Adaptive Decentralized Control of Underwater Sensor Networks for Modeling Underwater Phenomena

Carrick Detweiler[†1], Marek Doniec[†], Mingshun Jiang[⋆], Mac Schwager[†],
Robert Chen[⋆], Daniela Rus[†]
[†]CSAIL MIT [⋆]UMass Boston
{carrick, doniec, schwager, rus}@csail.mit.edu {mingshun.jiang, bob.chen}@umb.edu

Abstract

Understanding the dynamics of bodies of water and their impact on the global environment requires sensing information over the full volume of water. We develop a gradient-based decentralized controller that dynamically adjusts the depth of a network of underwater sensors to optimize sensing for computing maximally detailed volumetric models. We prove that the controller converges to a local minimum. We implement the controller on an underwater sensor network capable of adjusting their depths. Through simulations and experiments, we verify the functionality and performance of the system and algorithm.

Categories and Subject Descriptors

C.3 [**Special-Purpose and Application-Based Systems**]: Microprocessor/microcomputer application; C.4 [**Performance of Systems**]: Measurement techniques; J.2 [**Physical Sciences and Engineering**]: Earth and atmospheric sciences

General Terms

Algorithms, Design, Experimentation, Measurement

Keywords

Depth Adjustment, Ocean, Sensing, Sensor Network

1 Introduction

Over 70% of the world is covered in water. Sensing just the surface of the rivers, lakes, and oceans provides a huge challenge to scientists. To more fully understand the dynamics of these bodies of water and their impact on the global environment, we must understand them at all depths, not just on the surface. Collecting sufficient data to understand the full water environment requires either high density placement of sensors or autonomous motion of sensors, both of which are incredibly challenging in underwater environments.

[1]Now at the University of Nebraska–Lincoln.

SenSys'10, November 3–5, 2010, Zurich, Switzerland.

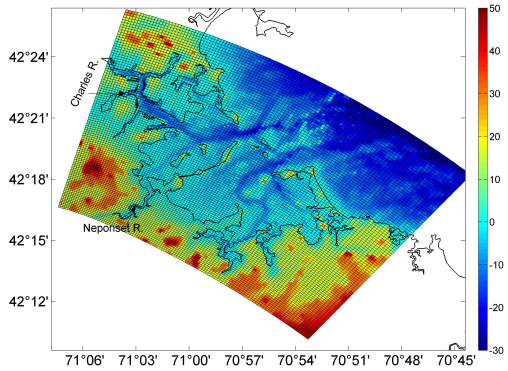


Figure 1. Boston Harbor bathymetry and grid for hydrodynamic model.

Current systems for studying the dynamics of water bodies at multiple depths fall into three categories: (1) static sensor buoys; (2) ships/ROVs/AUVs; and (3) water column profilers. The first two cannot be deployed in a sufficient density to cover the full water volume due to cost; the latter lacks the autonomy and communications to cover unknown and dynamic bodies of water. Solving this requires algorithms and systems that enable adaptive and decentralized sensing.

In this paper, to solve this, we develop, analyze, and test a decentralized, adaptive algorithm for positioning an underwater sensor network. Our sensor network nodes dynamically adjust their depths through a new decentralized, gradient-descent based algorithm with guaranteed properties. The dynamic depth adjustment algorithm runs online, which enables the nodes to adapt to changing conditions (e.g. tidal) and does not require a-priori decisions about node placement in the water. Through neighbor communication the algorithm collaboratively optimizes the nodes' depths for sensing in support of computing maximally detailed volumetric models. We prove the controller algorithm converges to a local minimum. Through simulations and experiments on our AQUANODE hardware platform, we show the local minimum is near the global minimum of the system; often the local minimum equals the global.

We apply the algorithm to the problem of monitoring chromophoric dissolved organic matter (CDOM) in the Neponset River that feeds into Boston harbor (see Figure 1). CDOM is part of the dissolved organic matter in rivers,

lakes, and oceans. An understanding of CDOM dynamics in coastal waters and of its resulting distribution is important for remote sensing and for estimating light penetration in the ocean. In this paper 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. We also use the CDOM model as input for a four-node test deployment in the Charles River.

The decentralized controller positions the nodes so 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. 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 [25]. We also compute a numeric covariance for the Neponset River based on data from a numerical model of the river. The model is based on readings from a few specific sensor locations and is extended to all points in the river using a physics-based hydrodynamic model [21].

The algorithm assumes a fixed covariance model, however, we show in a river test that the algorithm can be iterated with different covariance models to capture dynamic phenomena. In the Neponset River, the concentration of CDOM is highly dependent on the tide level, which causes river level variations of about 2m. The physics-based model lets us numerically compute the covariance of the CDOM readings in the Neponset River based on different tide levels. The nodes adjust their covariance model, and therefore their depths, based on the tide charts. The model data is accurate on average, however, it may not accurately capture small-scale temporal and spatial variations. The sensor nodes and algorithm fill this gap and provide detailed measurements at informative locations. Over time, the physics-based CDOM model can be enhanced by using the new measurements.

The controller uses the covariance in a decentralized gradient descent algorithm. We implement this solution in simulation and on our AQUANODE underwater sensor network that has depth adjustment capabilities and test it in the lab, pool, and river. The algorithm requires very little communication, allowing each node to only send its own depth information, as well as providing fault tolerance in instances where packets are lost. Both are important in underwater sensor networks that 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 sensor network.

We introduce our underwater sensor network in Section 2. We then discuss the model of the river and ocean environment, and its importance for scientific understanding in Section 3. We next introduce and analyze our decentralized depth controller algorithm in Section 4. Section 5 explores results of simulations, pool, and river experiments. We then analyze and discuss issues related to parameters, placement, and implementation in Section 6. This is followed by a discussion of related work in Section 7. Finally, we discuss future work and conclude in Section 8.

