. This shows that in rivers with relatively fast currents additional flotation material should be added to the nodes to increase their buoyancy and reduce the anchor line angle in the water.

The third and fourth plot down in Figure 5-26 (blue and yellow, respectively), show the CDOM sensor reading as well as the temperature sensor reading. These show that at the surface there was a higher CDOM reading as well as a higher temperature reading. However, after slightly more than 4 hours into the experiment, there was a significant rain storm. The sensor readings indicate that after this point the conditions in the water were much more uniform across the depths in the water.

Figure 5-27 shows the data from Node 0. From hour 1 to 3 of this experiment the node stayed at the bottom of the water column. This was caused by software over current protection on the winch that disabled motion. Most likely the node was tangled (perhaps with marker float line) or was otherwise impeded. After hour 3 the node recovered and performed the depth transects.

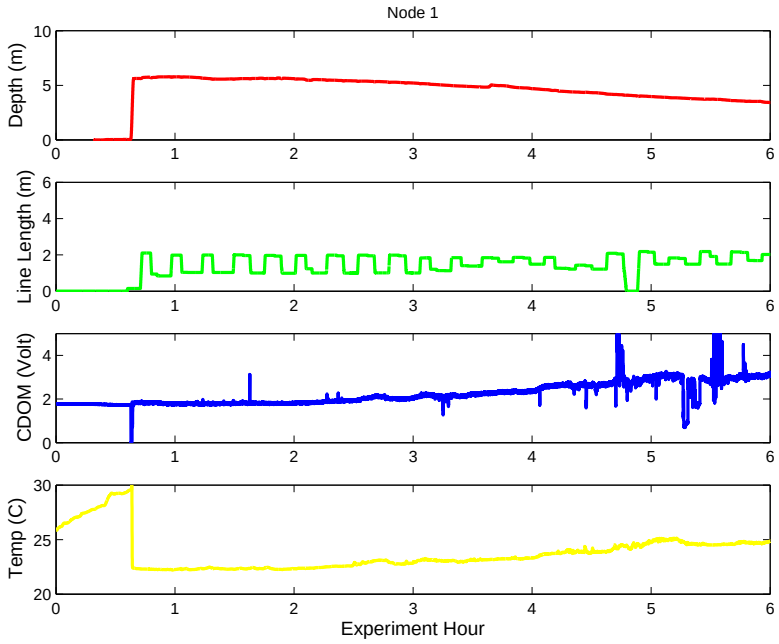


Figure 5-28: Data from Node 1 in the Neponset River.

Figures 5-28 and 5-29 show the data from Nodes 1 and 3. These nodes had the large CDOM sensors. The depth readings show that these nodes stayed continuously at the bottom of the water column. This was most likely caused by the large drag from the CDOM sensor. The current caused the sensor to be dragged completely horizontally in the water and as such it was resting on the bottom. This triggered the software current limit on the winch, which disabled motion for the majority of the experiment. Further experiments need to be performed to determine the proper amount of buoyancy and the proper current limit for the nodes under these conditions. The sensors on Node 3 failed to record for some unknown reason. Tests before the experiment showed that the sensors were working, further tests need to be performed to determine the reason for failure.

Communication and Depth Adjustment Algorithm

Figure 5-30 shows the communication success rate between the nodes during the experiment. The distance from Node 0 to 1 was 35m, 1 to 2 was 64m, and 2 to 3

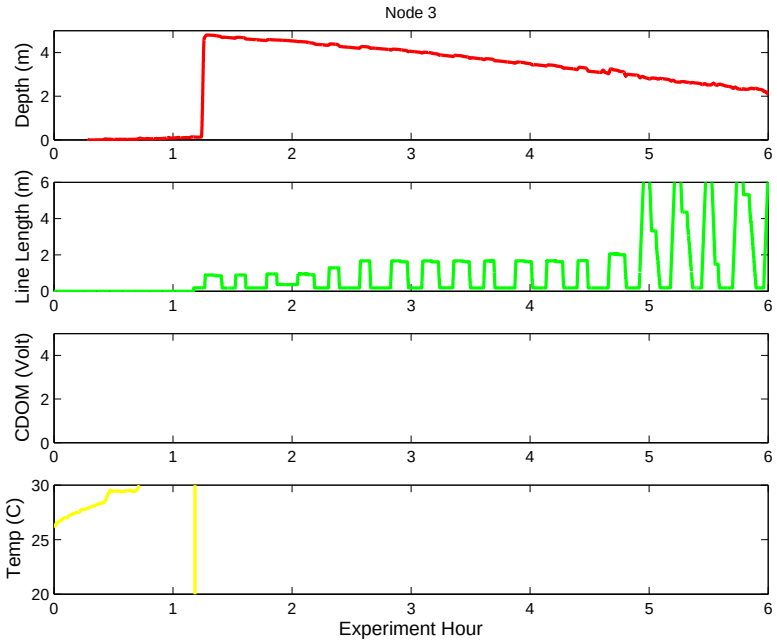


Figure 5-29: Data from Node 3 in the Neponset River.

	Node0	Node1	Node2	Node3
Node 0	-	8.3	8.1	200.7
Node 1	4.6	-	6.6	238.7
Node 2	39.0	59.0	-	46.1
Node 3	248.6	248.6	51.2	-

Table 5.5: Average time, in minutes, between communication during the Neponset River experiment.

was 93m. Table 5.5 summarizes the average time between hearing a messages. This shows, that the nodes were able to hear their single hop neighbors and occasionally a two hop neighbors. Figure 5-30 also shows that some periods of time had better communication than others. For instance in (a) and (b) between hours 2 and 4 of the experiment the communication success rate was much better than other times. The communication statistics show that it is possible to communicate in a shallow river environment at ranges of over 100m.

Figure 5-31 shows the value of the decentralized depth adjustment controller output over the course of the deployment. During the experiment the gain on the controller output was set low so that it converged slowly. In addition a deadband of 20

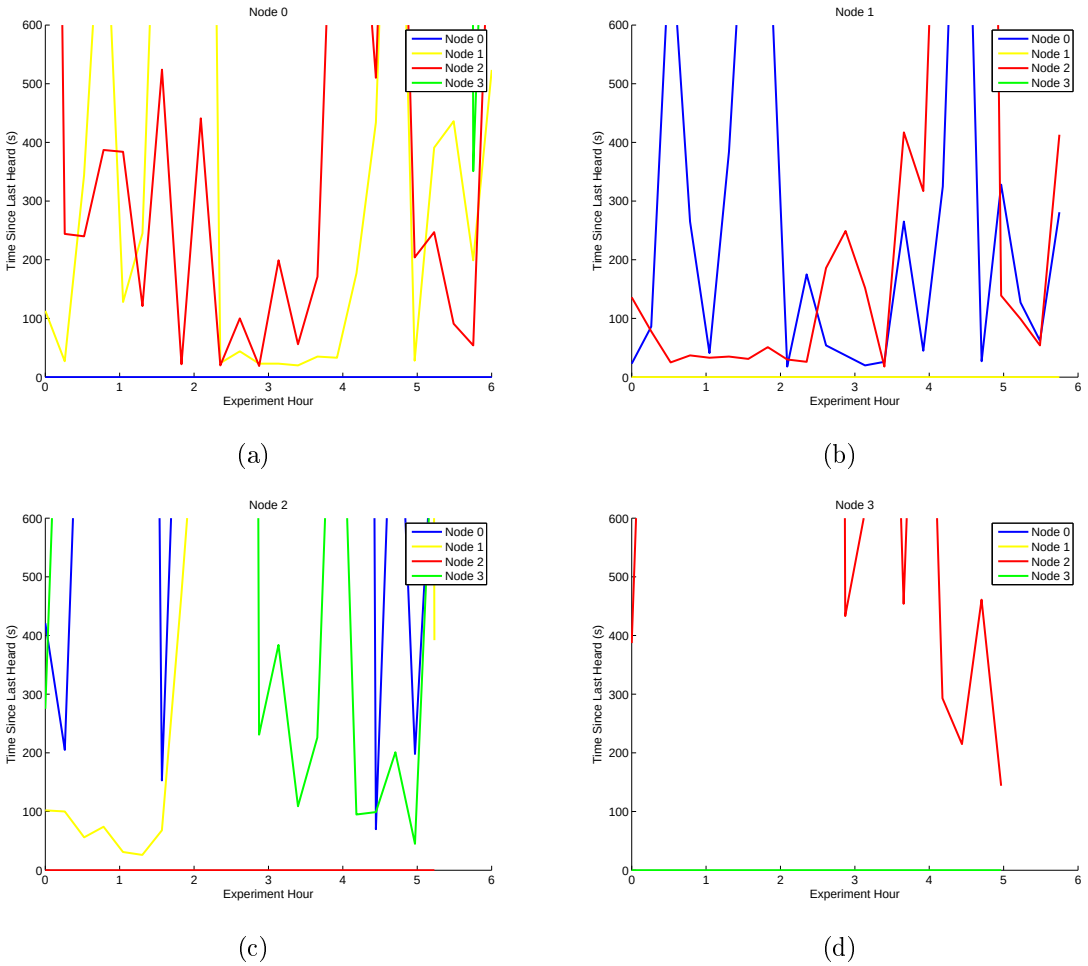


Figure 5-30: Communication data for the AquaNodes in the Neponset River.

was placed on the controller, meaning the nodes would not move if the value of HDz was less than 20. This explains the reason why the controller output does not go to zero.

Neponset River Summary

In this experiment we demonstrated the performance of the depth adjustment system and the decentralized gradient controller in a real setting. We collected pressure, temperature, and CDOM measurements for over 7 hours from high to low tide. We learned that properly adjusting the buoyancy of the nodes and minimizing the drag of external sensors is critical to enable proper depth adjustment.

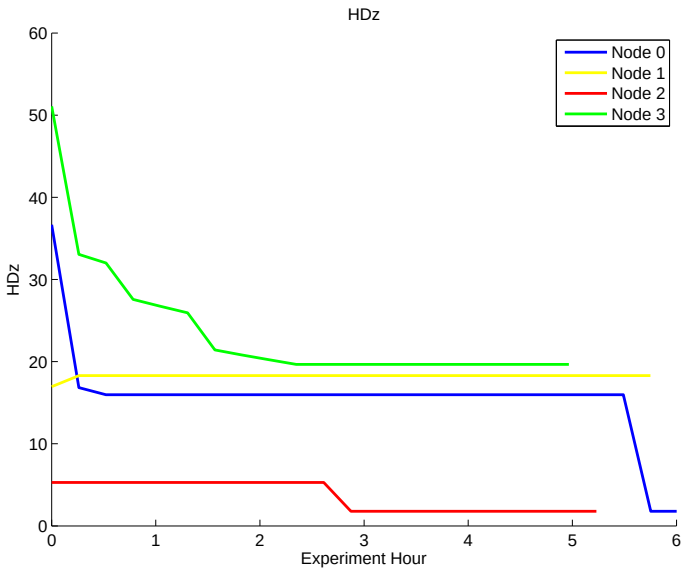


Figure 5-31: The controller output HDz during the Neponset River experiment.

Chapter 6

Underwater Robot Localization: Algorithm

6.1 Introduction

In Chapters 4 and 5 we presented algorithms and experiments to adjust the depth of the underwater sensor nodes to optimize sensing, while assuming known positions. In this chapter we develop and describe algorithms for localizing static and dynamic nodes in a 3D underwater sensor network where nodes can exchange and/or estimate ranges. In particular, we develop an acoustic underwater localization algorithm that allows the underwater robot, AMOUR, to act as a mobile sensor in the underwater sensor network and to fill in any gaps left by the AQUANODES.

Sensors need to associate sensed values with the locations of these values. In some applications it is advantageous to introduce a mobile networked robot in the system. In case the sensors detect an interesting event, the robot can travel to that location to collect data using more powerful sensors. For instance, if the AQUANODES detect an increase in pollution in an area of coral reef, AMOUR can be sent out to photograph the coral to detect any early onset of bleaching. In addition, the robot acts as a mobile data-mule. It travels to each AQUANODE and downloads their data by hovering over them and using the optical modem [103]. Both of these require accurate positioning of

the robot as it moves through the network to add location information to the sensed data and to find the static sensor nodes.

While at the surface, AMOUR uses GPS to obtain position information. Once underwater, however, GPS is unavailable due to the absorption of radio frequencies by the water. In this chapter we develop an underwater localization algorithm for underwater robots that finds the provably optimal location for the robot given range information to the static sensor nodes. The algorithm is geometrically inspired and only requires an upper bound on the robot speed. In addition to optimally localizing the robot, the algorithm is efficient, as in practice it only requires constant computation per location update.

There are a number of challenges associated with localizing a robot underwater when GPS is unavailable. The first is that the mobile robot only receives ranges to the static nodes periodically and asynchronously. The second is the lack of good dead-reckoning in inexpensive underwater vehicles. On land, robots use wheel encoders to estimate their motion and position over time. This type of simple and inexpensive dead-reckoning is unavailable underwater.

The first challenge, only receiving asynchronous and periodic range measurements, implies that the robot will move significantly between range measurements, thereby eliminating straight-forward tri- or multi-lateration approaches. The range measurements are obtained by measuring the round-trip transmission time of an acoustic message between the robot and sensor nodes. Due to the narrow and slow acoustic communication channel, the robot only obtains range measurements every 1 to 4 seconds. During this time between ranges AMOUR may move over five meters.

An additional challenge with inexpensive underwater robots is the lack of precise dead-reckoning systems.[1] To reduce costs, AMOUR only uses a basic inertial measurement unit (IMU). The IMU provides accurate orientation information but poor estimates of translational motion. In part, the poor translational motion calculation

[1]For example, a Doppler velocity logger (DVL) provides a very good estimate of the absolute motion of an underwater vehicle by measuring the Doppler shift of an acoustic signal reflected off of the floor of the ocean. For AMOUR and other inexpensive AUVs, a DVL is cost prohibitive, costing tens of thousands of dollars, many times the cost of AMOUR, which costs less than $10k.

is due to currents in the water that cause the absolute translational motion to vary greatly from the computed motion based on IMU and thruster data. Thus, the localization algorithm cannot rely on dead-reckoning and only uses a maximum bound on the speed of the robot.

Fortunately, it is very easy to measure the depth of the robot underwater using an inexpensive pressure sensor. These sensors measure the pressure of the water and can accurately estimate the depth based on water pressure. This reduces the localization problem from a 3D problem to a 2.5D problem since the depth of the robot is known.

The work described in this chapter covers and extends the previous work first described in our Master's thesis [31], and is included here for completeness. This chapter starts by formulating the localization problem in Section 6.2. We then formally present and analyze the algorithm in Section 6.3, showing that it finds the optimal localization regions. Chapter 7 details results of the localization algorithm simulations and a new experimental implementation and results.

6.2 Problem Formulation and Intuition

While at the surface the robot is able to use GPS, however, once it is underwater GPS is not available. Instead, the robot relies on acoustic range information obtained periodically from the static underwater sensor nodes. The main challenge is that the ranges from static nodes are received over time, not consecutively. This means that the robot will have moved significantly between range measurements. In our system we obtain ranges every 1 to 4 seconds. In this time the robot can move on the order of meters. In addition, the robot does not have dead-reckoning capabilities, so we must rely only on an upper bound on robot speed.

In this section we formalize the problem setup. First, however, we discuss the assumptions that the algorithm makes.

6.2.1 Assumptions

The underwater mobile robot localization algorithm makes a very minimal set of assumptions. Namely:

- The locations of the static nodes are known;
- Periodic ranges are obtained to the static nodes;
- An upper bound on the mobile node speed is known;
- The vehicle does not have dead-reckoning capabilities.

Determining the locations if the static nodes is discussed in Section 6.2.2. The mobile node is able to obtain period range measurements to the static nodes by using the acoustic modem. The details of obtaining ranges with the acoustic modem is discussed in Section 7.6.2. Determining the upper bound on maximum water speed of the mobile node is easily measured by driving the vehicle at full thrust. In practice, there are some other considerations that must be included in this calculation. For instance, in the presence of strong currents the vehicle ground speed may be much larger than the actual water speed. Thus, the speed must be increased to account for any potential currents.

Finally, we assume that the vehicle does not have expensive dead-reckoning capabilities. If it does, then this information can be incorporated into the algorithm, however, there are a large set of cases where this information is unavailable. In large part the main constraint is cost. The systems needed to obtain good dead-reckoning capabilities, such as a Doppler velocity logger (DVL), cost tens of thousands of dollars. Our underwater vehicle costs less than $10k. Including a DVL would drastically increase the price. In addition, in some environments, such as in deep water, a DVL does not work. Other instruments can be used, such as inertial measurement units and compasses, however, these too can fail in some setups. For instance, near ship hulls or oil pipelines the compass will have erroneous readings, causing problems for the dead-reckoning systems. Thus, it is realistic to assume that dead-reckoning information is not always available and that algorithms must be developed to handle cases where it is unavailable.

6.2.2 Localizing Static Nodes

In order for the mobile robot localization algorithm to work, the robot must know the locations of the static nodes from which it is receiving ranges. We utilize two approaches for localizing static nodes in the network. The first, which we have implemented and tested on our physical system, is an extension from 2D to 3D of a robust localization algorithm by Moore *et al.* [71]. This algorithm requires the exchange of all range information throughout the network ($O(N^2)$ communication overhead). Each node can then robustly compute the locations of all nodes in the network. This algorithm takes care to prevent flip ambiguities[2] which can lead to large positional errors based on small errors in range measurements. In typical deployments it takes approximately 30 minutes to obtain sufficient range information for the static localization algorithm.