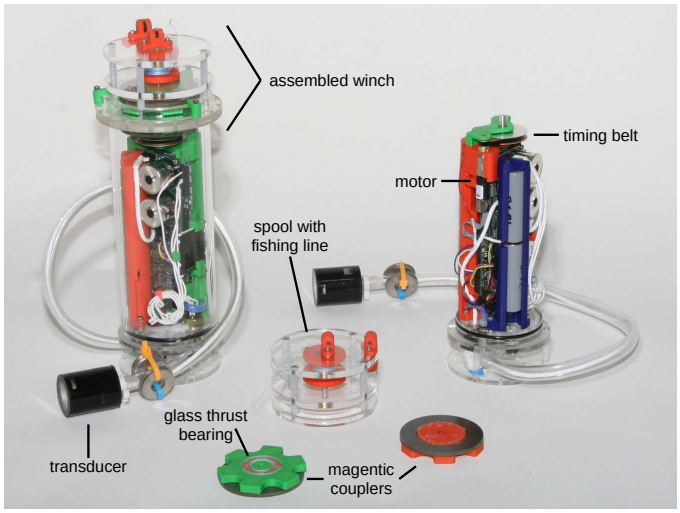


Figure 2. Depth adjustment system and AQUANODE.

2 Underwater Sensor Network Platform

We have developed an inexpensive underwater sensor network system that incorporates the ability to dynamically adjust its depth. The base sensor node hardware is called the AQUANODE platform and is described in detail in [13, 15, 35]. We have extended this basic underwater sensor network with autonomous depth adjustment ability and created a five node sensor network system, whose nodes move up and down in the water column under their own control. Each node costs under two thousand dollars. Here we will briefly summarize the system and describe some details of the winch-based depth adjustment system.

Figure 2 shows a picture of two AQUANODES with the depth adjustment hardware. The AQUANODE operating system runs on an NXP LPC2148 ARM7TDMI processor clocked at 60MHz. This processor has 40kB of ram and 512kB of on-chip flash. A SD card slot allows logging of gigabytes of data onboard. Each node has a pressure and temperature sensor as well as inputs for both analog and digital sensors connected via an underwater connect. Examples of sensors we have connected include CDOM, salinity, dissolved oxygen, and cameras.

The main communication system we use is a custom developed 10W acoustic modem [15]. The modem uses a frequency-shift keying (FSK) modulation with a 30KHz carrier frequency and has a physical layer baud rate of 300b/s. The MAC layer is a self-synchronizing TDMA protocol with 4 second slots. In each slot the master can send a packet and receive a response from one other sensor node. Each packet contains 11 bytes of payload. The acoustic modems are also able to measure distances between pairs of nodes. In previous work we have demonstrated how this can be used to localize static and mobile nodes in the underwater network [14]. We can use this capability to determine the positions of the nodes in our experiments.

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 about two days. The depth adjustment system typically uses under 1W allowing for continuous depth adjustment for over two days. By varying the duty cycle of the sensing, communication, and motion the desired deployment time can be achieved.

The AQUANODE is anchored at the bottom and floats mid-water column. 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 winch mechanism and 30.5cm with the winch attached. It weighs 1.8kg and is 200g buoyant with the depth adjustment system attached. The depth adjustment system allows the AQUANODES to adjust their depth at a speed of up to 0.5m/s and use approximately 1W when in motion.

The winch is driven by a 1.5A motor controller with a software quadrature decoder. This is connected to a 1.95 watt Faulhaber motor, which 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 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 and it holds over 50 meters of line.

3 Modeling Underwater Phenomena

One prominent phenomenon in coastal oceans is the existence of tidal fronts, which are located at the meeting place of freshwater and oceanic water, normally determined by the strongest salinity gradient. Such fronts are typically associated with high concentrations of suspended sediments and hence with high turbidity, so-called the maximum turbidity zone (MTZ) [30]. The frontal zones also tend to be biologically active areas attracting zooplankton and fish larvae due to turbulent mixing and physical retention [16, 29]. For a small river, the flow may be approximated as two-dimensional (one vertical and one along the river channel), and hence the front can be approximated as a thin line from the surface tilting down to the bottom with a vertical thickness about 10s of cm. The frontal position is determined by the river flow and tidal movement, but is moving back and forth along with tidal flow. In Neponset River, the dominant period of frontal movement is 12.4 hrs., controlled by the M2 tide component.

Underwater sensor networks enable detecting and measuring the tidal front in Neponset River in conjunction with a numerical model. The model we use in this paper was developed for Boston Harbor (BH), based on the Estuarine, Coastal, Ocean Model (ECOM-si) with Mellor and Yamada 2.5 turbulent closure for the vertical mixing [8, 7, 33]. The model domain covers the entire BH (over 500km^2) and a part of the Massachusetts Bay (MB) with a grid resolution around 70m (Figure 1). The model was forced by surface winds and heat fluxes derived from measurements at NOAA buoy 44013 in western MB, and freshwater discharges at the USGS gauges, and boundary forcing (tides, currents, temperature and salinity) derived from the model output of the MB hydrodynamic model [21]. The model captures the general dynamic processes including tidal cycle, seasonal development of stratification, and wind- and river-driven circulation.

We are particularly interested in CDOM, which 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 [17]. Additional sources (sinks) from mid-estuary production (removal) will make the mixing curve concave upward. An understanding of CDOM dynamics in coastal waters and of its resulting distribution is important for remote sensing and for estimating light penetration in the ocean [10, 5, 6].

Improved understanding of CDOM dynamics requires sensor networks measuring the Neponset River. To ensure a reasonable dataset is collected we can use our numerical model to inform the decentralized control algorithm with depth adjustment. Using this information allows better placement for sensing. The collected data can then be used to calibrate and further improve the Boston Harbor model. In turn, better models will allow improved placement of the nodes for collecting sensory information. We now detail the decentralized control algorithm that can use covariance data derived from these models of the CDOM concentration in the Neponset River.

4 Decentralized Control Algorithm

In this section we give the intuition behind the approach, formulate the problem, develop a general decentralized controller, introduce a Gaussian covariance function, and define the controller in terms of the covariance function. We also prove the convergence of the controller and provide a sensor network implementation.

4.1 Problem Formulation and Intuition

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 $P(i)$. In the case of our underwater sensor network the nodes are constrained to move in 1D along z with fixed x, y.

Intuitively, 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. 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. This correlation is captured by covariance, so

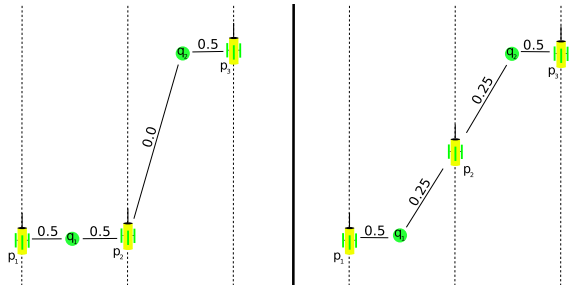


Figure 3. Dashed lines are the motion constraints on the AQUANODE motion, green circles are the points of sensor interest. Solid lines and the labels indicate the covariance between the point of interest and the indicated sensor.

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:

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

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:

$$\arg\max_{p_i} \sum_{j=1}^{M}\sum_{i=1}^{N} Cov(p_i,q_j) \tag{2}$$

This objective function, however, has the problem that some areas may be covered well, while others are not covered. Figure 3 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 3 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 2 and illustrated in Figure 3 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:

$$\arg\max_{p_i} \sum_{j=1}^{M}\left(\sum_{i=1}^{N} Cov(p_i,q_j)\right)^{-1} \tag{3}$$

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.

This changes the example at left in Figure 3 to give an objective value of $(.5+.5)^{-1}+.5^{-1}=3$. It changes the right side in Figure 3 to $(.5+.25^{-1}+(.5+.25)^{-1}=2\frac{2}{3}$. Our new minimization of the objective function will select the rightmost configuration in Figure 3, 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:

$$\int_Q \left(\sum_{i=1}^{N} Cov(p_i,q)\right)^{-1} dq \tag{4}$$

4.2 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:

$$g(q,p_1,...,p_N) = \left(\sum_{i=1}^{N} f(p_i,q)\right)^{-1} \tag{5}$$

This is the inside of Eqn 4 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:

$$\mathcal{H}(p_1,...,p_N) = \int_Q g(q,p_1,...,p_N)\,dq + \sum_{i=1}^{N}\phi(p_i) \tag{6}$$

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:

$$\phi(p_i) = \left(\frac{z_i-(d_i/2)}{d_i/2}\right)^{\beta} \tag{7}$$

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

4.3 General Decentralized Controller

Given the objective function in Equation 6, 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:

$$\begin{aligned}\frac{\partial\mathcal{H}}{\partial z_i} &= \frac{\partial}{\partial z_i}\int_Q g(q,p_1,...,p_N)\,dq + \frac{\partial}{\partial z_i}\sum_{j=1}^{N}\phi(p_j)\\ &= \int_Q -g(q,p_1,...,p_N)^2\frac{\partial}{\partial z_i}f(p_i,q)\,dq + \frac{\partial}{\partial z_i}\phi(p_i)\end{aligned} \tag{8}$$

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

$$\frac{\partial}{\partial z_i}\phi(p_i) = \frac{\partial}{\partial z_i}\left(\frac{z_i-(d_i/2)}{d_i/2}\right)^{\beta} = \beta\left(\frac{z_i-(d_i/2)}{d_i/2}\right)^{\beta-1} \tag{9}$$

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:

$$\dot{p}_i = -k\frac{\partial\mathcal{H}}{\partial z_i} \tag{10}$$

where k is some scalar constant. This provides a general controller usable for any sensing function, $f(p_i,q)$. To use this controller we next present a practical function for $f(p_i,q)$.

4.4 Gaussian Sensing Function

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

$$f(p_i,q) = Cov(p_i,q) \tag{11}$$

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 [25]. In Section 5.1 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 our sensing function as:

$$f(p_i,q) = Cov(p_i,q) = Ae^{-\left(\frac{(x_i-x_q)^2+(y_i-y_q)^2}{2\sigma_s^2}+\frac{(z_i-z_q)^2}{2\sigma_d^2}\right)} \tag{12}$$

where A is a constant related to the two variances, which can be set to 1 for simplicity.

4.5 Gaussian-Based Decentralized Controller

We take the partial derivative of the sensing function from Equation 12 to complete the gradient of our objective function shown in Equation 8. The gradient of the sensing function $\frac{\partial}{\partial z_i} f(p_i,q)$ is:

$$\begin{aligned}\frac{\partial}{\partial z_i} f(p_i,q) &= \frac{\partial}{\partial z_i} Ae^{-\left(\frac{(x_i-x_q)^2+(y_i-y_q)^2}{2\sigma_s^2}+\frac{(z_i-z_q)^2}{2\sigma_d^2}\right)} \\ &= -f(p_i,q)\frac{(z_i-z_q)}{\sigma_d^2}\end{aligned} \tag{13}$$

Substituting this into Equation 8, we get the objective function:

$$\begin{aligned}\frac{\partial \mathcal{H}}{\partial z_i} &= -\int_Q \left(\sum_{j=1}^{N} f(p_j,q)\right)^{-2} \frac{\partial}{\partial z_i} f(p_i,q)^{-1}\, dq + \frac{\partial}{\partial z_i}\phi(p_i) \\ &= \int_Q g(q,p_1,...,p_N)^2 f(p_i,q)\frac{(z_i-z_q)}{\sigma_d^2} dq + \frac{\partial}{\partial z_i}\phi(p_i)\end{aligned} \tag{14}$$

4.6 Controller Convergence

To prove that our gradient controller (equation 10) converges to a critical point of $\mathcal{H}$, we must show [9, 23, 31]:

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 [31].