The static localization algorithm does not, however, give absolute geographic coordinates. Rather, it computes a local coordinate system, which all nodes can agree on. The process of transforming this local coordinate system into a real geographic coordinate system can be done by obtaining three GPS positions with the correlating local coordinate. This enables solving for the rotation, translation, and flip ambiguity parameters between the systems.

The second approach to localize the static nodes is to make use of the depth adjustment system. By going to the surface the sensor nodes can obtain GPS positions. This is useful to figure out the transform between the coordinate systems of the self computed positions, but can also be used to compute the locations of all sensor nodes if they are all equipped with the winch hardware. The final depth of the nodes can be measured using an inexpensive pressure sensor that measures the water pressure to estimate depth. These sensors are also used in the static and mobile robot localization algorithm.

[2] A flip ambiguity is when a node can have multiple locations based on the available, noisy range information.

6.2.3 Definitions

We now define a generic formulation for the localization problem. This setup can be used to evaluate range-only, angle-only, and other localization problems, however, we will focus on the range-only localization problem. This material is based on [31]. For details on other formulations see the Master's Thesis [31].

We start by defining a *localization region*.

Definition 1. *A localization region at some time t is the set of points in which a node is assumed to be at time t.*

We will often refer to a localization region simply as a region. It is useful to formulate the localization problem in terms of regions as the problem is typically under-constrained, so exact solutions are not possible. Probabilistic regions can also be used, however, we use a discrete formulation. In this framework the localization problem can be stated in terms of finding *optimal* localization regions.

Definition 2. *A localization region is optimal with respect to a set of measurements at time t if at that time it is the* smallest *region that must contain the true location of the mobile node, given the measurements and the known velocity bound. A region is individually optimal if it is optimal with respect to a single measurement.*

For example, for a range measurement the individually optimal region is an arc or annulus. Another way to phrase optimality is if a region is optimal at some time t, then the region contains the true location of the mobile node and all points in the region are reachable by the mobile node.

6.2.4 Intuition

Suppose that from time $1 \cdots t$ we are given regions $A_1 \cdots A_t$ each of which is individually optimal. The times need not be uniformly distributed, however, we assume that they are in sorted order. By definition, at time k region A_k must contain the true location of the mobile node and furthermore, if this is the only information we

have about the mobile node, it is the smallest region that must contain the true location. We now want to form regions, $I_1 \cdots I_t$, which are optimal given all the regions $A_1 \cdots A_t$ and an upper bound on the speed of the mobile node which we will call s. We refer to these regions as *intersection regions* as they will be formed by intersecting regions.

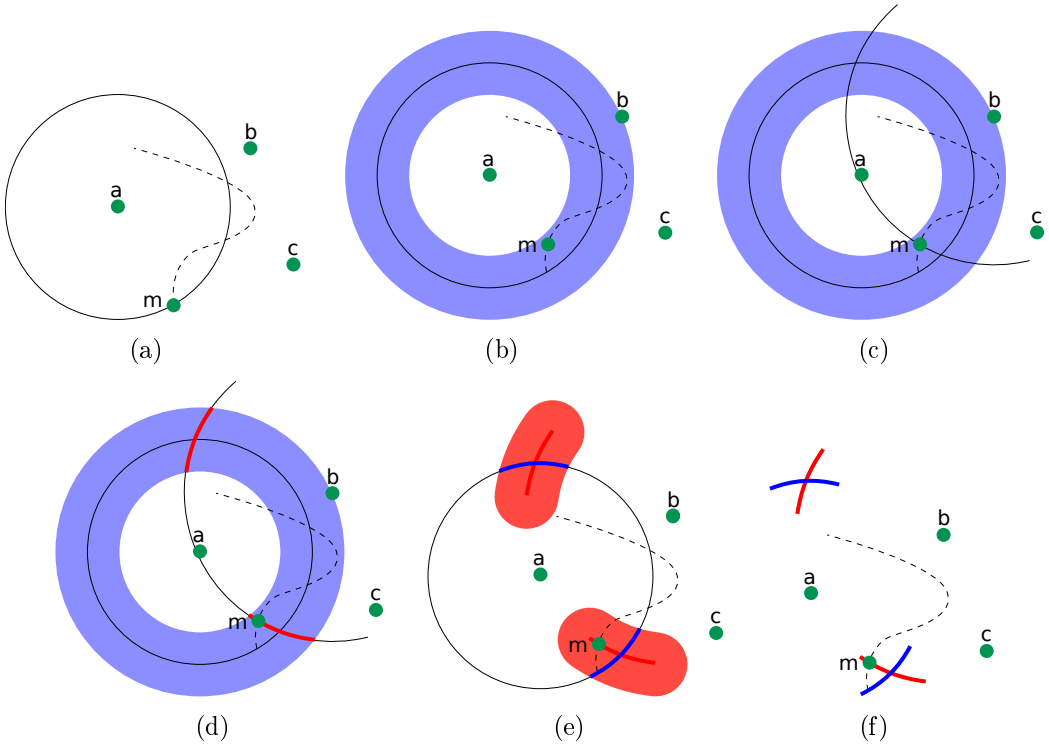


Figure 6-1: Example of the range-only localization algorithm.

Figure 6-1 shows the critical steps in the range-only localization of mobile Node **m**. Node **m** is moving through a field of localized static nodes (Nodes **a**, **b**, **c**) along the trajectory indicated by the dotted line.

At time t Node **m** obtains a range to Node **a**. This allows Node **m** to localize itself to the circle indicated in Figure 6-1(a). At time $t+1$ Node **m** has moved along the trajectory as shown in Figure 6-1(b). It expands its localization estimation to the annulus in Figure 6-1(b). The size of the annulus is determined by the known maximum speed of the robot. Node **m** then enters the communication range of Node **b** and obtains a ranging to Node **b** (see Figure 6-1(c)). Next, Node **m** intersects the circle and annulus to obtain a localization region for time $t+1$ as indicated by the

Algorithm 3 Localization Algorithm

```
1:  procedure LOCALIZE(A_1 ··· A_t)
2:      s ← max speed
3:      I_1 = A_1                          ▷ Initialize the first intersection region
4:      for k = 2 to t do
5:          △t ← k − (k − 1)
6:          I_k =Grow(I_{k−1}, s△t) ∩ A_k      ▷ Create the new intersection region
7:          for j = k − 1 to 1 do                  ▷ Propagate measurements back
8:              △t ← j − (j − 1)
9:              I_j =Grow(I_{j+1}, s△t) ∩ A_j
10:         end for
11:     end for
12: end procedure
```

bold red arcs in Figure 6-1(d). This is the localization region. This region depends on the maximum speed of the robot and the specific ranges it hears.

The range taken at time $t+1$ can be used to improve tracking at time t as shown in Figure 6-1(e). The arcs from time $t+1$ are expanded to account for all locations the mobile node could have come from. This is intersected with the range taken at time t to obtain the refined location region illustrated by the bold blue arcs. Figure 6-1(f) shows the final result. Note that for times t and $t+1$ there are two possible location regions. This is because two range measurements do not provide sufficient information to fully constrain the system. Range measurements from other nodes will quickly eliminate this.

6.3 Localization Algorithm

The localization algorithm follows the same idea as in Section 6.2. Each new region computed will be intersected with the grown version of the previous region and the information gained from the new region will be propagated backwards. Algorithm 3 shows the details.

Algorithm 3 can be run online by omitting the outer loop (lines 4-6 and 11) and executing the inner loop whenever a new region/measurement is obtained.

The first step in Algorithm 3 (line 3), is to initialize the first intersection region to be the first region. Then we iterate through each successive region.

The new region is intersected with the previous intersection region grown to account for any motion (line 6). Finally, the information gained from the new region is propagated back by successively intersecting each optimal region grown backwards with the previous region, as shown in line 9.

6.3.1 Algorithm Details

Two key operations in the algorithm, which we will now examine in detail, are `Grow` and `Intersect`. `Grow` accounts for the motion of the mobile node over time. `Intersect` produces a region that contains only those points found in both localization regions being intersected.

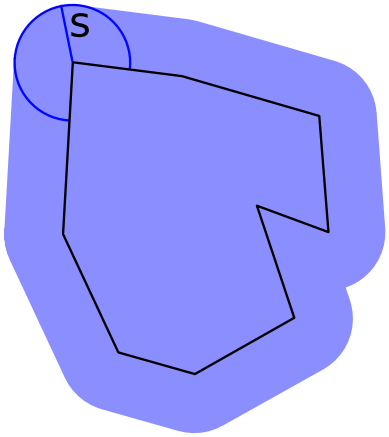


Figure 6-2: Growing a region by s. Acute angles, when grown, turn into circles as illustrated. Obtuse angles, on the other hand, are eventually consumed by the growth of the surroundings.

Figure 6-2 illustrates how a region grows. Let the region bounded by the black lines contain the mobile node at time t. To determine the smallest possible region that must contain the mobile node at time $t+1$ we `Grow` the region by s, where s is the maximum speed of the mobile node. The `Grow` operation is the Minkowski sum [34] (frequently used in motion planning) of the region and a circle with diameter s.

Notice that obtuse corners become circle arcs when grown, while everything else "expands." If a region is convex, it will remain convex. Let the complexity of a region be the number of simple geometric features (lines and circles) needed to describe it. Growing convex regions will never increase the complexity of a region by more than

a constant factor. This is true as everything just expands except for obtuse angles which are turned into circles and there are never more than a linear number of obtuse angles. Thus, growing can be done in time proportional to the complexity of the region.

A simple algorithm for `Intersect` is to check each feature of one region for intersection with all features of the other region. This can be done in time proportional to the product of the complexities of the two regions. While better algorithms exist for this, for our purposes this is sufficient as we will always ensure that one of the regions we are intersecting has constant complexity as shown in Section 6.3.3. Additionally, if both regions are convex, the intersection will also be convex.

Next we prove the correctness and optimality of Algorithm 3. We show that the algorithm finds the location region of the node, and that this computed location region is the smallest region that can be determined using only maximum speed. We assume that the individual regions given as input are optimal, which is trivially true for the range-based localization.

6.3.2 Algorithm Correctness

Theorem 2. *Given the maximum speed of a mobile node and t individually optimal regions, $A_1 \cdots A_t$, Algorithm 3 will produce optimal intersection regions $I_1 \cdots I_t$.*

Without loss of generality assume that $A_1 \cdots A_t$ are in time order. We will prove this theorem inductively on the number of range measurements for the online version of the localization algorithm. The base case is when there is only a single range measurement. Line 3 implies $I_1 = A_1$ and we already know A_1 is optimal.

Now inductively assume that intersection regions $I_1 \cdots I_{t-1}$ are optimal. We must now show that when we add region A_t, I_t is optimal and the update of $I_1 \cdots I_{t-1}$ maintains optimality given this new information. Call these updated intersection regions $I'_1 \cdots I'_{t-1}$.

First we will show that the new intersection region, I'_t, is optimal. Line 6 of the localization algorithm is

$$I'_t = \mathbf{Grow}(I_{t-1}, s\triangle t) \cap A_t. \tag{6.1}$$

The region $\mathbf{Grow}(I_{t-1})$ contains all possible locations of the mobile node at time t ignoring the measurement A_t. The intersection region I'_t must contain all possible locations of the mobile node as it is the intersection of two regions that constrain the location of the mobile node. If this were not the case, then there would be some point p which was not in the intersection. This would imply that p was neither in I_{t-1} nor A_t, a contradiction as this would mean p was not reachable. Additionally, all points in I'_t are reachable as it is the intersection of a reachable region with another region. Therefore, I'_t is optimal.

Finally we will show that the propagation backwards, line 9, produces optimal regions. The propagation is given by

$$I'_j = \mathbf{Grow}(I'_{j+1}, s\triangle t) \cap A_j \tag{6.2}$$

for all $1 \leq j \leq t-1$. The algorithm starts with $j = t-1$. We just showed that I'_t is optimal, so using the same argument as above I_{t-1} is optimal. Applying this recursively, all $I_{t-2} \cdots I_1$ are optimal. Q.E.D.

6.3.3 Computational Complexity

Algorithm 3 has both an inner and outer loop over all regions which suggests an $O(n^2)$ runtime, where n is the number of input regions. However, `Grow` and `Intersect` also take $O(n)$ time as proven later by Theorem 3. Thus, overall, we have an algorithm which runs in $O(n^3)$ time. We show, however, that we expect the cost of `Grow` and `Intersect` will be $O(1)$, which suggests $O(n^2)$ runtime overall.

The runtime can be further improved by noting that the correlation of the current measurement with the past will typically decrease rapidly as time increases. In

Section 7.4.1 we show in simulation that only a fixed number of steps are needed, eliminating the inner loop of Algorithm 3. Thus, we can reduce the complexity of the algorithm to $O(n)$.

The range-only instantiation of Algorithm 3 is obtained by taking range measurements to the nodes in the sensor fields. Let $A_1 \cdots A_n$ be the circular regions formed by n range measurements. $A_1 \cdots A_n$ are individually optimal and as such can be used as input to Algorithm 3. We now prove the complexity of localization regions is worst case $O(n)$. Experimentally we find they are actually $O(1)$ leading to an $O(n^2)$ runtime.

Theorem 3. *The complexity of the regions formed by the range-only version of the localization algorithm is $O(n)$, where n is the number of regions.*

The algorithm intersects a grown intersection region with a regular region. This will be the intersection of some grown segments of an annulus with a circle (as shown in the Figure 6-3). Let regions that contain multiple disjoint sub-regions be called *compound regions*. Since one of the regions is always composed of simple arcs, the result of an intersection will be a collection of circle segments. We will show that each intersection can at most increase the number of circle segments by two, implying linear complexity in the worst case.

Consider Figure 6-3. At most the circular region can cross the inner circle of the annulus that contains the compound region twice. Similarly, the circle can cross the outer circle of the annulus at most twice. The only way the number of sub-regions formed can increase is if the circle enters a sub-region, exits that sub-region, and then reenters it as illustrated in the figure. If any of the entering or exiting involves crossing the inner or outer circles of the annulus, then it must cross at least twice. This means that at most two regions could be split using this method, implying a maximum increase in the number of regions of at most two.

If the circle does not cut the interior or exterior of the annulus within a sub-region then it must enter and exit though the ends of the sub-region. But notice that the ends of the sub-regions are grown such that they are circular, so the circle being

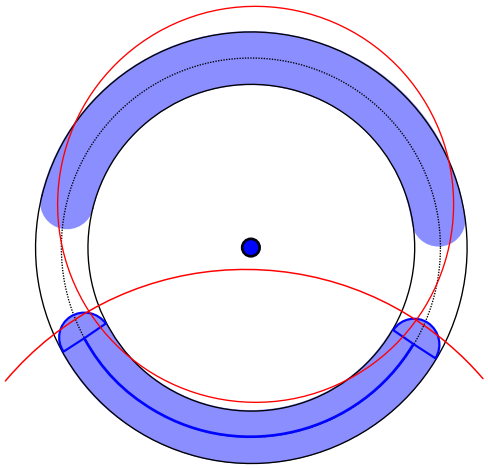


Figure 6-3: An example compound intersection region (in blue) and some new range measurement in red. With each iteration it is possible to increase the number of regions in the compound intersection region by at most two.

intersected can only cross each end twice. Furthermore, the only way to split the subregion is to cross both ends. To do this the annulus must be entered and exited on each end, implying all of the crosses of the annulus have been used up. Therefore, the number of regions can only be increased by two with each intersection proving Theorem 3.

In practice it is unlikely that the regions will have linear complexity. As seen in Figure 6-3 the circle which is intersecting the compound region must be very precisely aligned to increase the number of regions (note that the bottom circle does not increase the number of regions). In the experiments described in Chapter 7 we found some regions divided in two or three (e.g. when there are only two range measurements, recall Figure 6-1). However, there were none with more than three sub-regions. Thus, in practice the complexity of the regions is constant leading to $O(n^2)$ runtime.

Chapter 7

Underwater Robot Localization: Simulations and Experiments

7.1 Introduction

Chapter 6 introduced the underwater mobile robot localization algorithm. This algorithm is a minimalist, geometrically-inspired algorithm that only relies on knowledge of an upper bound on the mobile robot speed and periodic, asynchronous range measurements to static nodes. This chapter analyzes the algorithm in extensive simulations, considers extensions of the algorithm that enable an implementation on AquaNodes and Amour, and presents the results of experiments in lakes, rivers, and oceans.

The simulator gives visual and numeric feedback on the performance of the underwater mobile robot localization algorithm. We use the simulator to verify the algorithm developed in Chapter 6. In particular, we validate the intuition discussed in Section 6.3.3 that each new measurement requires only $O(1)$ time to localize the robot. This is because measurements only need to be propagated back a fixed amount of time, and the complexity of the localization regions is constant. In addition, we use the simulator to examine the effect of changing the speed bound and number of static nodes. Finally, we explore the impact of measurement noise and the rate the mobile node obtains ranges.