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:*

$$\lim_{t\leftarrow\infty} -k\frac{\partial \mathcal{H}}{\partial z_i} = 0 \tag{15}$$

PROOF. We show that the objective function satisfies the conditions outlined above. In Section 4.5 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:

$$\mathcal{H}(p_1,...,p_N) = \int_Q g(q,p_1,...,p_N)\, dq + \sum_{i=1}^{N}\phi(p_i) \tag{16}$$

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:

$$\int_Q g(q,p_1,...,p_N)\, dq = \int_Q \left(\sum_{j=1}^{N} f(p_j,q)\right)^{-1} dq \tag{17}$$

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 our system are along the z axis. Equation 7 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 our controller $-k\frac{\partial \mathcal{H}}{\partial z_i}$ converges. □

4.7 Implementation

Algorithm 1 shows the implementation of the decentralized depth controller (Equation 8) 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, $[x,y,z]$, 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.

Algorithm 1 Decentralized Depth Controller

```
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:    changeDepth(delta)
22: end procedure
```

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. Finally, the procedure changes the node depth. Note that for this implementation we ignore the impact of $\phi(p_i)$ on the controller. Instead, `changeDepth` puts a hard limit on the range of the nodes, preventing them from moving out of the water column. We found that this had little impact on the results.

5 Simulation and Hardware Experiments

In this section we discuss the results of simulation experiments and of experiments performed in a pool and river with our underwater sensor network hardware. First we detail how to obtain a covariance model.

5.1 Covariance Model

In Section 4 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 4 (bottom) shows the concentration of CDOM in the Neponset River, which feeds into Boston Harbor at two different tide levels. We find that the CDOM concentration is highly dependent on the tide level. Other parameters that may impact CDOM concentration include rainfall and wind patters. This data was computed from a numerical model for Boston Harbor as described in Section 3. The model is accurate on average, however, new measurements from the AQUANODES will be used to further refine and calibrate the Neponset River and Boston Harbor models.

Figure 4 (top) shows the numerically computed covariance along the length of the river and 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.

Most of the sensors we use only measure the value at the location of the sensor node. Using covariance allows us to expand the range of the sensor by determining the likely relation between the location of the sensor and other locations. This can fail in some situations where there are discontinuities in the sensory field, for instance strong currents bringing in different water. However, the covariance function can be adjusted to take into account this type of geographic dependence if these types of discontinuities are expected or detected.

We fit a Gaussian basis function to the numeric covariance curve. To do this we use Matlab's `newrb` function. Figure 5 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.

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 [26].

Figure 4 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. Section 5.4 discusses changing covariance functions to account for tidal changes in a river.

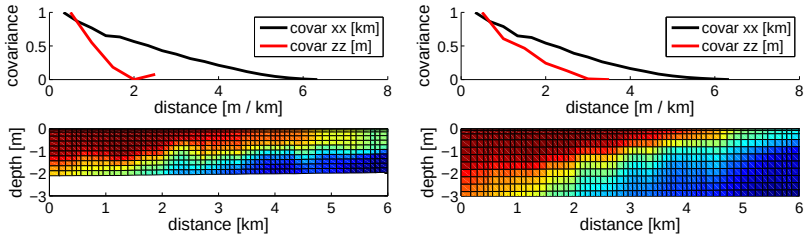
5.2 Simulation Experiments

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

In these experiments, unless otherwise noted, we use a "k" value of 0.001, capped with a maximum speed of ± 2 m/s. 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. In Section 6, we provide a detailed analysis of parameters used in the algorithm.

Figure 6(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.

Figure 6(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.



(a) (b)

Figure 4. 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.

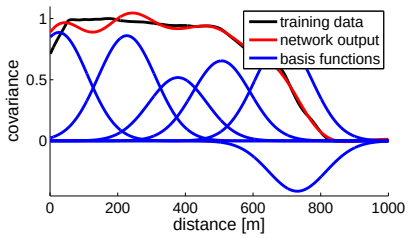
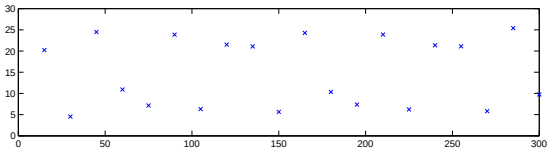
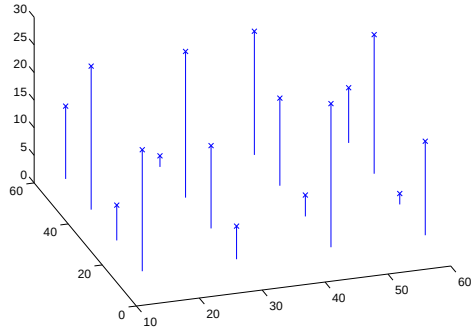


Figure 5. The Gaussian basis function elements for a fit with 6 basis functions.



(a)



(b)

Figure 6. The final positions after the distributed controller converges for a (a)2D and (b)3D setup.

5.3 Lab and Pool Hardware Experiments

We performed experiments in the lab and in a pool using four of our underwater sensor nodes. For both setups the sensor network ran the decentralized depth controller. We tested both the base covariance model discussed in Section 4.4 and the model-based covariance discussed in Section 5.1.

We implemented Algorithm 1 on the system. For the implementation of `FDz(p_i,qx,qy,qz)` (line 10) we use the numerical gradient of `F` at that position. We use this to avoid deriving the gradient for every covariance function. The two different covariance models require different implementations of the function `F` (line 8).

For the base covariance model discussed in Section 4.4 we implemented the `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.

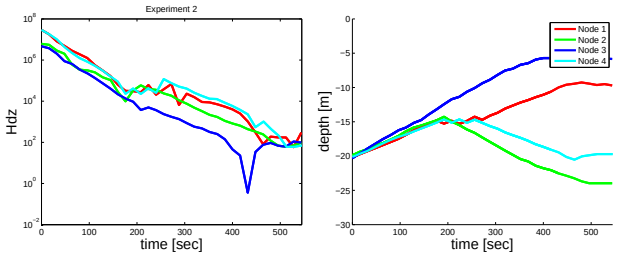
For the covariance based on the model data (CDOM covariance) we implemented `F` based on the Gaussian basis function. Since we used Matlab's `newrb` function to compute the basis function, we used their documentation to determine the reconstruction of `F`:

```
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 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.

With 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 winch 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 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.



(a) (b)

Figure 7. (a) The value of $\frac{\partial \mathcal{H}}{\partial z_i}$. (b) The depths over the course of a experiment.

	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 1. Selected start and end configurations with the base covariance function.

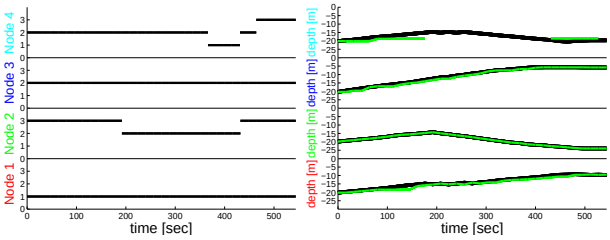
5.3.1 Results

We successfully ran multiple 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.

Figure 7(a) depicts 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 on a log-scale. 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. The dip in the objective function for one node seen in Figure 7(a) is caused by a temporary configuration that was slightly better from the standpoint of the one node and is amplified by the log-scale of the plot.

Figure 7(b) shows the depths of each of the nodes over the course of the same 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 1 shows the start and end configurations for some 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.



(a) (b)

Figure 8. (a) Number of neighbors used to calculate $\frac{\partial \mathcal{H}}{\partial z_i}$. (b) One node's estimate the others' depths versus actual.

	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 2. Selected start and end configurations with river covariance function.

Similarly, Table 2 shows the start and end configurations for some of the pool and bucket experiments for the river model covariance function. A different local minimum of up-mid-down-up can be seen in trial Pool 2. Section 6.2.1 explores how different start configurations effect the final positioning of the nodes.

5.3.2 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 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 [27, 34, 36].

In our implementation, each node only used depth information from neighboring nodes whose last depth message was received within the past two minutes. Figure 8(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 8(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 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 converges and is robust.

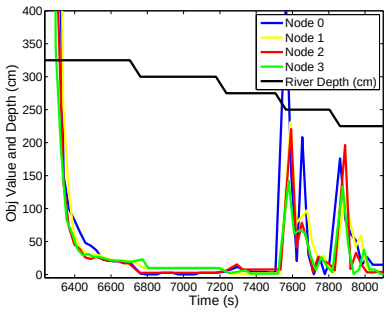
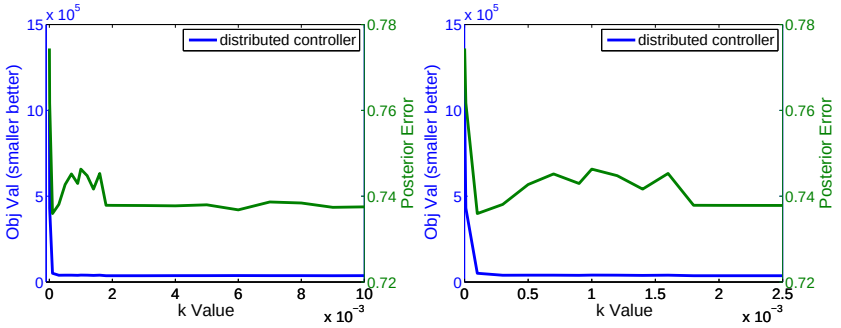


Figure 9. Objective value for 4 nodes when changing the depth-based covariance function.



(a) (b)

Figure 10. The objective value and posterior error 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 River Hardware Experiment With Changing Covariance

We performed experiments to characterize the performance of the decentralized depth adjustment algorithm when the covariance function changes periodically. Figure 4(bottom) shows the concentration of CDOM along the Neponset River based on the model described in Section 5.1 for 2.0m and 3.0m of water. Changes in water level are due to tidal effects. Figure 4(top) shows the numeric covariance for each of these plots normalized to fall between zero and one.

We deployed 4 nodes in the Charles River in Cambridge, MA and simulated updating the covariance function to determine the effect of periodic covariance function updates. We used five different covariance functions based on 3.25m, 3.0m, 2.75m, 2.5m, and 2.25m of average water depth. Figure 9 shows the results. This figure plots the value of the objective function that each node computes over time as well which covariance function is currently in use (step function at top of Figure).

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 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 provides initial verification of the hardware system in a river environment. In addition, it 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.

6 Analysis and Discussion

Several practical considerations arise in implementing this controller on real hardware. In this section we discuss parameter sensitivity, positioning sensitivity, comparison to other methods, and data reconstruction.

6.1 Parameter Sensitivity

We analyze k, neighborhood size, and grid size.

6.1.1 *Changing k*

Figure 10 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:

$$-k\frac{\partial \mathcal{H}}{\partial z_i} \tag{18}$$

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.

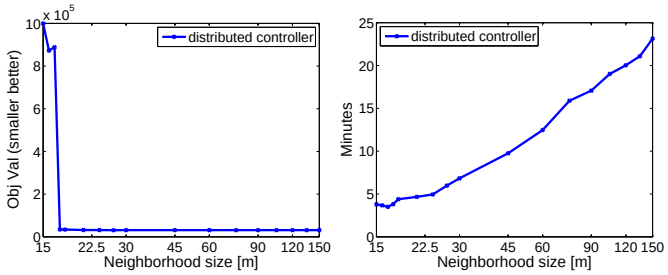
Figure 10(a) shows the range of k's explored, while Figure 10(b) zooms in on k values less then 0.003. Using a k value of less than 0.0005 yields very poor results since the system does not converge within the 20 iterations of the algorithm that we allow. However, values larger than this perform well.

In Figure 10(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 they system tends to converge and not oscillate.

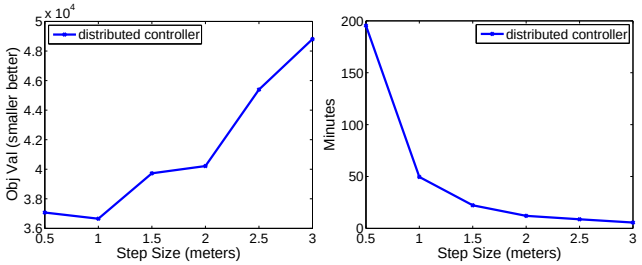
6.1.2 *Changing Neighborhood Size*

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



(a) (b)