Theory	Practice
Ranges are exact	Ranges have noise (non-Gaussian)
Range measurements are precise	Speed of sound variations cause scaling in ranges
Static nodes are exactly localized	Static node positions have errors
Ranges received from random nodes	Often only get ranges from nearby nodes

Table 7.1: Comparison of assumptions from the theory and those found in practice.

We also implemented the underwater localization algorithm on the acoustic modem that is used in the AquaNodes and on our underwater robot, Amour. This implementation is different from the simulator in that it is a discretized, grid-based implementation. This implementation is compiled both on a computer for testing and post-processing data and on the acoustic modem to run in real-time on the system. We have performed numerous experiments using our sensor network system and underwater robot to verify the localization algorithm on real datasets and in-situ.

This chapter starts by discussing what it takes to go from algorithmic theory to a practical implementation in Section 7.2. We then implement the simulation in Section 7.3. This is followed by a detailed parametric analysis in Section 7.4. Next, we describe the implementation of the underwater mobile robot localization algorithm on our hardware platform in Section 7.6. Finally, we present results of experiments in lakes, rivers, and oceans in Section 7.7.

7.2 From Theory to Practice

Moving from theory to practice required the development of different algorithms to match the different assumptions that hold true in simulations versus in the physical world. We implemented two different versions of the theoretical Algorithm 11 described in Chapter 6. The first is a region-based algorithm that we use as a simulator and to post-process collected data. This algorithm was developed as part of our Master's thesis [31]. The second is a grid-based implementation that runs on the underwater robot, which was developed for this thesis.

Table 7.1 compares some of the assumptions of the theory with the challenges that the real-world implementations must handle. The theory assumes that the ranges

from the static nodes are exact and that the locations of the static nodes are known. In practice, the range measurements will have non-Gaussian noise. In addition, the speed of sound in water depends on the salinity, pressure, and temperature of the water. As such, it may vary during or between experiments, causing a systemic scaling error in range information. Finally, we determine the locations of the static nodes by implementing another algorithm that uses range information or alternatively by surfacing to use GPS. Both of these methods have errors that effect the mobile robot localization algorithm.

We implemented two different versions of Algorithm 11: a region-based and a grid-based version. The region-based implementation is nearly a direct implementation of the algorithm in Java. It has a user interface and is able to simulate range data from a network of static nodes. In addition, the region-based simulator has inputs to replay real data collected on the robot and underwater sensor network. The second version is a grid-based version of the algorithm implemented in C. The grid-based implementation runs on a PC to replay collected data. In addition, the grid-based version runs on the acoustic modem on the underwater robot, enabling real-time operation of the localization algorithm.

Table 7.2 summarizes the characteristics of the theory and the two different algorithms. The preferable characteristics are highlighted in bold. The actual runtimes of the implementations are less than that dictated by theory due to limited measurement propagation windows and the constant complexity of the localization regions in practice. The grid-based algorithm requires extra memory to store the grid. The size of the grid then limits the overall area that the algorithm can be cover. In addition, the grid limits the localization resolution of the algorithm. In practice, a few megabytes of memory suffices for high resolution localization using the grid-based algorithm.

The grid-based algorithm has a number of advantages over the region-based implementation. First, it is extremely easy to implement arbitrary regions in the grid-based algorithm. Implementing a new region in the grid-based implementation only requires implementing a "drawing" method for the new type of region. This method also needs to have the ability to grow the region to account for the motion of the mobile node.

Characteristic	Theory	Region-Based Alg.	Grid-Based Alg.
Simulate networks	N/A	**Yes**	No
Replay collected data	N/A	**Yes**	**Yes**
Run on robot (real-time)	N/A	No	**Yes**
Runtime for new range	$O(n^2)$	**O(1)**	**O(1)**
Memory requirements	**O(n)**	**O(n)**	O(grid size + n)
Cover large area	**Unlimited**	**Unlimited**	Memory-limited
Localization Resolution	**High**	**High**	Grid-limited
Different regions	**Easy**	Hard	**Easy**
Initialization	**Easy**	**Easy**	**Easy**
Outlier rejection	None	Pre-filter	**Implicit**
Bad data recovery	Never	Hard	**Easy**
Adding dead-reckoning	Hard	Hard	**Easy**
Code portability	N/A	PCs	**PCs + Embedded**

Table 7.2: Summary of the differences between the theory and the practical implementations of the algorithm. The region-based implementation is used in the simulator and the grid-based implementation is used on the robot. The value n is the number of ranges. Preferred characteristics are **bold**.

The region-based implementation requires these, but it also needs a method to intersect the new region with all other region types. To not impact runtime, care has to be taken to avoid intersection methods that cause the complexity of the regions to grow quickly.

In addition, it is easy to initialize localization, it rejects outliers implicitly, and it easily recovers if a long sequence of bad data puts it in a bad state. The theory does not handle outliers and the region-based algorithm must filter range measurements before using them. A bad state may not be detected for some time with the region-based implementation and the only course of action is to reinitialize the algorithm.

In the grid-based algorithm allows easy addition of dead-reckoning information, whereas adding dead-reckoning information to the region-based algorithm adds significant code complexity. Finally, the grid-based algorithm is written in POSIX C and as such is portable to most all operating systems and most embedded systems (such as those found on AMOUR). The region-based algorithm is portable to various PCs as it is written in Java, however, few embedded systems contain full Java Virtual Machines.

7.3 Simulation Implementation

We have implemented the localization algorithm in a simulator we designed. In this section we will discuss the design of the simulator and illustrate it in action. In the next section we study the effect of various parameters on the algorithm.

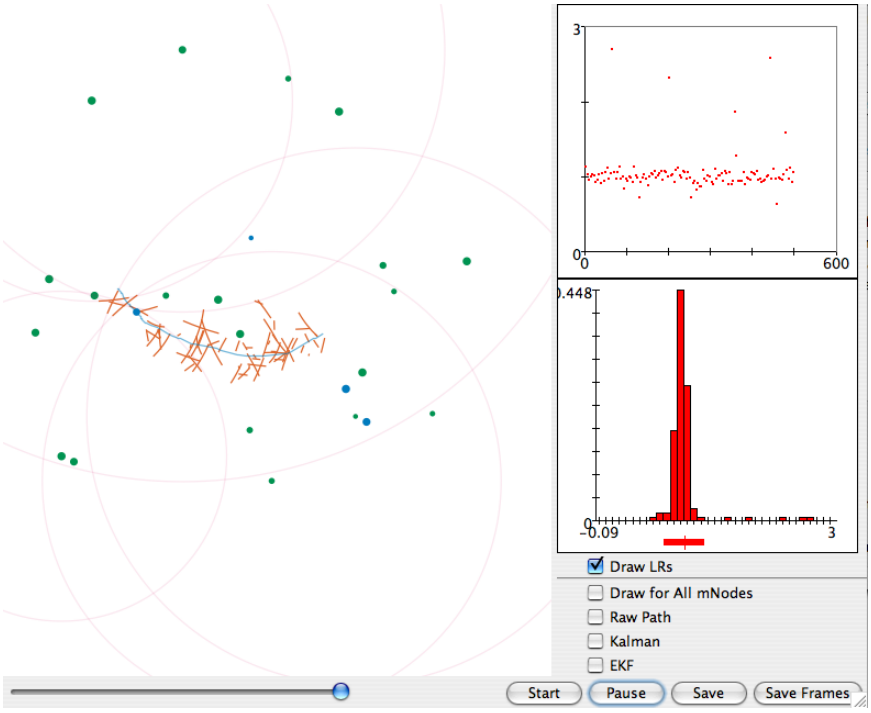


Figure 7-1: The simulator. A mobile node can be selected and its path drawn using a mouse. The plots at right show the probability distribution of the received ranges.

The simulator is a Java application designed to give visual and numerical feedback on the performance of the localization algorithm. Figure 7-1 shows the simulator in action. A mobile node is selected by clicking on it. By dragging the mouse the desired path for the selected mobile node is drawn on the screen.

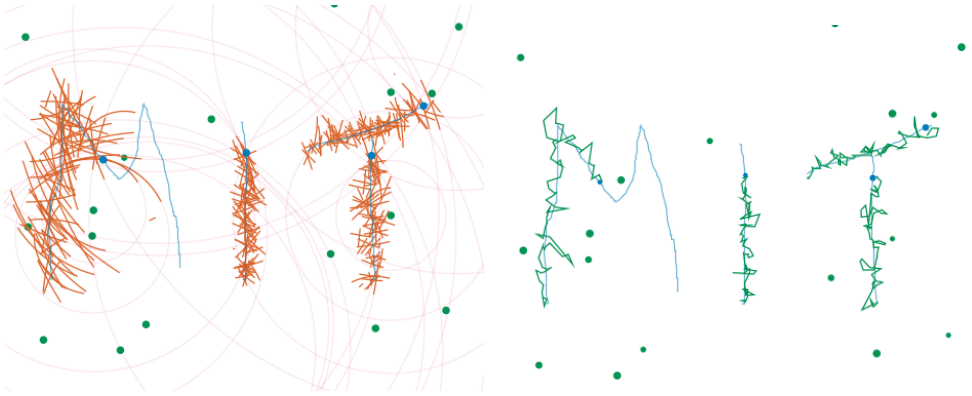
The received range measurements can be customized to have particular error characteristics. The graphs on the right hand side record the error in the measurement being fed into the algorithm. The buttons and slider on the bottom allow the simulation to be paused, started, rewound, or saved. Additionally, there is a button which enables the recording of the sequence to a video.

On the right hand side there are also a number of check boxes that allow the user to display the recovered localization regions, an estimate of the path, a Kalman filtered version of the path, or an extended Kalman filtered path.

The simulator region-based implementation and is implemented in such a way that makes it easy to try different types of inputs into the algorithm. The only methods that needs to be implemented for different inputs are `intersect` and `grow` methods, which tells the simulator how to perform the growing and intersecting needed for the localization algorithm. We implemented range-only and angle-only versions, although we will only discuss the range-only setup.

The localization algorithm that we use in the simulator is almost a direct implementation of Algorithm 11 discussed in Chapter 6. The main difference is that we do not propagate information from a new measurement all the way back. Instead, we only propagate it back a fixed number of steps. This, combined with the constant complexity of the regions, allows the algorithm to run in constant time per update. The main loop of the localization algorithm is shown below:

```
private void updateLR(LocalizationRegion l){
  int nt = l.getTime();
  //Forward propagation
  LocalizationRegion reg = lr.getPrevious(l);
  l.intersect(reg,
              getMaxSpeed(reg.getTime(),nt)
              *(nt-reg.getTime()));
  //Back propagation
  for(LocalizationRegion reg : lr){
    /* only use it if it is a fairly recent measurement */
    if(nt - reg.getTime() <= pastTimeToProcess){
      reg.intersect(l,
                    getMaxSpeed(reg.getTime(),nt)
                    *(nt-reg.getTime()));
    }
  }
}
```



(a) (b)

Figure 7-2: Multiple mobile nodes drawing MIT. At left, the raw regions. Right, the recovered path using the center of each region.

Figure 7-2 shows the results of a simulation with multiple nodes moving through the field of static nodes. This example uses a variety of mobile node speeds ranging between .3m/s to .6m/s. The upper bound on the speeds of the mobile nodes was set fairly high to nearly 2.0m/s. Random measurement errors of up to 4% were used. Notice that all of the regions are fairly simple regions. We never found a region with a complexity greater than three in all of our simulations. A full analysis of the range-only system is presented in Section 7.4.

7.4 Simulation Analysis

In this section we explore the effect of various parameters on the performance of the range-only implementation of the mobile robot localization algorithm. We measured the performance by looking at the average difference in the actual location of the mobile node to that of raw recovered path. Unless otherwise noted we use the exact range measurements and did not introduce any noise. Each of the data points was obtained by averaging together the results of multiple (typically 8) trials, each with a different pseudo-random path. In all cases the paths stayed nearly within the convex-hull of the static nodes. The static nodes were randomly distributed over a 300 meter by 300 meter area.

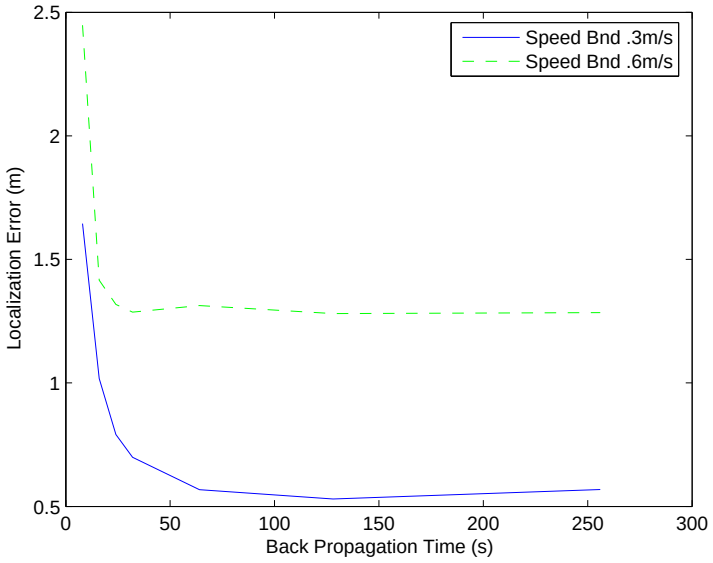


Figure 7-3: Effect of the back propagation time on localization error with two different speed bounds. This data shows that increasing the back propagation time to more than 75 seconds has little benefit.

7.4.1 Back Propagation

In this study we examine the effect of changing the amount of back propagation time. The results can be seen in Figure 7-3. In this experiment one mobile node traveled at .3m/s using an upper bound on the speed of .3m/s (solid line) and .6m/s (dotted line). There were twenty static nodes and a range was taken to a random static node every 8 seconds.

The results shown in Figure 7-3 show that the back propagation helps significantly up to the point where the error levels off, however, not more after that point. The results are similar for a mobile node traveling at its maximum speed and for one traveling at half speed except that there is a large constant offset. From this plot it can be determined that a back propagation of 75 seconds is more than sufficient as after this point the localization error does not decrease. For nodes not moving at their maximum speed a back propagation of 25 seconds works well. This experiment shows that a constant number of back propagation steps can be used, and therefore the algorithm has a constant runtime per each new measurement.

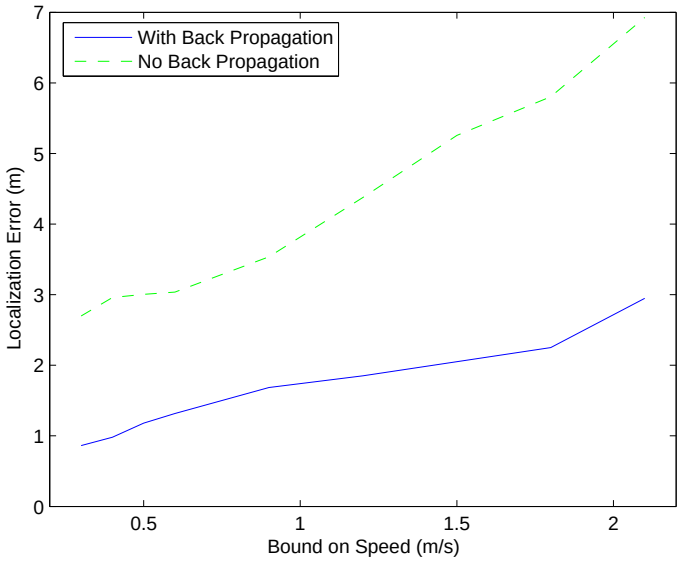


Figure 7-4: The effect of the upper bound on the speed of a mobile node moving at .3m/s using back propagation (solid) and not (dotted). The error increase is nearly linear, but when no back propagation is used there is a large initial offset.

Figure 7-4 shows a comparison of using back propagation (solid line) and using no back propagation (dotted line) while varying the upper bound on the speed. This shows that the error is more than 2 meters higher when not using the information gained from back propagation.

7.4.2 Speed Bound

Figure 7-4 shows how changing the upper bound of the mobile node speed effects the localization. Two lines are shown, one with and one without back propagation. For both, the real speed was .3m/s and the upper bound on the speed was adjusted. A range measurement was taken every 8 seconds to a random node of the 20 static nodes. The back propagation time used was 25 seconds.

As the upper bound on the speed increases, the error also increases. From the data collected this appears to be a fairly linear relationship implying that the errors will not be too bad even if the mobile node is not traveling at its maximum speed. The back propagation, however, is critical. With an upper bound on speed seven

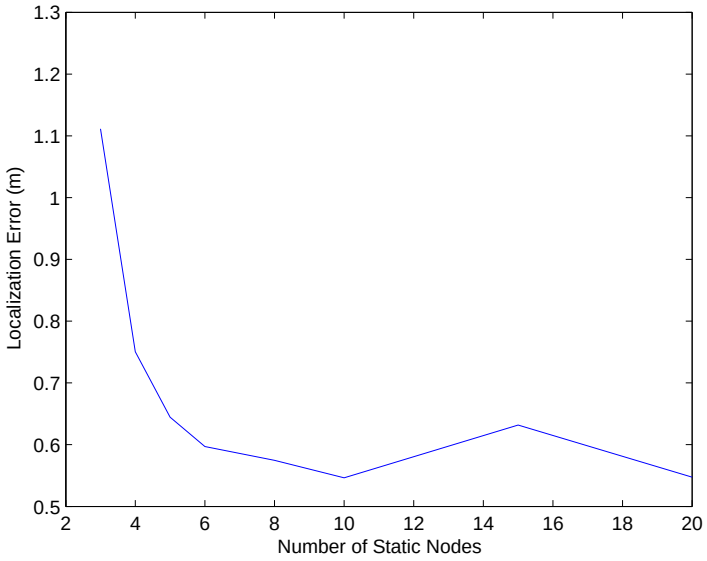


Figure 7-5: Adjustment of the number of randomly placed static nodes. Having fewer than six leads to larger error.

times larger than the actual speed we still achieve results comparable to that of the algorithm that does not use back propagation but knows the exact speed of the mobile node.