Figure 11. The (a) objective value and (b) runtime for a 15 node network when changing the size of the neighborhood over which the integration occurs.



(a) (b)

Figure 12. Changing the grid size. (a) Objective value and (b) total search time as the step size changes.

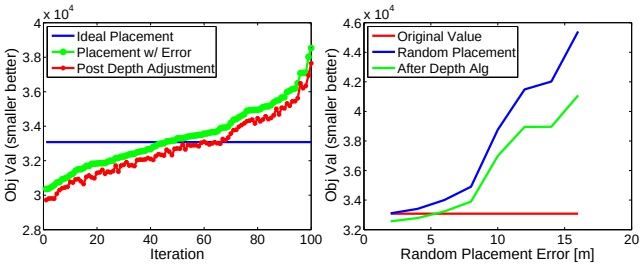
due to poor acoustic communication. The covariance function decays rapidly with distance, reducing the affect of far away sensors, allowing nodes to ignore the sensors they cannot hear. This means that multi-hop messaging is not needed to communicate information for the depth controller, enabling very large deployments without communication overhead.

Figure 11 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 size varied from $\pm$15m to $\pm$150m. As can be seen from Figure 11 (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. Figure 11 (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.

6.1.3 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



(a) (b)

Figure 14. Nodes deployed with random error in x-axis placement. (a) Plot of 100 runs with 6m error. (b) Average over many runs and positions.

along the x-axis and N is the number of sensor nodes in the system. As shown in Figure 12 (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 12 (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.

6.2 Positioning Sensitivity

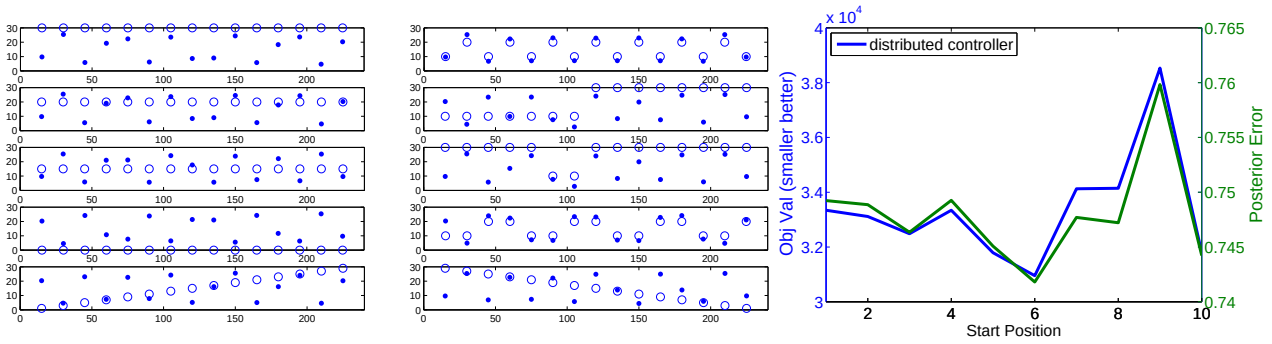
We analyze start configurations and placement error.

6.2.1 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 13 (a-b) 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 start 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 13(b).9, which alternated two down and two up resulted in a similar final configuration. As can be seen in Figure 13 (c) this configuration yielded the worst objective value and posterior error of all the trials.

We can contrast this with the configuration in Figure 13(b).6, which resulted in the best objective value. The remainder of the configurations fell somewhere in between 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. It is possible to avoid local minima by starting with configurations that are known to be near-optimal (for example, up-down-up configurations). In addition, in situations where the covariance function is periodically updated, the nodes can return to a near-optimal configuration before performing the update.



(a) (b) (c)

Figure 13. (a-b)The results of the running the depth adjustment algorithm on various node start positions (circles) and (a) the resultant objective value and posterior error.

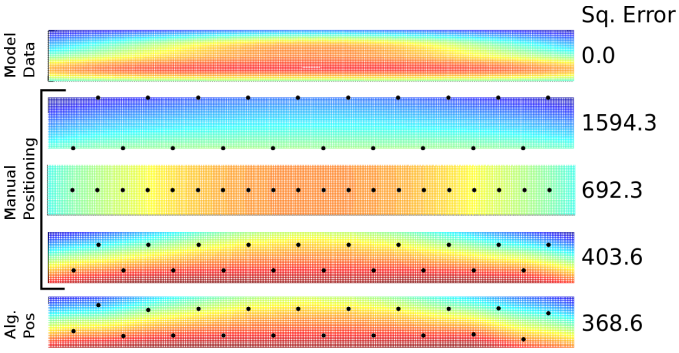


Figure 15. From top: model data, reconstructed data for three manual configs, and for algorithm positioning.

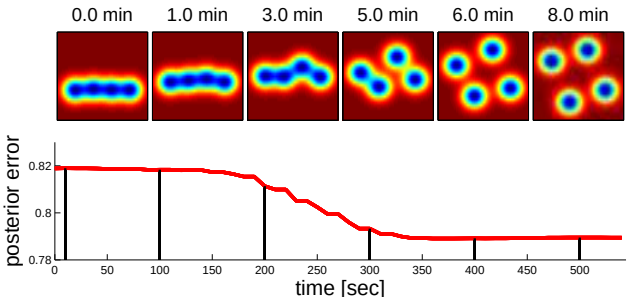


Figure 16. Posterior error.

6.2.2 *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 14(a) demonstrates the results of 100 trials with $\pm$6m random error added to the ideal node positions in the x-axis. The 100 trials are sorted by the objective function in the plot. We then run the depth adjustment algorithm on the mispositioned nodes, which improves their overall depth positioning for sensing.

Figure 14(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 adjustment algorithm can improve the overall sensing even if the start depths (z-axis) are ideal and x-axis placement is not.

6.3 Data Reconstruction

The ultimate goal of placing sensors 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 15 shows the results of reconstructing a field given three manually configured sensor placements and the depth adjustment algorithm. At top in Figure 15 we show the actual field we attempt to recover. This field is a semi-randomly chosen field that 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 sum of squared error values are shown in the right of Figure 15. The dynamic depth adjustment algorithm outperforms the three manually chosen configurations.

6.4 Comparison to Other Methods

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

6.4.1 *Posterior Error*

A common metric for defining how an area is covered by sensors is to examine the posterior error of the system [18]. 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:

$$\sigma^2_{q|P} = Cov(q,q) - \Sigma_{q,P} \cdot \Sigma^{-1}_{P,P} \cdot \Sigma_{P,q} \qquad (19)$$

The vector $\Sigma_{q,P}$ is the vector of covariances between q and the sensor node positions $P = \{p_1, ..., p_N\}$. The vector $\Sigma_{P,q}$

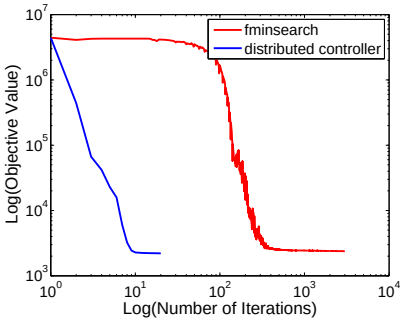
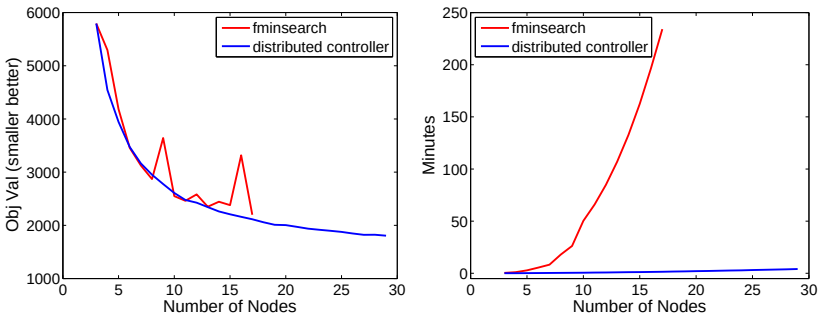


Figure 17. The objective value versus the number of iterations for the decentralized controller and fminsearch.



(a) (b)

Figure 18. The objective value (a) and total search time (b) for the decentralized controller and fminsearch.

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 13(c), 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 16 (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 16 (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.

6.4.2 *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 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, it does not, however, make use of the derivative of the objective function. In our simulations we find that `fminsearch` and our algorithm optimize the objective function similarly for small numbers of nodes. However, `fminsearch` has poor runtime and is thus limited to running on systems with few nodes.

Figure 17 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 18(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` 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 18(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 6.1.2, 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.

These experiments show that the distributed depth adjustment algorithm obtains similar or lower objective values in most all cases than a standard nonlinear optimizer. In addition, the distributed depth controller provides a much more reasonable linear increase in computation time per node compared to `fminsearch`.

7 Related Work

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.* simulates distributing underwater sensors to maximize coverage of a region by breaking the region into cubes and filling each cube [11]. Akkaya *et al.* simulate optimizing sensor positioning by spreading out the nodes while maintaining communication links in an underwater sensor network with depth adjusting capabilities [2]. Our approach is different in that we account for sensing covariances, use realistic communication assumptions, and implement and test the algorithm on a real underwater sensor network.

Ko *et al.* develop an algorithm for sampling at informative locations based on minimizing the entropy [22]. Guestrin *et al.* introduce the optimization criterion called mutual information [18]. 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 [24]. Yilmas *et al.* plan a path for an AUV that will decrease measurement uncertainty using a linear programming approach [37]. 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 [28].

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 Section 6.4.1, that our algorithm also tends to minimize the posterior error criteria, while requiring less computational complexity. Our algorithm also runs continuously and can adapt to changing conditions.

Further, our algorithm uses a decentralized gradient-descent controller. Cortes *et al.* use a decentralized gradient controller to perform Voronoi tessellations for a known event distribution [12]. Details on these types of algorithms can be found in the book by Bullo *et al.* [9]. Schwager *et al.* extend these controllers to learn the underlying sensing function through consensus [32]. Our work draws inspiration from these techniques, but differs in 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 and is a common technique. Leonard *et al.* use a multivariate Gaussian function, similar to ours, to estimate the covariance in their system [24]. 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 [25]. They use the covariance models they develop as input into objective analysis models. Objective analysis is statistical estimation under Gauss-Markov conditions. 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.

A number of trial and longer-term underwater sensor network systems have been deployed. MOON aims to create an ocean observatory in the Mediterranean Sea for monitoring and forecasting weather, environmental monitoring, and marine research [1]. MOON is part of the Global Ocean Observing System (GOOS) which was created in the 1980s to monitor all of the world's oceans [3]. The Ocean Observatories Initiative (OOI) combines deep-sea buoys, cabled underwater networks, as well as independent AUVs and sensors [4, 19]. Jannasch *et al.* have developed a system of statically moored ocean sensors to monitor environmental processes off the coast of California in Monterey Bay [20].

Our depth adjustment system is a novel contribution to the field of underwater sensor networks. Other underwater systems have made use of column profilers, however, they have not been integrated as a key component of the underwater sensor network.

8 Conclusions and Future Work

In this paper we present a gradient-based decentralized controller that dynamically adjusts the depth of a network of underwater sensors to optimize sensing. We prove that the controller converges. We implement two covariance models: a multivariate Gaussian and one derived from a physics-based hydrodynamic model. We perform extensive simulations and experiments on our sensor network platform verifying the functionality.

Deploying underwater sensor networks entails numerous challenges that are not found in most terrestrial sensor networks. First and foremost, the housings must be designed to keep the electronics dry, while maintaining easy access for adding sensors, debugging, and reconfiguring. Finding suitable test locations is also challenging. Acoustic communication, already slow, has even worse performance in confined pools. Rivers and near-shore ocean deployments are difficult due to heavy boat traffic which make accessing the sensors challenging and potentially dangerous. In addition, it is nearly impossible to recharge the batteries once deployed. Finally, the enormous size of the ocean makes full sensor coverage unlikely. All of these challenges call for the development of new algorithms and systems that optimize sensing, communication, and energy usage simultaneously.

The depth adjustment system adds a number of capabilities to our underwater sensor network that ease some of these challenges. These include ability to: surface to use the radio; surface to obtain a GPS location fix; surface to ease node retrieval; surface to recharge via solar panels; go to the bottom to avoid boat traffic; and change depth to optimize other parameters such as communication.