7.4.3 Number of Static Nodes

Figure 7-5 shows how the number of static nodes effects the localization error. For this experiment the mobile node was moving at its upper bound speed of .3 m/s. A range measurement was taken every 8 seconds and the measurements were propagated back 80 seconds. In this experiment any regions which contained disjoint arcs were not used in computing the error. This means that the error was actually much worse when there were few nodes than is reported in the figure.

The main configuration that leads to large errors is the singular configuration where the nodes were nearly collinear. The probability of having more than six nodes randomly deployed in a line is small. Another source of error is when ranges were only obtained from a single pair of nodes (which happens with higher probability with few nodes), then there would be an ambiguity, which leads to a large error. To avoid

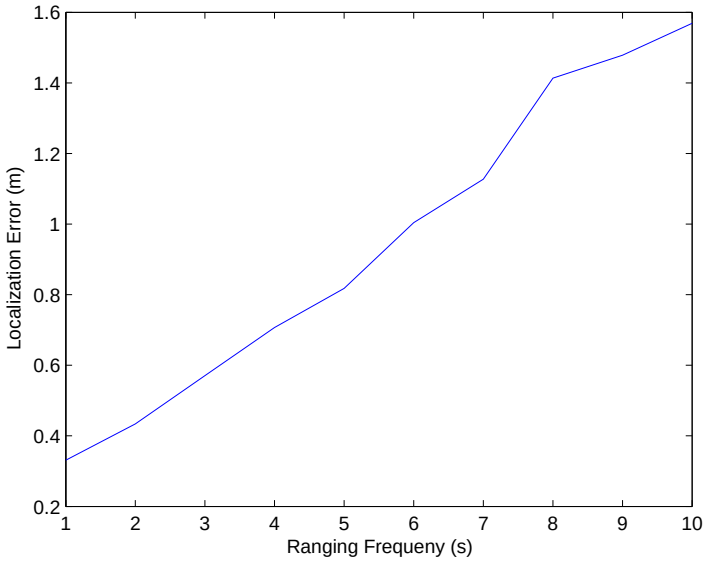


Figure 7-6: Experiment showing the effect of changing the frequency of the ranging. The relationship is linear and indicates that decreasing the ranging frequency is always preferable.

this situation a large back propagation time of 80 seconds was used. This was chosen to be larger than the maximum back propagation we found useful in Section 7.4.1. As can be seen in the figure six or more static nodes produced good results. Thus, unless the static nodes can be carefully deployed, it is important to have at least six static nodes.

7.4.4 Ranging Frequencies

Figure 7-6 shows the effect of changing the frequency of ranging. Experiments were conducted in which the ranging frequency was varied between ranging every second to ranging every ten seconds. Each time a range was obtained from a randomly selected node. A speed of .3m/s and a bound of .6m/s were used for the mobile node in this experiment. There were 20 static nodes and the measurements were propagated back to the four previous measurements.

The figure shows that the relationship is very linear. As the ranging frequency increases, the error decreases. Thus, it is always desirable to get as high a rate of measurements as possible.

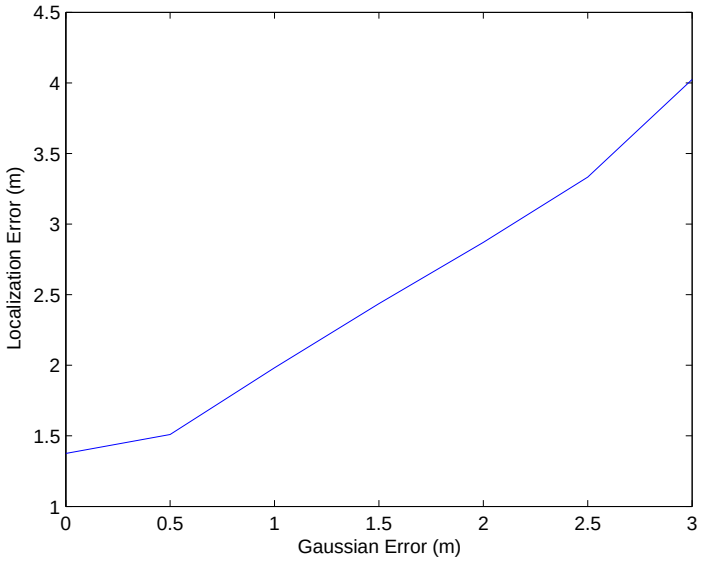


Figure 7-7: The effect of Gaussian noise in the measurements. The localization error is approximately one plus the Gaussian error over this range.

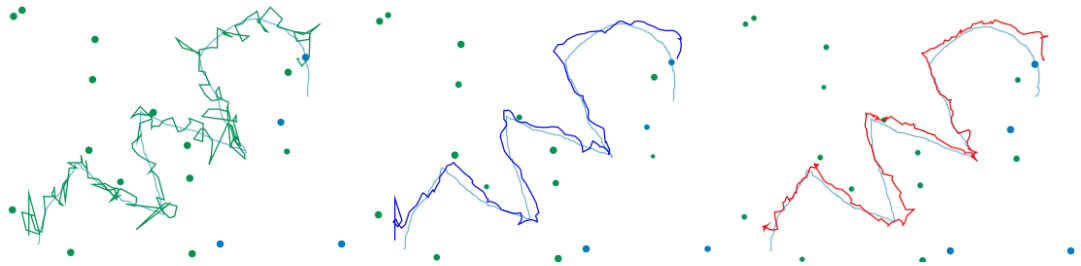
7.4.5 Gaussian Error

Figure 7-7 shows the effect of Gaussian error in the measurements on the localization. In practice, the error is not Gaussian, however, this serves as a reference to compare the impact of Gaussian noise on the algorithm. The mobile node had a speed of .3m/s and an upper bound speed of .6m/s. It was moving through a field of 20 static nodes with range measurements taken every 8 seconds. The measurements were propagated back over 30 seconds.

Even with the error in the measurements the algorithm performs well. The localization error scales linearly with the measurement error. With three meter average measurement error, the localization error is four meters. As the error is one meter when there is no Gaussian error we can say that the error, Err, will be approximately

$$Err = Err_G + 1 \text{ meters},$$

where Err_G is the Gaussian error in the measurement.



(a) (b) (c)

Figure 7-8: (a) Raw recovered path. (b) Kalman filtered raw path. (c) EKF path.

7.4.6 Post Filters

There are a number of methods to compare the results of the algorithm to the true path of the mobile node. The first is to just determine if the true path is within the found region. This is useful, although for navigation purposes it is often useful to report a single point as the current location. To do this we implemented three different methods for recovering the path.

The first we call the "raw path." This is the path formed by connecting the midpoints of the arcs formed by the localization algorithm. The second is using a Kalman filter as a filter on the raw path. This works, although it is not ideal as the range measurements introduce a non-linearity into the system. To better account for this we also implemented an extended Kalman filter which takes as input the regions found.

Figure 7-8 show a comparison of these three. Having a post-filter certainly smoothes the path, however, these filters require tuning for each particular setup depending on factors such as the speed, acceleration, measurement certainty, etc. We found the tuning of these to be rather sensitive and difficult to optimize for even a single run. For instance, in order to get a smooth path while the mobile node was going straight required reducing the acceleration to the point where when the mobile node turned the filter would take too much time to compensate, leading to overshoots. This effect can be seen in Figure 7-8. Due to these effects it is difficult to use these filters on real-world systems where tuning online can be difficult or impossible.

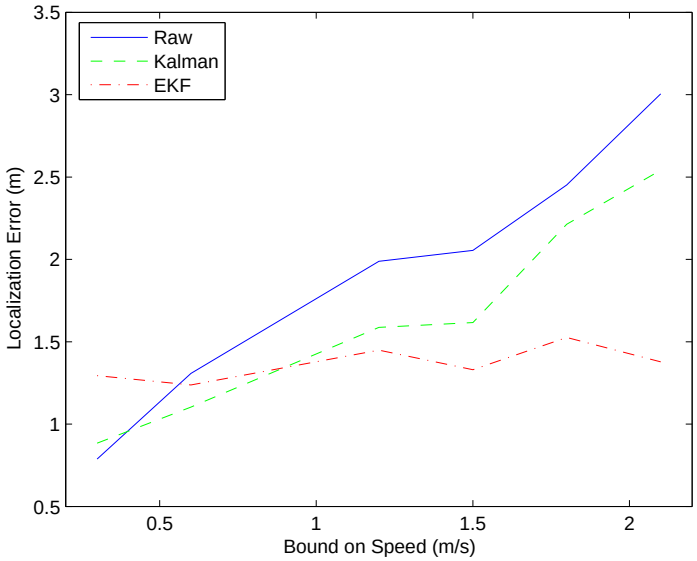
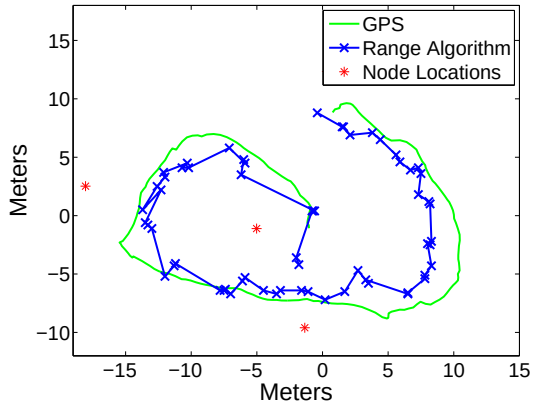


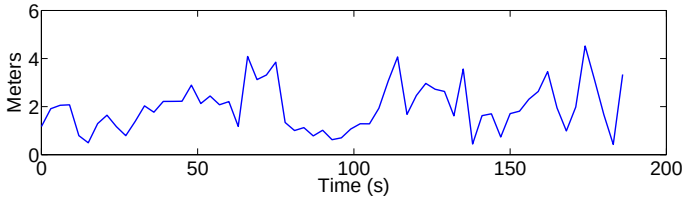
Figure 7-9: The errors of different post filters when changing the upper bound of the mobile node speed. The mobile node traveled at .3m/s. The Kalman filter is typically better than the raw data. The EKF performs best when the upper bound on the speed is more than twice the real speed

Figure 7-9 shows the results of using the three different post filters. The mobile node had a speed of .3m/s and the upper bound on the speed was varied. The mobile node was moving through a field of 20 static nodes and ranges were taken every 8 seconds. The measurements were propagated back over 30 seconds. The filters were tuned by hand to produce good results when the upper bound on the speed was 1.2m/s.

The figure shows that when the upper bound equals the real speed the raw recovered path has the lowest error, while the EKF produces the worst results. The error produced by the EKF, however, tends to remain fairly constant throughout as indicated by the near horizontal line. The regular Kalman filter performs a little bit better than the raw data, but not significantly. The EKF outperforms the regular Kalman filter due to the fact that it is able to take into account the non-linearities of the errors. The regions found are arcs, which are directly inputed into the EKF. The Kalman filter must make use of the point produced by the raw path filter with some linear variance which does not properly represent the information from the algorithm.



(a)



(b)

Figure 7-10: Localization algorithm and GPS track (a) and the difference between them (b).

7.5 Extended Kalman Filter and Particle Filter

In this section we implement an extended Kalman filter (EKF) and a particle filter (PF) using the same set of assumptions that the localization algorithm uses. These assumptions are that the locations of the static nodes are known, periodic ranges are obtained to the static nodes, and we only know an upper bound on the speed of the mobile node. We then compare the localization algorithm, EKF, and PF using data collected in the Moorea experiment detailed in Section 7.7.2.

The Kalman filter (KF) is a provably optimal recursive state estimator when the processes being measured and estimated are linear with Gaussian noise characteristics. Non-linear systems are handled by extending the KF with an extended Kalman filter. The EKF is not provably optimal, however, it often performs well. See the paper by Welch and Bishop [106] for a nice introduction and details on the KF and EKF.

The EKF we implement assumes that the position of the node stays constant,

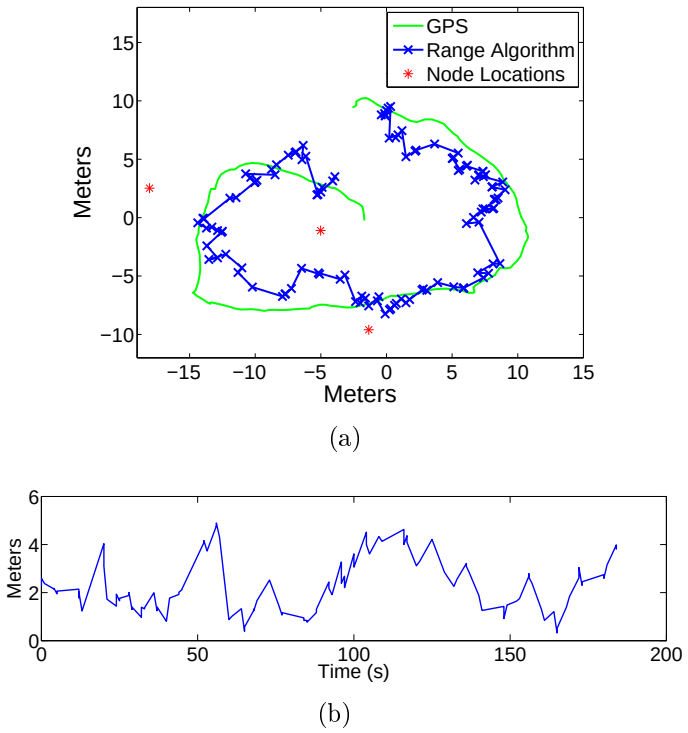


Figure 7-11: EKF algorithm and GPS track (a) and the difference between them (b).

but the variance in the position grows based on the maximum possible speed of the mobile node. In other words, the state projection has no control input, it just takes the previous state as the next state or $x_k = x_{k-1}$. The position estimate is updated every time a range measurement is received by defining the measurement, z_k, as the distance from x_k to the beacon. Other EKFs can be defined that incorporate information on the mobile node's velocity or acceleration, however, this would make use of information beyond the maximum speed assumption we use.

The particle filter, also known as a Monte Carlo method works by defining a set of "particles." Each particle has a location and a probability of being at that location. With each new measurement the probabilities are updated to incorporate the new information and only the best particles are kept. New particles are then derived from these particles to take into account the probable motion of the node.

The PF we implement is fairly straight forward. For each range measurement, the top 50 particles that are nearest to the measured range are kept. From each of these

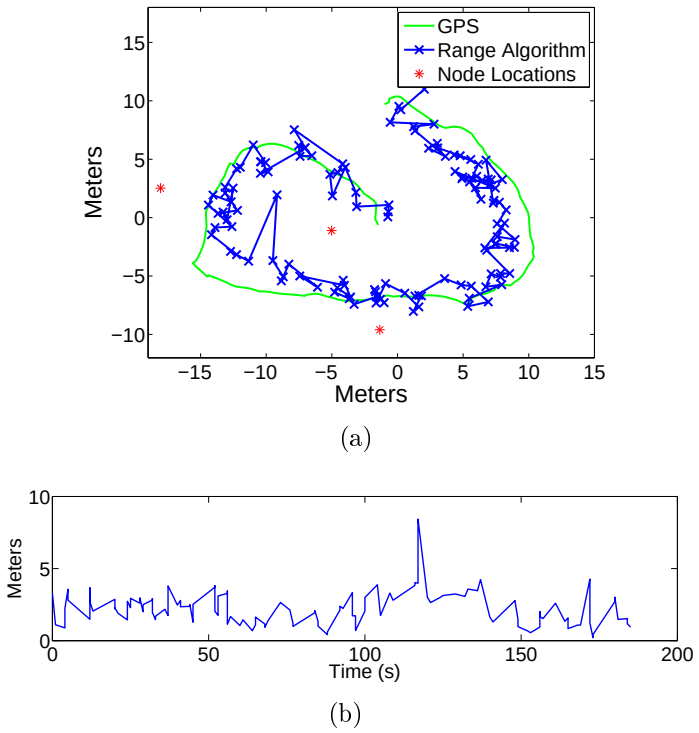


Figure 7-12: Particle Filter and GPS track (a) and the difference between them (b).

particles 50 new particles are created. Each new particle is randomly distributed within a circle with a radius around the original particle defined by the maximum distance the node could travel given its maximum speed. The centroid of all particles is then used as the estimated position.

Figure 7-10, 7-11, and 7-12 show the results versus GPS ground truth for the localization algorithm, EKF, and PF, respectively. The lower plot shows the error over the run. For the localization algorithm the average error was 1.97m with a maximum of 4.52m. The EKF average error was 2.30m with a maximum of 4.89m. And the PF had an average error of 2.21m and a maximum error of 8.43m.

All of the errors are relatively comparable. However, there are a number of challenges associated with the EKF and PF. One problem with the EKF is initialization. In this implementation we used the first point obtained by our localization algorithm to seed the EKF. However, this type of information may not always be available. In addition, bad measurements (with non-Gaussian noise) can cause the EKF to diverge.

Detecting a divergent EKF can be challenging and if it is detected a new position must be obtained to reinitialize the filter. Tuning the parameters of the EKF can also be challenging. The actual variance of the process and measurements are often not known accurately. Thus, it must be tuned to find the proper set of parameters.

Initializing the PF is easier than the EKF. The initial set of particles for the PF is particles laying on the circle defined by the first range measurement. However, picking which particle to use as the current state estimate is challenging. In this implementation we use the centroid of all particles, but this may bias the position improperly. Consider, for example, when there are particles over a half-circle, the centroid will be inside the circle, while all particles and most likely the true position will be on the circle. In addition, the PF can be thrown off by bad measurements by eliminating all particles near the true position of the node.

The comparison of the errors shows that our localization algorithm outperforms the EKF and PF in terms of average and maximum error for this dataset. In general, all of these methods have similar errors. Our localization algorithm, however, does not suffer from initialization and parameter tuning problems and tends to be more robust to bad measurements than the basic PF.

7.6 Hardware Implementation

We implemented the localization algorithm on the acoustic modem inside the robot. Alternatively, the algorithm could have been implemented on a mini PC or on the main AquaNode processor. We choose to implement the localization algorithm on the acoustic modem for a number of reasons. First, the algorithm requires ranges obtained by the acoustic modem. Implementing the algorithm on the acoustic modem eliminates the need to have other systems present to run the algorithm. Second, the acoustic modem has enough processing power and memory capacity to run the algorithm in under a second, allowing real-time, on-line operation. Using a PC would have been overkill and the AquaNode processor would not have been fast enough to allow real-time operation.

We typically house the acoustic modem within an AquaNode. The AquaNode provides logging of the raw communication, ranges, and location information from the acoustic modem as the acoustic modem lacks this capability. In addition, an AquaNode is easily attached to the side of an underwater robot to provide localization capabilities. The AquaNode logs and forwards location information to the underwater robot and provides an interface for the robot to communicate over the acoustic link.

In this section, we detail the implementation and operation of the underwater localization algorithm on the acoustic modem. The implementation is a discretized, grid-based implementation of the acoustic mobile robot localization algorithm. We then describe the user interface that allows quick and easy setup and testing of the algorithm.

7.6.1 Implementation Details

We implemented the acoustic localization algorithm on the acoustic modem using a discretized, grid-based method. We divide the region of operation into a grid. Each cell represents the likelihood that the robot is at that location. Keeping this representation in memory could be problematic for large regions or depths. Fortunately, however, the algorithm can be implemented in 2D since the robot has a pressure sensor that is used to precisely determine its depth. The range measurements are projected onto a plain at the depth of the robot to account for the foreshortening caused by the node and robot being at different depths.

The grid-based implementation of the algorithm has some advantages and disadvantages as compared to the region-based implementation used in the simulator. The grid-based algorithm has the following characteristics and advantages:

- Allows different region models to be easily implemented;
- Can integrate inertial or other navigation information;
- Better inherent outlier rejection;
- Implementation is in `POSIX C` for portability and microcontroller execution

Algorithm 4 Grid-based Localization Algorithm

```
1: procedure LOCALIZE(R_1 ··· R_t)
2:     clearGrid()
3:     drawOldPositionIfAvailable()
4:     for i = 1 to t do
5:         drawCircle(Grow(R_i,curretTime-time_i,maxspeed))
6:     end for
7:     location ← findCentroid(findMaxRegion())
8:     return location
9: end procedure
```

The region-based algorithm implemented in the simulator has the following characteristics and advantages:

- Does not require memory allocation for grid;
- Closer to theoretical algorithm

We choose the grid-based implementation mainly due to its ability to easily integrate inertial or other navigation information available from the robot.

The details of the grid-based algorithm are similar to the region-based algorithm. As each range is obtained, an annulus is overlaid on the grid, incrementing each bin in the grid by one. Areas with multiple, overlapping annuli have a larger value. The algorithm searches to find the bin with the largest value. This area indicates the current location of the underwater robot. As in the region-based simulation implementation, to account for robot motion, the algorithm grows older annuli based on the maximum possible speed. In addition, if estimates of previous positions are important, the algorithm propagates new measurements back to refine previous position estimates. This method is related to other Markov-based localization methods [23, 48, 61].

Algorithm 4 shows the pseudo-code for the grid-based localization algorithm. The localization algorithm takes as parameters ranges $R_1 \cdots R_t$. These are the t ranges received over the past time window. In practice, we use ranges from approximately the last 60 seconds (as this was determined in Section 7.4.1 to be the proper back propagation time). The next step in the algorithm is to clear the grid (setting each bin to zero). After clearing the grid, the algorithm "draws" the previous location estimate on the grid. This is the region found the last time the algorithm ran, grown to account

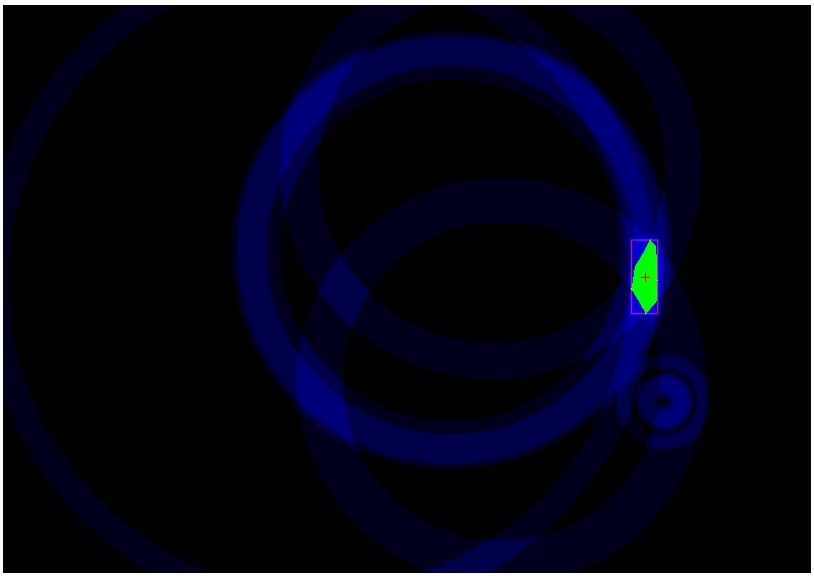


Figure 7-13: Snapshot from the localization algorithm. Note that a range from one of the nodes was far off. Green is the estimated position.

for motion of the robot. If the algorithm knows information on the direction of travel of the robot, it can incorporate this information into the growing of the previous location region.

Next, the algorithm loops over all of the received ranges and draws each annulus on the grid. Each annulus is grown to account for motion of the robot since that range was received. When drawing a region on the grid, each point is incremented by one. Finally, the algorithm searches the grid for the bins with the maximum value. This region is stored for use in the next iteration of the algorithm. The algorithm returns the centroid of the region as the point estimate of the robot location.

Figure 7-13 is a snapshot of an image taken from the algorithm. The blue regions are the annuli drawn based on the grown ranges. Note that some are brighter than others. These are multiple annuli drawn on top of each other. The green, highlighted region is the current location estimate. In the bottom right, there are two ranges from a node that do not intersect with the green region. These are bad range measurements, which the algorithm easily handles as there are more good measurements that intersect at the true location of the robot.

The algorithm automatically rejects poor range measurements as long as there are a sufficient number of good ranges. In addition, by placing the grown previous location estimate on the grid, the algorithm can handle long periods of only receiving a few range measurements. Figure 7-14 shows a snapshot when only ranges to two nodes are available. By carrying forward the previous location estimate, the algorithm disambiguates the two regions of intersection and selects the proper location.

Notice that Algorithm 4 clears the grids and redraws all of the grown regions each time. An alternate approach is to grow the regions based on those previously drawn on the grid. One way to implement this is to use a mask to process the grid. For instance, each bin would be the sum of all neighboring bins in the next step (depending on the robot speed). This can save some computation time needed to redraw each of the annuli. This works for a while; however, after many time steps the discretization introduces aliasing effects (grown circles start to look like diamonds). In practice it is more accurate to redraw each of the grown annuli each time.

The grid-based implementation varies somewhat from the version of the algorithm discussed in Chapter 6 and region-based simulation implementation. The grid-based method requires a large fixed array of memory for the grid storage. This may be problematic when large areas must be covered with high resolution. The grid method, however, intrinsically rejects outliers without additional filtering needed by the simulation implementation. Finally, the grid-based method can easily incorporate further information about the direction and speed of the robot if that information is available.

7.6.2 Operation Details

Before the underwater localization algorithm is run, the robot must know the locations of the static nodes. In our system, we first run a static localization algorithm to determine the locations of the static nodes as discussed in Section 6.2.2. We extended an algorithm by Moore *et al.* [71] for use in the underwater environment. This algorithm requires as many of the inter-node ranges as possible to accurately localize the static nodes. In this algorithm, if there are N nodes, approximately N^2 messages containing ranges are transmitted to share all the ranges. First, however, the nodes

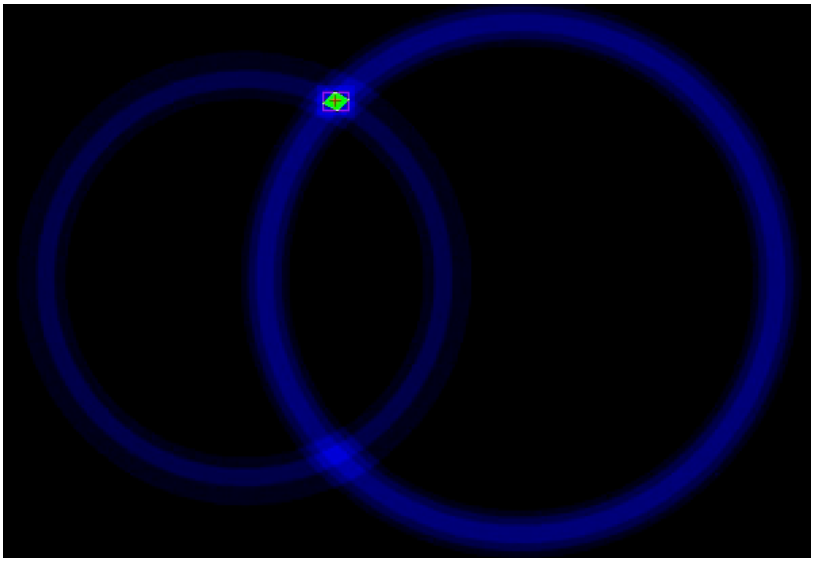


Figure 7-14: Snapshot from the localization algorithm when only two ranges are available. The algorithm is able to carry the previous location estimate to maintain a position estimate.

must obtain the pair-wise ranges. This also requires order N^2 messages as ranges are obtained by measuring round-trip communication time.

Using the acoustic modem's TDMA communication scheme, a message is transmitted at most every 4 seconds. For a network with 8 nodes this requires a minimum of $8 \times 8 \times 4 = 256$ seconds (or 4.26 minutes) to obtain ranges and also to transmit all ranges. This means that at a minimum the static localization algorithm requires about 9 minutes to initialize. However, on average, our acoustic modems have a packet success rate of about 50%. Thus, in the best case the algorithm needs 18 minutes of communication to initialize. In practice, due to asymmetric communication, a deployment time of over 30 minutes is needed before a network of 8 nodes can run the static localization algorithm.

Once the static nodes have been localized, the mobile localization algorithm can run. The algorithm, which is implemented in `C` code, runs on the acoustic modem processor. In addition, we can compile the same code as a stand-alone application on a computer. This allows running the same set of code on recorded measurements to test different algorithm parameters. In addition, the stand-alone application allows

saving of the grid as an image (Figures 7-13 and 7-14). This makes the algorithm much easier to debug and test.

The acoustic modem is implemented on a 600MHz processor with 32MB of ram. At the physical layer, the modem operates at 300b/s with a packet length of 128 bits. Packets are transmitted and received in well under a second, but requires guard times (to reduce the impact of previous reflected packets) of over a second. During this time the acoustic modem is largely idle. We take advantage of this idle time to run the localization algorithm.

Using a grid of 80m by 80m with a resolution of 0.1m with each bin represented by 8 bits requires $800 \times 800 \times 8 = 5120000$ bits of storage, less then 1MB. This easily fits within the memory of the processor and the localization algorithm (Algorithm 4) typically takes less than a second to execute with this configuration.

7.6.3 User Interface

Figure 7-15 shows the user interface (UI) for the static and mobile localization code. The Java application runs on a computer connected to the AquaNode on the robot using a RS232, RS485, bluetooth, or 900MHz radio. In addition, when the robot is tethered with an Ethernet cable the UI runs by forwarding a serial port over Ethernet. The UI graphically displays the locations of all the nodes in the network as well as ranges to the currently selected node from other nodes. The user can rotate and scale the network picture to easily match the visual feedback to the physical layout. This enables quick development and debugging of the system.

In addition, the UI shows the current table of ranges and locations. A user can manually update the tables on the UI and the changes can be sent to the acoustic modem. This allows the user to experiment and run the algorithm with different configurations and setups, without having to have the system in the water. Finally, the UI allows the current configuration to be saved to a file and later reloaded.

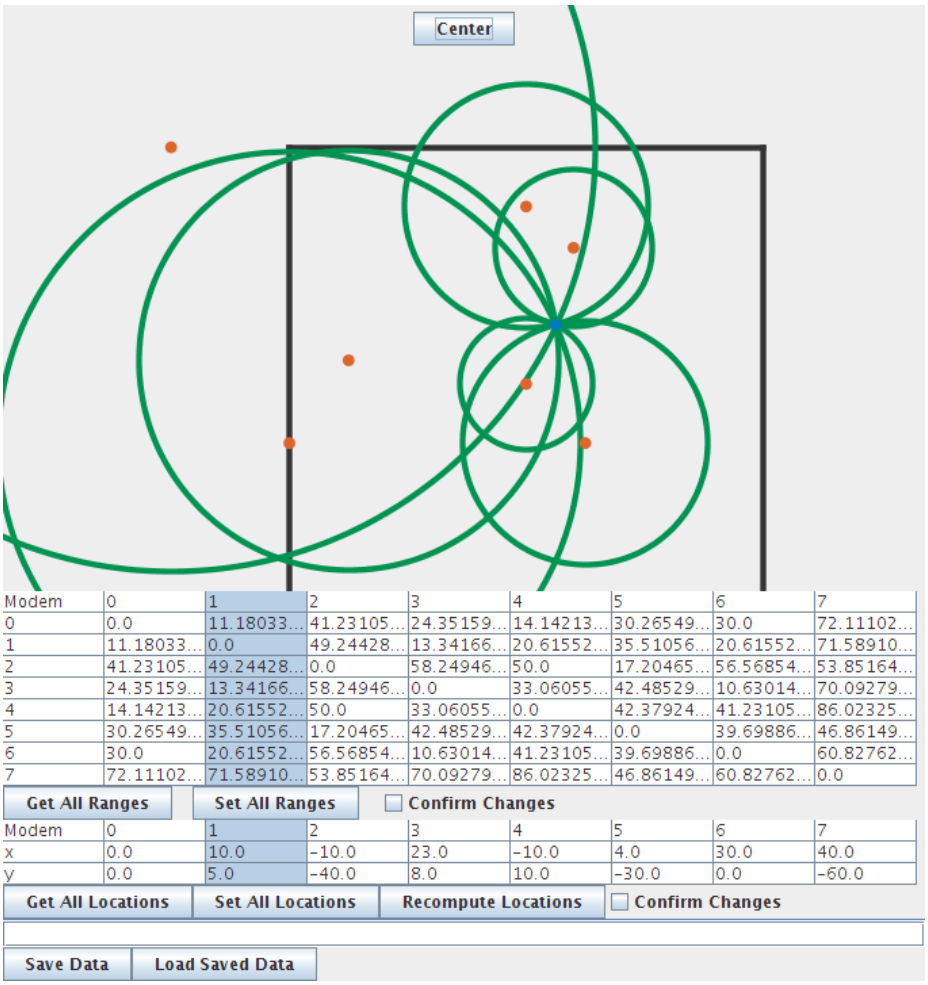


Figure 7-15: The localization user interface.

7.7 Experimental Results

We performed numerous experiments in rivers, lakes, and oceans. These experiments verify the performance and accuracy of the acoustic mobile robot localization algorithm. We tested the algorithm by running the underwater robot at the surface so that the results can be compared to GPS ground-truth. We tested the algorithm in the simulator using real data and the grid-based implementation on the acoustic modem.

For the simulator, we collected range information as the robot drove through a network of sensors. We input this data into the simulator and verified the results. This shows that the algorithm works with real-world data. We also used collected data to

test the grid-based implementation on a computer. These post processed tests verify that the grid-based algorithm performs as well as the simulator implementation. This is the same implementation that runs on the acoustic modem in real-time. Finally, we ran the algorithm online on the acoustic modem as the robot drove through a field of AquaNodes in numerous locations.