We have performed deployments in the Charles River and preliminary, larger scale deployments Neponset River (not reported on here). In the future we plan to perform long term, large scale deployments of the system to verify the longevity of the system and collect detailed time series data. We are also exploring variations of the algorithm that maintain communication links while optimizing depth for sensing. Finally, we are looking at algorithms to minimize the communication and motion to maximize deployment time.

9 Acknowledgements

We are grateful to DSO Singapore, MURI Antidote (138802), MURI SMARTS (N00014-09-1-1051), and NSF ITR (IIS-0426838) for supporting parts of this research. We also thank Elizabeth Basha, Bernie Gardner, and Francesco Peri for their help on this project.

10 References

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

[2] 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.

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

[4] 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.

[5] 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.

[6] N. V. Blough and R. D. Vecchio. Chromophoric dom in the coastal environment. In D. A. Hansell and C. A. Carlson, editors, *Biogeochemistry of Marine Dissolved Organic Matter*, pages 509 – 546. Academic Press, San Diego, 2002.

[7] A. Blumberg, R. Signell, and H. Jenter. Modelling transport processes in the coastal ocean. *J. Marine Env. Engg.*, 1:31–52, 1993.

[8] 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.

[9] F. Bullo, J. Cortés, and S. Mortínez. *Distributed Control of Robotic Networks*. Applied Mathematics Series. Princeton University Press, 2009.

[10] 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.

[11] 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.

[12] 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.

[13] C. Detweiler, M. Doniec, I. Vasilescu, E. Basha, and D. Rus. Autonomous depth adjustment for underwater sensor networks. In *International Workshop on Underwater Networks (WUWNet 2010)*, Woods Hole, Massachusetts, USA, Sept. 2010.

[14] C. Detweiler, J. Leonard, D. Rus, and S. Teller. Passive mobile robot localization within a fixed beacon field. In *Proceedings of the 2006 International Workshop on Algorithmic Foundations of Robotics*, New York, aug 2006.

[15] C. Detweiler, I. Vasilescu, and D. Rus. An underwater sensor network with dual communications, sensing, and mobility. In *OCEANS 2007 - Europe*, pages 1–6, 2007.

[16] J. J. Dodson, J. C. Dauvin, R. G. Ingram, and B. d'Anglejan. Abundance of larval rainbow smelt (osmerus mordax) in relation to the maximum turbidity zone and associated macroplanktonic fauna of the middle st. lawrence estuary. *Estuaries*, 12(2):66–81, 1989.

[17] 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.

[18] C. Guestrin, A. Krause, and A. P. 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.

[19] A. Isern. The ocean observatories initiative: Wiring the ocean for interactive scientific discovery. In *OCEANS 2006*, pages 1–7, 2006.

[20] 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.

[21] 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.

[22] C. Ko, J. Lee, and M. Queyranne. An exact algorithm for maximum entropy sampling. *Operations Research*, 43(4):684–691, Aug. 1995. ArticleType: primary_article / Full publication date: Jul. - Aug., 1995 / Copyright 1995 INFORMS.

[23] J. LaSalle. Some extensions of lyapunov's second method. *IRE Transactions on Circuit Theory*, 7(4):520–527, 1960.

[24] N. Leonard, D. Paley, F. Lekien, R. Sepulchre, D. Fratantoni, and R. Davis. Collective motion, sensor networks, and ocean sampling. *Proceedings of the IEEE*, 95(1):48–74, 2007.

[25] D. R. Lynch and D. J. McGillicuddy. Objective analysis for coastal regimes. *Continental Shelf Research*, 21(11-12):1299–1315, July 2001.

[26] J. Park and I. W. Sandberg. Universal approximation using radial-basis-function networks. *Neural Comput.*, 3(2):246–257, 1991.

[27] J. Partan, J. Kurose, and B. N. Levine. A survey of practical issues in underwater networks. *SIGMOBILE Mob. Comput. Comm. Rev.*, 11(4):23–33, 2007.

[28] P. Rigby. *Autonomous Spatial Analysis using Gaussian Process Models*. PhD thesis, University of Sydney, 2008.

[29] M. R. Roman. Temporal and spatial patterns of zooplankton in the chesapeake bay turbidity maximum. *Mar. Ecol. Prog. Ser.*, 213:215–227, 2001.

[30] J. R. Schubel. Turbidity maximum of the northern chesapeake bay. *Science*, 161:1013 – 1015, 1968.

[31] M. Schwager. *A Gradient Optimization Approach to Adaptive Multi-Robot Control*. PhD thesis, MIT, 2009.

[32] M. Schwager, D. Rus, and J. Slotine. Decentralized, adaptive coverage control for networked robots. *The International Journal of Robotics Research*, 28(3):357–375, Mar. 2009.

[33] R. Signell, H. Jenter, and A. Blumberg. Predicting the physical effects of relocating boston's sewage outfall. *Estuarine, Coastal and Shelf Science*, 50(1):59 – 71, 2000.

[34] 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.

[35] I. Vasilescu, C. Detweiler, and D. Rus. AquaNodes: an underwater sensor network. In *Proc. of 2nd WUWNet*, pages 85–88, Montreal, Quebec, Canada, 2007. ACM.

[36] 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.

[37] N. Yilmaz, C. Evangelinos, P. Lermusiaux, and N. 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.