We have conducted robot tracking and localization experiments using Amour and up to six underwater sensor nodes at four different locations: (1) in an open bay in Moorea where the water had an average depth of 5m; (2) off a dock in Singapore where the water depth had an average depth of 5m; (3) in a Lake Otsego in upstate New York where the water had an average depth of 20m; (4) and in the Charles River in Cambridge, MA. In each case the robot traveled on the surface of the water in order to collect GPS position information, which we use as ground truth when comparing to the position information computed by the underwater sensor network. Each node was anchored in water at a depth of approximately 1 to 2m. The node was deployed by manually throwing it overboard at the approximate desired location. The exact location of the node was established by the underwater sensor network using the underwater static location algorithm. In each experiment, the robot traveled in the area covered by the sensor network, along different trajectories. In this section we present subsets of localization data collected for approximately 2 hours in Moorea, 7 hours in Singapore, 1.5 hour in Lake Otsego, and 2 hours in the Charles River. We have performed other similar experiments at these locations.

This section presents a selection of these results. We start with the results of the simulator using real data. We then show the results of the grid-based algorithm running on collected data. Finally, we present the results of the algorithm run online on the robot.

7.7.1 Real Data Region-Based Algorithm

We logged ranges and GPS data while moving the underwater robot at the surface through the field of static sensor nodes. This data was post-processed by the region-based simulator to recover the path of the robot traveling through the network and

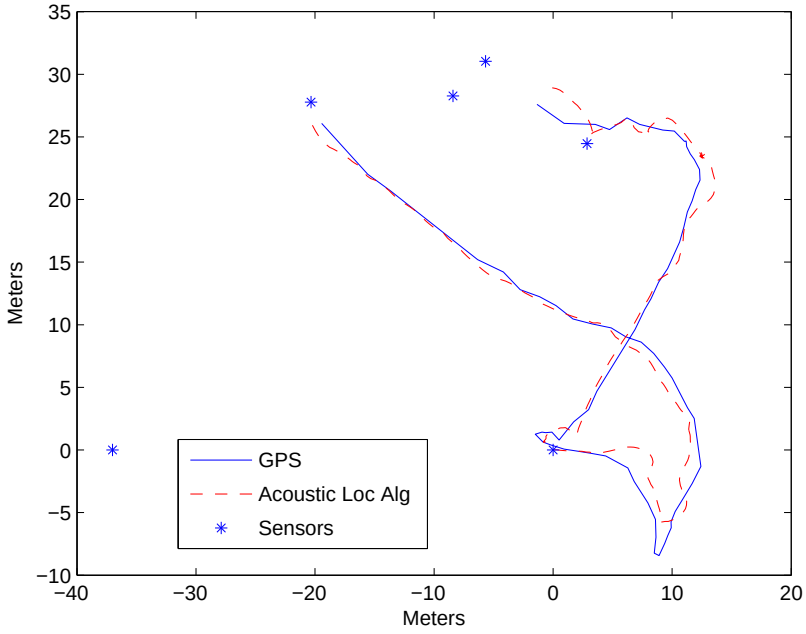


Figure 7-16: The GPS data (red) and acoustic localization path (blue) in Moorea.

compare it to the GPS path. The simulator uses a direct implementation of the localization algorithm discussed in Chapter 6

Figure 7-16 shows the results of an experiment performed in the ocean off the island of Moorea, French Polynesia for 2 weeks in June, 2006. In this experiment six nodes were placed and statically localized during the first 30 minutes of the experiment. The nodes were located in a region approximately 40m by 40m. The nodes were in 3m deep water located about 1m below the surface. The robot moved on the surface collecting GPS data and range information. The two tracks in the figure show 10 minutes of the path recorded by the GPS and the path computed by the acoustic localization algorithm. The average error was 0.2m with a maximum error of 2.75m. The average percent error based on the maximum range is 0.5%.[1]

Figure 7-17 shows the results of a similar experiment performed at Lake Otsego in New York State for a week in July, 2006. Four AquaNodes were deployed in a 70m by 80m region of water with an average depth of 20m. The nodes floated

[1]This is computed 0.2m/40m*100%, based on a maximum range of approximately 40m in the experiment.

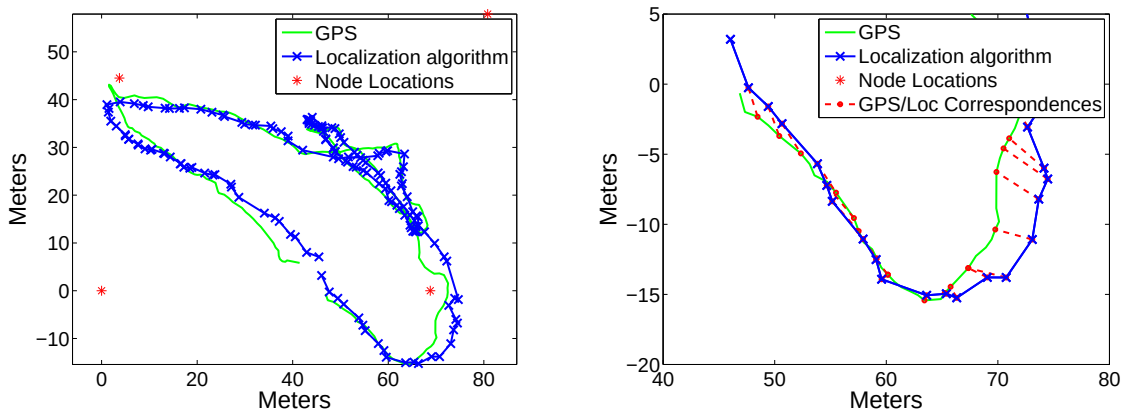


Figure 7-17: Localization algorithm and GPS track left from an experiment in Lake Otsego in New York. On right, zoomed in section showing the point correlations between the GPS and localization algorithm.

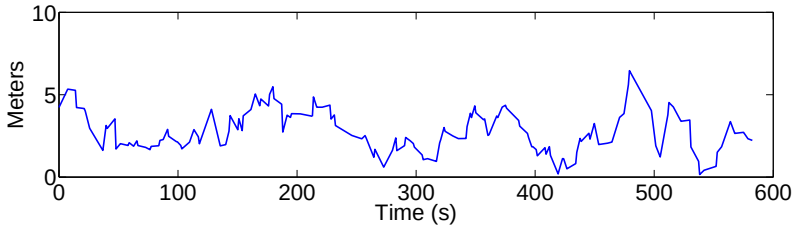


Figure 7-18: Plot of the error between the GPS and localization algorithm over time for the experiment in Lake Otsego.

between 3 and 5 meters below the water surface. The nodes were statically localized using the static localization algorithm during the first 30 minutes of the experiment. After static localization completed, the robot drove in the network for 1.5 hours. Ten minutes of results are shown in the figure. Figure 7-17(a) shows the GPS track and the results of the localization algorithm. Figure 7-17(b) shows a zoomed-in section illustrating the corresponding points from the localization algorithm and the GPS track. Figure 7-18 shows the error between the two over the experiment. The average error in this experiment was 2.74m with a maximum of 6.46m. The average percent error based on a maximum range of 80m is 3.4%. Note that this error is well within the expected GPS location noise.

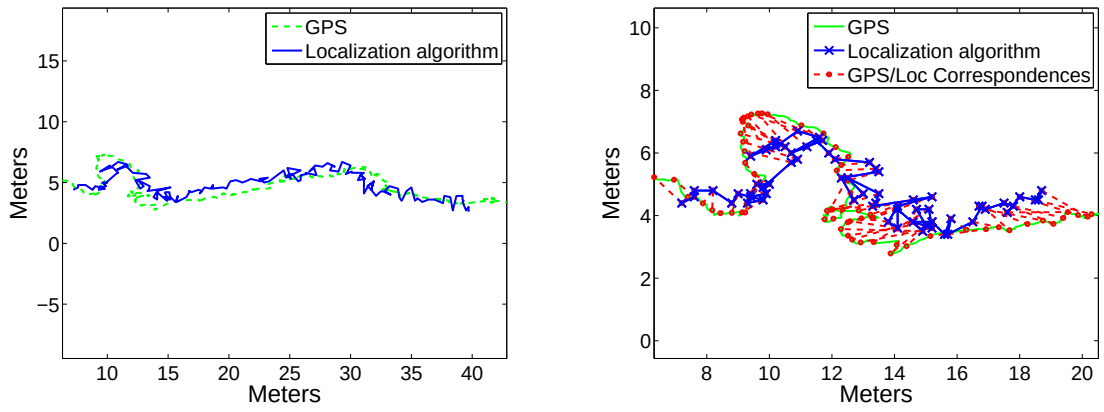


Figure 7-19: Results of the tracking algorithm in Singapore. (a) The localization algorithm track and GPS readings and (b) includes the lines indicating corresponding points between the two for a smaller section of the experiment.

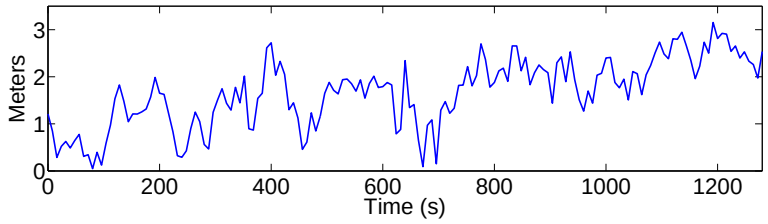


Figure 7-20: Error between the localization algorithm and GPS for the Singapore experiment.

7.7.2 Real Data Grid-Based Algorithm

In these experiments, the robot moved on the surface collecting both range measurement information and GPS location information. We ran the grid-based localization algorithm on the collected data to determine the location of the robot. This is the same code that can run online on the robot, but it was compiled to run on a PC. These experiments verify the correctness of the grid-based localization algorithm.

Figure 7-19 shows a data segment from the localization and tracking experiment conducted in Singapore in May 2009. In this experiment, we deployed six AquaNodes in static locations off a dock. The static sensor nodes self localized in an area of approximately 30 by 30 meters. The self-localization took approximately 30 minutes.

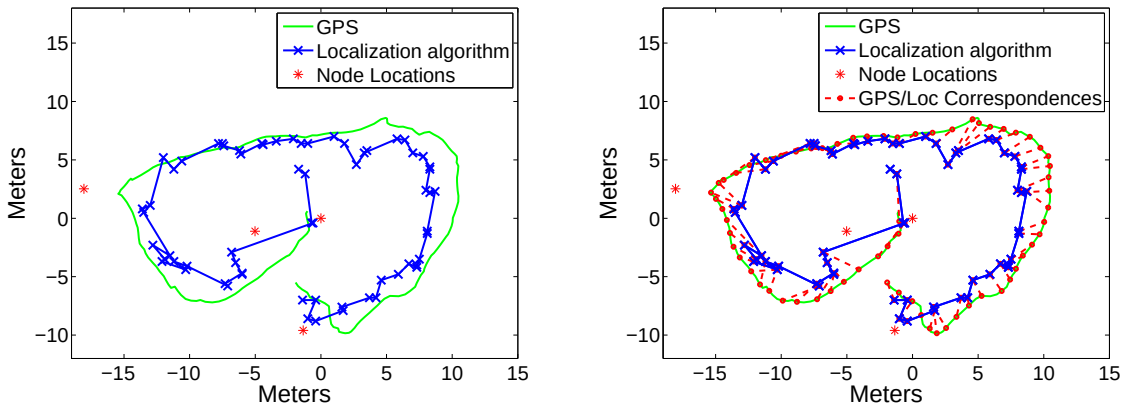


Figure 7-21: Results from an experiment in Moorea. (a) The localization algorithm and GPS tracks and (b) the same figure with lines drawn to indicate the correspondence between the two methods.

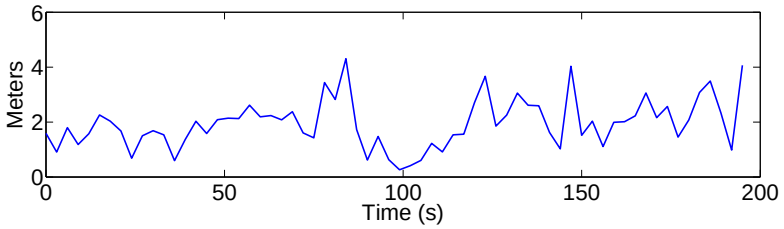


Figure 7-22: The error between the GPS and localization algorithm for the Moorea experiment.

The system requires this time to obtain enough range information to statically locate the static nodes and establish a system of coordinates.

The robot moved through the field of static nodes collecting range information received from the sensor nodes. Figure 7-19(a) shows a comparison of the GPS readings obtained at the surface and the acoustic tracking algorithm over a segment of the mission when the robot was at the surface. Figure 7-19(b) shows the same tracks with red lines drawn to indicate the correspondence between the points in the localization algorithm and the GPS readings. Figure 7-20 shows the difference between the GPS and tracking algorithm over time. The maximum error is about 3.15m with an average error of 1.67m. The average percent error based on a maximum range of 40m is 7.9%.

Figure 7-21 shows sample results of two weeks of experiments with localization and tracking performed in August 2008 at the UC Berkeley Gump Marine Research Station in Moorea, French Polynesia. This experiment was conducted with the robot at the surface, so it was able to obtain GPS positions as well as tracking data from the acoustic network. The first thirty minutes of the experiment were used for the sensor nodes to perform self-localization. The nodes for this experiment were deployed in a 20 by 30 meter area. The robot then moved through the field collecting ranges to the sensor nodes. Figure 7-21(a) shows the GPS and localization algorithm tracks and (b) adds the lines of correspondence between the two methods. Figure 7-22 shows the error between the localization algorithm and the GPS over the course of the experiment. The mean error in this experiment was 1.94m and the maximum was 4.31m, which is well within the GPS error. The average percent error based on a maximum range of 30m is 6.5%.

7.7.3 Running Algorithm Online

We also performed experiments running the grid-based algorithm in real-time on the system as it moved through a network of static nodes. In these experiments a boat was used to move an AquaNode running the localization algorithm through the water as Amour was not available. These experiments use the same code-base as the post-processed grid-based algorithm discussed in the previous section. These experiments verify that the algorithm performs as expected on the acoustic modem and that it can run in real-time.

In this section we discuss two different experiments performed in the Charles River. In the first, the node moved at the surface to obtain GPS ground-truth data. In the second experiment, GPS was unavailable. The Charles River has a depth of approximately 3m. We found that the acoustic modem performed relatively poorly in this environment. This is typical of acoustic modems, as the shallow environment introduces many reflections that interfere with the acoustic signal.

Figure 7-23 shows an experiment performed in the Charles River. In this experiment 5 static nodes were deployed for over one hour to collect static range information. The nodes were deployed in an area of approximately 20 by 30 meters. After this time

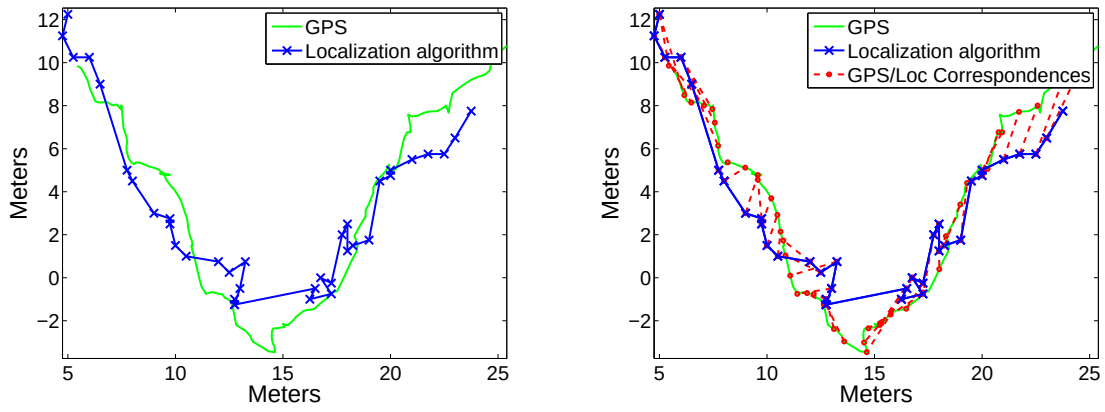


Figure 7-23: An experiment performed in the Charles River. (a) The GPS data compared to the path recovered by the acoustic localization algorithm and (b) their correspondences.

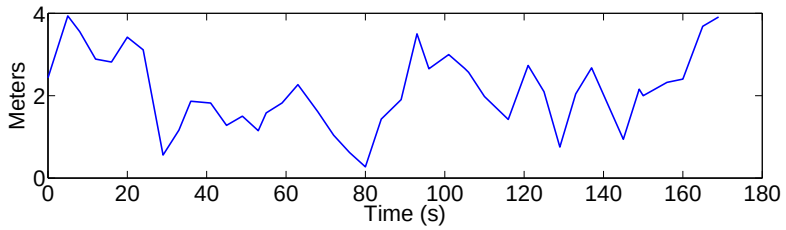


Figure 7-24: Error between the GPS and localization algorithm for the Charles River experiment.

only four of the nodes were able to statically localize themselves due to very poor acoustic communications in the river environment. We then moved a node through the water at the surface for approximately one hour. The node ran the localization algorithm online as it moved in the water. Each iteration of the algorithm took approximately a second to compute. The node also collected GPS to use for ground truth. Figure 7-23(a) shows three minutes of the track computed by the localization algorithm compared to the GPS path. Figure 7-23(b) shows the same data but adds lines to indicate the correspondence between the two tracks. Figure 7-24 plots the errors between the two over time. On average the error was 2.13m and at most was 3.94m. The average percent error based on a maximum range of 30m is 7.1%.

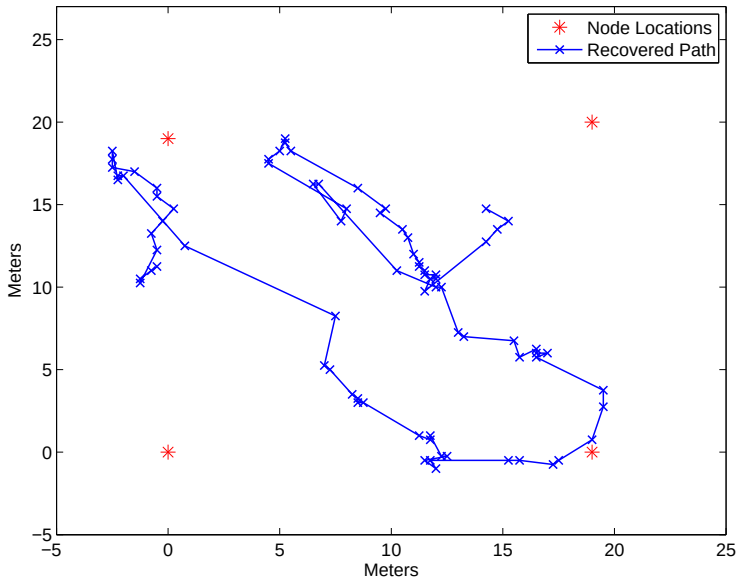


Figure 7-25: Part of the path computed by the acoustic localization algorithm in the Charles River.

Figure 7-25 shows the results of a different experiment performed in the Charles River. In this test four static nodes were deployed. Instead of running the static localization algorithm, which has poor results in the river environment, the nodes were placed and localized manually. The nodes were placed using a rope of known length to measure their locations. A node was then moved through the network, however, GPS was unavailable on the mobile node used for this experiment. The node computed its location online as it moved in the network.

Figure 7-25 shows seven minutes of data from the hour-long experiment. In most cases the locations were computed regularly along the path. However, there are some segments (e.g. around 5m,10m) where there were no locations computed for a rather large interval. In this area of the network the acoustic communication was extremely poor (despite being centrally located). Thus, the algorithm did not receive ranges for an extended period of time.

We have tested the localization algorithm in a variety of different water conditions. We found the most difficult environment to be the shallow river basin due to the poor acoustic communications in this environment. However, when we were able to obtain

regular ranges from the acoustic modem we have shown that we can compute the location of the robot to within a few meters of the GPS ground truth. This error is well within the error of the GPS receiver.

Chapter 8

Conclusions

Current underwater sensor systems lack the spacial and temporal resolution scientists need to improve their understanding of the underwater ecosystem. Data collection is often performed with sensors that are not networked and exhibit large lag times between the time the sensors collect the data and when scientists retrieve the data. Most sensors are fixed at a particular depth, often on the surface. To fully understand the dynamics of the ocean, scientists must know the details of the sub-sea processes at all depths over time.

In this thesis, we developed an underwater sensor network with the ability to dynamically adjust its depth. We use this capability to develop a decentralized algorithm that dynamically adjusts the depths of the nodes. By adjusting their depths the nodes improve their placement for collecting data for 3D sensory field reconstruction. We also developed an acoustic underwater localization algorithm, with minimal assumptions, that enables underwater robots to navigate within the network. This capability allows the underwater robot to act as a mobile sensor node in the network and lets the robot locate the sensor nodes to retrieve data from them.

For the sensor network system and both algorithms we performed extensive experiments to verify the systems. These experiments took place in a diverse set of locations: two different pools at MIT; the Charles River in Cambridge, MA; Lake Otsego in upstate New York; Roatán, Honduras in the Caribbean Basin; the UC

Berkeley Gump Marine research station in Moorea, French Polynesia; and the Singapore Harbor. Through many dozens of experiments, we developed the system and algorithms under realistic conditions.

8.1 Thesis Contributions

This thesis makes a number of theoretical contributions, including:

- A decentralized depth adjustment algorithm for improving the positioning of sensor nodes for sensing. The algorithm is provably convergent and stable;
- Underwater mobile robot localization algorithm with minimal assumptions and requirements;
- Analysis of energy trade-offs between the three communication methods (acoustic, radio, and optical).

This thesis also contributes to the field from a system development and experiment perspective:

- Development of a generic sensor network hardware platform that is used for AquaNodes and in a variety of other projects;
- Development of a flexible sensor network operating system;
- Implementation of a sensor network operating system that is easily customizable for a variety of projects;
- A novel underwater sensor network platform, called AquaNodes;
- A novel winch-based depth adjustment system for the AquaNodes;
- Simulations verifying the depth adjustment algorithm;
- Experiments and implementation of the decentralized depth adjustment algorithm on the AquaNode platform. These experiments verify the algorithm under real-world communication, motion, and sensing;
- Simulations verifying the localization algorithm;
- Experiments and implementation of the localization algorithm showing its performance on embedded hardware in-situ.

8.2 Lessons Learned

In the course of developing and testing the algorithms and system we learned a number of lessons. In this section we discuss some of these lessons and discuss how their implications.

Importance of Fault-Tolerant Algorithms

First, we learned the importance of developing fault-tolerant, distributed and decentralized algorithms. In real-world deployments nodes will fail. Algorithms that rely on data from all nodes or have a communication bottleneck are likely to fail.

The two algorithms we developed in this thesis are robust and handle the removal or addition of nodes at any time during experiments. The decentralized depth adjustment algorithm only relies on hearing messages, when they are available, from neighbors. The mobile robot localization algorithm uses as many or as few range measurements as it hears. Algorithms that do not have some sort of method for coping with these types of problems will not work, especially as network sizes increase to dozens and hundreds of nodes.

Carrick's Rule of 2

The necessity for fault-tolerant algorithms led to an additional lesson, which is more of an observation, and has taken on the name "Carrick's Rule of 2." This rule states that for any experiment, 2 of the systems deployed will fail for all deployments of less than 30 nodes. We identified this rule after observing numerous experiments, including those with systems unrelated to ours. Fortunately, this is a rule and not a law. In most cases, the latest version of the AquaNodes are reliable enough to deploy systems, and we expect 100% of the sensors to work as expected. One of the lessons learned from this rule is that field deployments have to be approached conservatively. If you need 10 sensors to cover an area, you should prepare at least 12 (if not more). The nodes themselves may be designed to be fully robust, but it is extremely difficult to predict the conditions of field experiments.

For instance, in an unrelated experiment we deployed 5 sensors for virtual fencing on cows [35]. These boxes were designed to handle large amounts of abuse (as

we witnessed the cows head butting each other with them in earlier deployments). However, 2 of the cows still managed to break the connection to the external battery during a critical experiment, despite the fact that all the nodes were working on all 5 animals for many days prior.

Developing New Software and Hardware Platforms

Another lesson we learned is that there are advantages and disadvantages to developing custom software and software systems. We spent a significant amount of time developing the hardware and software systems for the AquaNodes. On the hardware side we had little choice but to develop our own system, as existing systems did not meet our needs. No commercial underwater sensor network system exists to this day. The closest systems are acoustic modems that have some networking and sensing capabilities. However, these tend to be closed source and difficult to modify in order to test new algorithms and capabilities.

On the software side, the main alternative was to use TinyOS. Deciding to design our own operating system from the ground up was a difficult decision. We decided to do this for a number of reasons. First, we were developing a new hardware platform and as such would have had to write nearly all of the low-level drivers for TinyOS. This is one of the most time consuming aspects in developing an operating system for an embedded system.

Second, we designed our system to make it easy for computer scientists to program complex behaviors and algorithms. TinyOS, on the other hand, was designed to be easily programmable by environmental scientists. It is our opinion, however, that it did not achieve this goal. It still requires someone with a large depth of knowledge to program. Unfortunately, in designing TinyOS to be accessible to scientists, it obscures much of the functionality a computer scientists wants. As such, TinyOS can be difficult for both environmental and computer scientists.

The drawbacks of not using TinyOS are twofold. First, many of the libraries available on TinyOS need to be ported for use on our system. On our system, for instance, we have an interface to support a variety of multi-hop routing algorithms. As we did not explicitly need this capability, we did not port the routing algorithms

to our system. Had we used TinyOS, we could have plugged them in more easily, if only for demonstration purposes. Thus, in designing our operating system we lost some of the available libraries, although most are fairly easy to port to our system. The second drawback of developing our own hardware and operating system is that it is more difficult to compare to the gold standard in the sensor network community of a Mote with TinyOS.

If we were to start over and reconsider using TinyOS we still would choose to develop our own system. This is not a decision that should be taken lightly. The development of a hardware and operating system platform took a large amount of time. In the end we are better off for having complete control and understanding of our system, but it was a steep learning/development curve.

Use of Platform in Many Projects

Another lesson learned is that using a system across multiple, diverse projects has both advantages and disadvantages. The core hardware and software of the AquaNode is used in a number of different projects. One advantage is that different people tested both the hardware and software under different sets of assumptions. This enabled broader testing of the system and the discovery of unintended consequences of design decisions. In addition to improving reliability, using the system across multiple projects sped development as more people were working on the platform and less code was duplicated as it could be shared. In our system over 80% of the base software system for all projects are shared and used amongst the projects.

Another advantage of using a common platform is the ability to quickly connect diverse systems. Since the systems use the same communication and processing structure, it would be easy to connect, for instance, the river flooding sensor network with the underwater sensor network. This combined system could monitor the effect of river floods on the water quality of coastal waters.

However, to satisfy the needs of all the projects, some software and hardware compromises had to be made. For instance, the base board in the AquaNode has a charger for a single cell Li-Ion battery as this is the type of battery used by most of the other projects. But, the acoustic modem requires a 3 cell battery. As such, we added

extra power management and charging circuitry to the midlayer board to address the needs of the acoustic modem and we did not use the single cell power systems. The cost of the system could have been reduced if the hardware was designed only for the AquaNode. The benefit of shared code and hardware testing and development, however, tends to outweigh the added cost.

8.3 Future Work

There are a number of directions to take the research started in this thesis in the future. The decentralized depth adjustment algorithm discussed in Chapters 4 and 5 changes the depths of the nodes to optimize placement for sensing. This algorithm can be extended to include the case where some or all nodes have more degrees of freedom. For instance, extending the algorithm to include mobile robots could greatly enhance the sensor coverage by allowing the robots to fill in gaps that cannot be covered by the nodes that can only change depth.

From an engineering perspective, the core AquaNode hardware is a mature and stable platform; the depth adjustment system, however, is a prototype. It worked well in our experiments, but further work has to be done to make the system robust for long-term deployments. A tensioning system should be added to prevent winding issues that arise from waves and currents. In addition, experiments need to be performed to determine the maximum deployment time before biofouling starts to effect the operation of the depth adjustment system. Additional screens and filters may be necessary to reduce the amount of bio-growth on the system. These changes and experiments will enable long-term deployments of the AquaNodes with the depth adjustment system.

Another interesting direction for future work, upon which we only touched on briefly, is the ability to send messages in a power-efficient manner using the multi-modal communication capabilities of the AquaNodes. In particular, depending on the type and amount of data that needs to be sent, the nodes should intelligently select to: send acoustically, surface with the depth adjustment system to send over

the radio, or call the robot and download the data optically. Looking at maximizing communication while minimizing energy usage within this network with 3 diverse communication systems is an interesting problem. For a pair of nodes it is fairly straight forward and Section 3.4.4 discusses this. However, for a network with many nodes and many messages, this is an open problem with many potential solutions.

Finally, the acoustic communication channel is an extremely challenging medium. We presented preliminary results showing that changing the depths of the sensor network nodes in the water can improve acoustic communication. In the future we plan to continue this avenue of investigation to develop decentralized algorithms to improve network communication by adjusting the depths of the nodes in the network.

The goal of this thesis has been to develop a platform and supporting algorithms to enhance data collection underwater. Furthermore, we hope that the systems and algorithms presented in this thesis will motivate future research in underwater robotics and sensor networks. We still know relatively little about the underwater environment, and as one of our most precious natural resources, it is critical that we develop and improve the systems that will allow us to better understand the world's water resources.

Bibliography

[1] Gump south pacific marine research station. http://moorea.berkeley.edu/.

[2] Laird technologies. http://www.lairdtech.com/wireless/.

[3] Moon science and strategy plan. Technical report.

[4] NOAA national data buoy center. http://www.ndbc.noaa.gov/.

[5] Undersea systems, equipment, and sonar. http://www.benthos.com/.

[6] R. Adler, M. Flanigan, J. Huang, R. Kling, N. Kushalnagar, L. Nachman, C. Wan, and M. Yarvis. Intel mote 2: an advanced platform for demanding sensor network applications. In *Proceedings of the 3rd international conference on Embedded networked sensor systems*, pages 298–298, San Diego, California, USA, 2005. ACM.

[7] K. Akkaya and A. Newell. Self-deployment of sensors for maximized coverage in underwater acoustic sensor networks. *Computer Communications*, 32(7-10):1233–1244, May 2009.

[8] I.F. Akyildiz, D. Pompili, and T. Melodia. Challenges for efficient communication in underwater acoustic sensor networks. *SIGBED Rev.*, 1(2):3–8, 2004.

[9] I.F. Akyildiz, D. Pompili, and T. Melodia. State-of-the-art in protocol research for underwater acoustic sensor networks. In *Proceedings of the 1st ACM international workshop on Underwater networks*, pages 7–16, Los Angeles, CA, USA, 2006. ACM.

[10] S. M. Nazrul Alam and Z.J. Haas. Coverage and connectivity in three-dimensional networks. In *Proceedings of the 12th annual international conference on Mobile computing and networking*, pages 346–357, Los Angeles, CA, USA, 2006. ACM.

[11] K. Alverson. Filling the gaps in GOOS. *Journa of Ocean Technology*, 3(3), 2008.

[12] M. Arrott, A. Chave, I. Krueger, J. Orcutt, A. Talalayevsky, and F. Vernon. The approach to cyberinfrastructure for the ocean observatories initiative. In *Oceans 2007*, pages 1–6, 2007.

[13] A. Bahr and J. Leonard. Cooperative localization for autonomous underwater vehicles. In *Experimental Robotics*, pages 387–395. 2008.

[14] E. Basha, S. Ravela, and D. Rus. Model-based monitoring for early warning flood detection. In *Proceedings of the 6th ACM conference on Embedded network sensor systems*, pages 295–308, Raleigh, NC, USA, 2008. ACM.

[15] J. Bicket, D. Aguayo, S. Biswas, and R. Morris. Architecture and evaluation of an unplanned 802.11b mesh network. In *Proceedings of the 11th annual international conference on Mobile computing and networking*, pages 31–42, Cologne, Germany, 2005. ACM.

[16] W. P. Bissett, O. Schofield, S. Glenn, J. J. Cullen, W. Miller, A. Pluddeman, and C. Mobley. Resolving the impacts and feedbacks of ocean optics on upper ocean ecology. *Oceanography*, 14(4):30–53, 2001.

[17] N. Blough and R. Del Vecchio. Chromophoric dom in the coastal environment. In Dennis A. Hansell and Craig A. Carlson, editors, *Biogeochemistry of Marine Dissolved Organic Matter*, pages 509 – 546. Academic Press, San Diego, 2002.

[18] A. Blumberg, R.P. Signell, and H.L. Jenter. Modelling transport processes in the coastal ocean. *J. Marine Env. Engg.*, 1:31–52, 1993.

[19] A. F. Blumberg and G. L. Mellor. A description of a three-dimensional coastal ocean circulation model. *Three-Dimensional Coastal Ocean Models, Coastal and Estuarine Sciences*, (4):1–16, 1987.

[20] V. Bokser, C. Oberg, G. Sukhatme, and A. Requicha. A small submarine robot for experiments in underwater sensor networks. In *International Federation of Automatic Control Symposium on Intelligent Autonomous Vehicles*, 2004.

[21] F. Bullo, J. Cortés, and S. Mortínez. *Distributed Control of Robotic Networks.* Applied Mathematics Series. Princeton University Press, 2009.

[22] K. L. Carder, R. G. Steward, G. R. Harvey, and P. B. Ortner. Marine humic and fulvic acids: Their effects on remote sensing of ocean chlorophyll. *Limnology and Oceanography*, 34(1):68–81, 1989.

[23] M. Castelnovi, A. Sgorbissa, and R. Zaccaria. Markov-localization through color features comparison. In *Proceedings of the 2004 IEEE International Symposium on Intelligent Control*, pages 437– 442, 2004.

[24] E. Cayirci, H. Tezcan, Y. Dogan, and V. Coskun. Wireless sensor networks for underwater survelliance systems. *Ad Hoc Networks*, 4(4):431–446, July 2006.

[25] M. Chaffey, L. Bird, J. Erickson, J. Graybeal, A. Hamilton, K. Headley, M. Kelley, L. McBride, E. Mellinger, T. Meese, T. O'Reilly, W. Paul, M. Risi, and W. Radochonski. MBARI's buoy based seafloor observatory design. In *OCEANS 2004*, volume 4, pages 1975–1984 Vol.4, 2004.

[26] V. Chandrasekhar, W. Seah, Y. Sang Choo, and H. Voon Ee. Localization in underwater sensor networks: survey and challenges. In *Proceedings of the 1st ACM international workshop on Underwater networks*, pages 33–40, Los Angeles, CA, USA, 2006. ACM.

[27] M. Channey. Short range underwater opical communication links. Master's thesis, North Carolina State University, 2005.

[28] B. Cochenour, L. Mullen, A. Laux, and T. Curran. Effects of multiple scattering on the implementation of an underwater wireless optical communications link. In *OCEANS*, pages 1–6, 2006.

[29] J. Cortes, S. Martinez, T. Karatas, and F. Bullo. Coverage control for mobile sensing networks. In *Robotics and Automation, 2002. Proceedings. ICRA '02. IEEE International Conference on*, volume 2, pages 1327–1332, 2002.

[30] T. B Curtin, J. G Bellingham, J. Catipovic, and D. Webb. Autonomous oceanographic sampling networks. *Oceanography*, 6(3):86–94, 1993.

[31] C. Detweiler. Passive mobile robot localiation within a fixed beacon field. Master's thesis, MIT, 2006.

[32] C. Detweiler, S. Sosnowski, I. Vasilescu, and D. Rus. Saving energy with buoyancy and balance control for underwater robots with dynamic payloads. In *Experimental Robotics*, pages 429–438. 2009.

[33] J. Djugash, S. Singh, and P. Corke. Further results with localization and mapping using range from radio. In *International Conference on Field and Service Robotics*, July 2005.

[34] D. Dobkin, J. Hershberger, D. Kirkpatrick, and S. Suri. Computing the intersection-depth of polyhedra. *Algorithmica*, 9(6):518–533, 1993.

[35] M. Doniec, C. Detweiler, I. Vasilescu, D. Anderson, and D. Rus. Autonomous gathering of livestock using a multi-functional sensor network platform. In *To appear in Proceedings of 2010 ACM Workshop on Hot Topics in Embedded Networked Sensors (HotEMNETS 2010)*, Killarney, Ireland, June 2010.

[36] M. Doniec, I. Vasilescu, M. Chitre, C. Detweiler, M. Hoffmann-Kuhnt, and D. Rus. AquaOptical: a lightweight device for high-rate long-range underwater point-to-point communication. In *OCEANS 2009*, pages 1 –6, October 2009.

[37] P. Dutta and D. Culler. Epic: An open mote platform for Application-Driven design. In *Information Processing in Sensor Networks, 2008. IPSN '08. International Conference on*, pages 547–548, 2008.

[38] J. Cumming et al. Ocean data assimilation systems for godae. *Oceanography*, 22(3):96–109, 2009.

[39] N. Farr, A.D. Chave, L. Freitag, J. Preisig, S.N. White, D. Yoerger, and F. Sonnichsen. Optical modem technology for seafloor observatories. In *OCEANS*, pages 1–6, 2006.

[40] Y. Feng and K. Yu. Nonparametric estimation of time-varying covariance matrix in a slowly changing vector random walk model. http://mpra.ub.uni-muenchen.de/1597/, 2006.

[41] L. Freitag, M. Grund, S. Singh, J. Partan, P. Koski, and K. Ball. The WHOI micro-modem: an acoustic communications and navigation system for multiple platforms. In *OCEANS, 2005. Proceedings of MTS/IEEE*, pages 1086–1092 Vol. 2, 2005.

[42] D. Frye, J. Ware, M. Grund, J. Partan, P. Koski, S. Sin H, L. Freitag, J. Collins, and R. Detrick. An Acoustically-Linked Deep-Ocean observatory. 2005.

[43] A. Galstyan, B. Krishnamachari, K. Lerman, and S. Pattem. Distributed online localization in sensor networks using a moving target. In *IPSN'04: Proceedings of the third international symposium on Information processing in sensor networks*, pages 61–70, New York, NY, USA, 2004. ACM Press.

[44] G. B. Gardner, R. F. Chen, and A. Berry. High-resolution measurements of chromophoric dissolved organic matter (cdom) in the neponset river estuary, boston harbor, ma. *Marine Chemistry*, 96(1-2):137 – 154, 2005.

[45] J.W. Giles and I.N. Bankman. Underwater optical communications systems. part 2: basic design considerations. *Military Communications Conference*, pages 1700–1705 Vol. 3, Oct. 2005.

[46] S.M. Glenn, O.M. Schofield, R. Chant, J. Kohut, J. McDonnell, and S.D. McLean. The leo-15 costal cabled observatory – phase II for the next evolutionar decade of oceanography. In *Proceedings of Scientific Submarine Cable 2006*, Dublin, Ireland, February 2006.

[47] C. Guestrin, A. Krause, and A. Singh. Near-optimal sensor placements in gaussian processes. In *Proceedings of the 22nd international conference on Machine learning*, pages 265–272, Bonn, Germany, 2005. ACM.

[48] J.-S. Gutmann. Markov-kalman localization for mobile robots. volume 2, pages 601–604, Aug 2002.

[49] M. J Hahn. *Undersea Navigation via a Distributed Acoustic Communications Network*. PhD thesis, Naval Postgraduate School Monterey CA, 2005.

[50] F. Hanson and S. Radic. High bandwidth underwater optical communication. *Appl. Opt.*, 47(2):277–283, 2008.

[51] J. Hill and D. Culler. *A wireless embedded sensor architecture for system-level optimization*. 2001.

[52] M.J. Howarth, R. Proctor, P.J. Knight, M.J. Smithson, and D.K. Mills. The liverpool bay coastal observatory: towards the goals. In *OCEANS 2006*, pages 1–6, 2006.

[53] B.M. Howe and T. McGinnis. Sensor networks for cabled ocean observatories. In *Underwater Technology, 2004. UT '04. 2004 International Symposium on*, pages 113–120, 2004.

[54] B. Hull, V. Bychkovsky, Y. Zhang, K. Chen, M. Goraczko, A. Miu, E. Shih, H. Balakrishnan, and S. Madden. CarTel: a distributed mobile sensor computing system. In *Proceedings of the 4th international conference on Embedded networked sensor systems*, pages 125–138, Colorado, USA, 2006. ACM.

[55] A.R. Isern. The ocean observatories initiative: Wiring the ocean for interactive scientific discovery. In *OCEANS 2006*, pages 1–7, 2006.

[56] H.W. Jannasch, L.J. Coletti, K.S. Johnson, S.E. Fitzwater, J.A. Needoba, and J.N. Plant. The land/ocean biogeochemical observatory: A robust networked mooring system for continuously monitoring complex biogeochemical cycles in estuaries. *Limnology and Oceanography: Methods*, 6:263—276, 2008.

[57] M. Jiang, M. Zhou, S. Libby, and C. D. Hunt. Influences of the gulf of maine intrusion on the massachusetts bay spring bloom: A comparison between 1998 and 2000. *Continental Shelf Research*, 27(19):2465 – 2485, 2007.

[58] L.S. Joeris. A horizontal sampler for collection of water samples near the bottom. *Limnology and Oceanography*, 9(4):595–598, October 1964. ArticleType: primary_article / Full publication date: Oct., 1964 / Copyright <A9> 1964 American Society of Limnology and Oceanography.

[59] K. S. Johnson. Observing biogeochemical cycles at global scales with profiling floats and gliders: Prospects for a global array. *Oceanography*, 22(3):216–226, 2009.

[60] A. Kansal, M. Goraczko, and F. Zhao. Building a sensor network of mobile phones. In *Proceedings of the 6th international conference on Information processing in sensor networks*, pages 547–548, Cambridge, Massachusetts, USA, 2007. ACM.

[61] G. Kantor and S. Singh. Preliminary results in range-only localization and mapping. In *Proceedings of the IEEE Conference on Robotics and Automation (ICRA '02)*, volume 2, pages 1818–1823, May 2002.

[62] C.W. Ko, J. Lee, and M. Queyranne. An exact algorithm for maximum entropy sampling. *Operations Research*, 43(4):684–691, August 1995. ArticleType: primary_article / Full publication date: Jul. - Aug., 1995 / Copyright l' 1995 INFORMS.

[63] N. Kottege and U. Zimmer. Mls-based, distributed bearing, range, and posture estimation for schools of submersibles. In *International Symposium on Experimental Robotics*, 2006.

[64] A. Krause, R. Rajagopal, A. Gupta, and C. Guestrin. Simultaneous placement and scheduling of sensors. In *Proceedings of the 2009 International Conference on Information Processing in Sensor Networks - Volume 00*, pages 181–192. IEEE Computer Society, 2009.

[65] D. Kurth. Range-only robot localization and SLAM with radio. Master's thesis, Robotics Institute Carnegie Mellon University, Pittsburgh PA, May 2004.

[66] J. LaSalle. Some extensions of lyapunov's second method. *IRE Transactions on Circuit Theory*, 7(4):520–527, 1960.

[67] N.E. Leonard, D.A. Paley, F. Lekien, R. Sepulchre, D.M. Fratantoni, and R.E. Davis. Collective motion, sensor networks, and ocean sampling. *Proceedings of the IEEE*, 95(1):48–74, 2007.

[68] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler. *TinyOS: An Operating System for Sensor Networks*, pages 115–148. 2005.

[69] D. Lynch and D. McGillicuddy. Objective analysis for coastal regimes. *Continental Shelf Research*, 21(11-12):1299–1315, July 2001.

[70] Meraki. Wireless routers & WiFi networks: Indoor and outdoor wireless networks by meraki. http://meraki.com/, 2009.

[71] D. Moore, J. Leonard, D. Rus, and S. Teller. Robust distributed network localization with noisy range measurements. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 50–61, Baltimore, MD, USA, 2004. ACM.

[72] L. Nachman, J. Huang, J. Shahabdeen, R. Adler, and R. Kling. IMOTE2: serious computation at the edge. In *Wireless Communications and Mobile Computing Conference, 2008. IWCMC '08. International*, pages 1118–1123, 2008.

[73] L. Nachman, R. Kling, R. Adler, J. Huang, and V. Hummel. The intel<ae> mote platform: a bluetooth-based sensor network for industrial monitoring. In *Information Processing in Sensor Networks, 2005. IPSN 2005. Fourth International Symposium on*, pages 437–442, 2005.

[74] P. Newman and J. Leonard. Pure range-only sub-sea slam. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 2, pages 1921–1926, Taiwan, 2003.

[75] E. Olson, J. Leonard, and S. Teller. Robust range-only beacon localization. In *Proceedings of Autonomous Underwater Vehicles*, pages 66–75, 2004.

[76] J. Park and I. W. Sandberg. Universal approximation using radial-basis-function networks. *Neural Comput.*, 3(2):246–257, 1991.

[77] J. Partan, J. Kurose, and B.N. Levine. A survey of practical issues in underwater networks. *SIGMOBILE Mob. Comput. Commun. Rev.*, 11(4):23–33, 2007.

[78] N. M. Patrikalakis, S. L. Abrams, J. G. Bellingham, W. Cho, K. P. Mihanetzis, A. R. Robinson, H. Schmidt, and P. C. H. Wariyapola. The digital ocean. In *Proc. of Computer Graphics International, GCI*, pages 45–53, 2000.

[79] J. Polastre, R. Szewczyk, and D. Culler. Telos: enabling ultra-low power wireless research. In *Information Processing in Sensor Networks, 2005. IPSN 2005. Fourth International Symposium on*, pages 364–369, 2005.

[80] D. Pompili, T. Melodia, and I.F. Akyildiz. Deployment analysis in underwater acoustic wireless sensor networks. In *Proceedings of the 1st ACM international workshop on Underwater networks*, pages 48–55, Los Angeles, CA, USA, 2006. ACM.

[81] D. Pompili, T. Melodia, and I.F. Akyildiz. Routing algorithms for delay-insensitive and delay-sensitive applications in underwater sensor networks. In *Proceedings of the 12th annual international conference on Mobile computing and networking*, pages 298–309, Los Angeles, CA, USA, 2006. ACM.

[82] J. Preisig. Acoustic propagation considerations for underwater acoustic communications network development. *SIGMOBILE Mob. Comput. Comm. Rev.*, 11(4):2–10, 2007.

[83] N. Priyantha, A. Chakraborty, and H. Balakrishnan. The cricket location-support system. *null*, pages 32—43, 2000.

[84] P. Rigby. *Autonomous Spatial Analysis using Gaussian Process Models*. PhD thesis, University of Sydney, 2008.

[85] F. Schill, U.R. Zimmer, and J. Trumpf. Visible spectrum optical communication and distance sensing for underwater applications. In *AGRA*, 2004.

[86] M. Schwager. *A Gradient Optimization Approach to Adaptive Multi-Robot Control*. PhD thesis, MIT, 2009.

[87] M. Schwager, C. Detweiler, I. Vasilescu, D. M. Anderson, and D. Rus. Data-Driven identication of group dynamics for motion prediction and control. *Journal of Field Robotics*, 25(6-7):305–324, 2008.

[88] M. Schwager, B. Julian, and D. Rus. Optimal coverage for multiple hovering robots with Downward-Facing cameras. In *Proc. of International Conference on Robotics and Automation (ICRA09)*, Kobe, Japan, May 2009.

[89] M. Schwager, D. Rus, and J.-J. Slotine. Decentralized, adaptive coverage control for networked robots. *The International Journal of Robotics Research*, 28(3):357–375, March 2009.

[90] M. Siderius, M. Porter, P. Hursky, and V. McDonald. Effects of ocean thermocline variability on noncoherent underwater acoustic communications. *J. of the Acoustical Society of America*, 121(4):1895–1908, April 2007.

[91] R.P. Signell, H.L. Jenter, and A.F. Blumberg. Predicting the physical effects of relocating boston's sewage outfall. *Estuarine, Coastal and Shelf Science*, 50(1):59 – 71, 2000.

[92] A. Singh, A. Krause, C. Guestrin, and W.J. Kaiser. Efficient informative sensing using multiple robots. *J. Artif. Int. Res.*, 34(1):707–755, 2009.

[93] J.H. Smart. Underwater optical communications systems part 1: variability of water optical parameters. In *Military Communications Conference*, pages 1140–1146 Vol. 2, 2005.

[94] F. Spiess. Apparatus and method for supporting oceanographic equipment at selected ocean depths - patent 4215572. http://www.freepatentsonline.com/4215572.html, 1980.

[95] J. Springer, J. Burkholder, P. Glibert, and R. Reed. Use of a real-time remote monitoring network (rtrm) and shipborne sampling to characterize a dinoflagellate bloom in the neuse estuary, north carolina, usa. *Harmful Algae*, 4(3):533–551, March 2005.

[96] M. Stojanovic. Underwater acoustic communications. In *ELECTRO-IEEE CONFERENCE-*, pages 435–435, 1995.

[97] M. Stojanovic, J. G. Proakis, and J. A. Catipovic. Performance of high-rate adaptive equalization on a shallow water acoustic channel. *Journal of the Acoustical Society of America*, 100(4):2213–2219, 1996.

[98] Y. Tsuchida, N. Hama, and M. Takahata. An optical telemetry system for underwater recording of electromyogram and neuronal activity from non-tethered crayfish. *J of Neuroscience Methods*, 2004.

[99] J. Vaganay, J. G. Bellingham, and J. J. Leonard. Comparison of fix computation and filtering for autonomous acoustic navigation, 1998.

[100] J. Vaganay, J.J. Leonard, J.A. Curcio, and J.S. Willcox. Experimental validation of the moving long base-line navigation concept. In *Autonomous Underwater Vehicles*, pages 59–65, 2004.

[101] I. Vasilescu. *Using Light Underwater: Devices, Algorithms and Systems for Maritime Persistent Surveillance*. PhD thesis, MIT, 2009.

[102] I. Vasilescu, C. Detweiler, M. Doniec, D. Gurdan, S. Sosnowski, J. Stumpf, and D. Rus. AMOUR v: A hovering energy efficient underwater robot capable of dynamic payloads. *The International Journal of Robotics Research*, 29(5):547–570, April 2010.

[103] I. Vasilescu, C. Detweiler, and D. Rus. Aquanodes: an underwater sensor network. In *Proceedings of the second workshop on Underwater networks*, pages 85–88, Montreal, Quebec, Canada, 2007. ACM.

[104] I. Vasilescu, C. Detweiler, and D. Rus. Color-Accurate underwater imaging using perceptual adaptive illumination. Zaragoza, Spain, 2010.

[105] T. Wark, P. Corke, P. Sikka, L. Klingbeil, Ying Guo, C. Crossman, P. Valencia, D. Swain, and G. Bishop-Hurley. Transforming agriculture through pervasive wireless sensor networks. *Pervasive Computing, IEEE*, 6(2):50–57, 2007.

[106] G. Welch and G. Bishop. An introduction to the kalman filter. Technical report, Chapel Hill NC USA, 1995.

[107] B. Woodward and R. S. H. Istepanian. The use of underwater acoustic biotelemetry for monitoring the ECGof a swimming patient. In *IEEE Engineering in Medicine and Biology Society*, page 4, 1995.

[108] N.K. Yilmaz, C. Evangelinos, P. Lermusiaux, and N.M. Patrikalakis. Path planning of autonomous underwater vehicles for adaptive sampling using mixed integer linear programming. *Oceanic Engineering, IEEE Journal of*, 33(4):522–537, 2008.