AQUATIC ENVIRONMENT MONITORING USING ROBOTIC SENSOR NETWORKS

By

Yu Wang

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

Computer Science – Doctor of Philosophy

2015

ABSTRACT

AQUATIC ENVIRONMENT MONITORING USING ROBOTIC SENSOR NETWORKS

By

Yu Wang

Aquatic environment has been facing an increasing number of threats from various harmful aquatic processes such as oil spills, harmful algal blooms (HABs), and aquatic debris. These processes greatly endanger the aquatic ecosystems, marine life, human health, and water transport. Hence, it is of great interest to detect these processes and monitor their evolution such that proper actions can be taken to prevent the potential risks. This dissertation explores four representative problems in aquatic environment monitoring, which include diffusion processing profiling, spatiotemporal field reconstruction, aquatic debris surveillance, and water surface monitoring.

First, we propose an accuracy-aware approach to profiling an aquatic diffusion process such as oil spill. In our approach, the robotic sensors collaboratively profile the characteristics of a diffusion process including its source location, discharged substance amount, and evolution over time. In particular, the robotic sensors reposition themselves to progressively improve the profiling accuracy. We formulate a novel movement scheduling problem that aims to maximize the profiling accuracy subject to limited sensor mobility and energy budget. To solve this problem, we develop an efficient gradient-ascent-based algorithm and a near-optimal dynamic-programming-based algorithm.

Second, we present a novel approach to reconstructing a spatiotemporal aquatic field such as HABs. This approach features a rendezvous-based mobility control scheme where robotic sensors collaborate in the form of a swarm to sense the aquatic environment in a series of carefully chosen rendezvous regions. We design a novel feedback control algorithm that maintains the desirable level of wireless connectivity for a sensor swarm in the presence of significant environment

and system dynamics. Moreover, information-theoretic analysis is used to guide the selection of rendezvous regions so that the reconstruction accuracy is maximized subject to the limited sensor mobility.

Third, we develop a vision-based, cloud-enabled, low-cost, yet intelligent solution to aquatic debris surveillance. Our approach features real-time debris detection and coverage-based rotation scheduling algorithms. Specifically, the image processing algorithms for debris detection are specifically designed to address the unique challenges in aquatic environment, e.g., constant camera shaking due to waves. The rotation scheduling algorithm provides effective coverage of sporadic debris arrivals despite camera's limited angular view. Moreover, we design a dynamic task offloading scheme to offload the computation-intensive processing tasks to the cloud for battery power conservation.

Finally, we design Samba – an aquatic surveillance robot that integrates an off-the-shelf Android smartphone and a robotic fish for general water surface monitoring. Using the built-in camera of on-board smartphone, Samba can detect spatially dispersed aquatic processes. To reduce the excessive false alarms caused by the non-water area, Samba segments the captured images and performs target detection in the identified water area only. We propose a novel approach that leverages the power-efficient inertial sensors on smartphone to assist the image processing. Samba also features a set of lightweight and robust computer vision algorithms, which detect harmful aquatic processes based on their distinctive color features.

This dissertation is dedicated to my family.

ACKNOWLEDGMENTS

I would like to thank my committee members, Professor Guoliang Xing, Professor Xiaobo Tan, Professor Rong Jin, and Professor Li Xiao, for their advise and support in each stage of my graduate study. In addition, I would like to thank Rui Tan, who mentored me during my early graduate years and introduced me to the field of research. And I would like to thank all my current and former colleagues at the experimental Laboratory for Advanced Networks and Systems (eLANS). They include, but are not limited to, Jun Huang, Pei Huang, Liqun Li, Chen Qiu, Bo Su, Yuanteng Pei, Ruogu Zhou, Jinzhu Chen, Tian Hao, Chongguang Bi, Fernando Cintron, Sayeed Choudhary, Mohammad Moazzami, Dennis Philips, and Chin-Jung Liu. Lastly but most importantly, I would like to thank my family for their continuous support and encouragement throughout my life.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

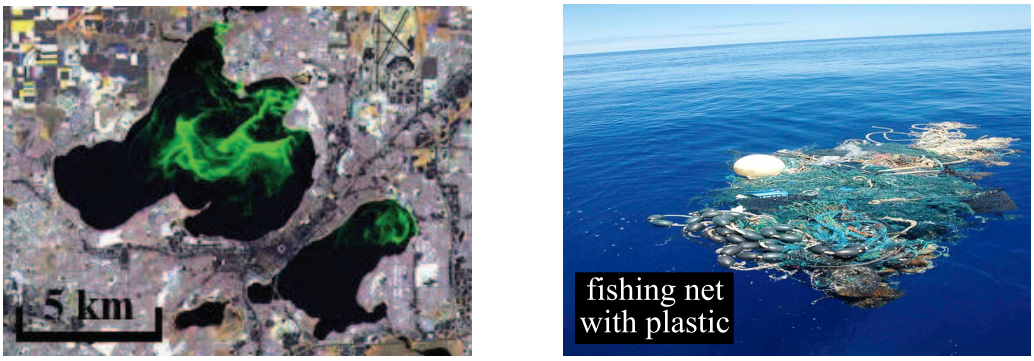
Chapter 1

Introduction

1.1 Aquatic Environment Monitoring

Aquatic environment has been facing an increasing number of biological, chemical, and physical threats from various harmful aquatic processes. The last four decades witnessed more than a dozen major oil spills with each releasing more than 30 million gallons of oil [70], causing severe attendant environmental damage. Other harmful aquatic processes such as harmful algal blooms (HABs) [21], e.g., the HABs occurred on two inland lakes in Wisconsin shown in Figure 1.1(a), also have disastrous impact on public health and ecosystem sustainability. For example, extensive HABs have been observed in many Ohio's inland lakes that supply local drinking water, resulting in 41 confirmed cases of health impacts to humans in 2010 [69]. Recently, aquatic debris – human-created waste found in water environment – has emerged to be another grave environmental issue. The 2011 Japan tsunami released about one million tons of debris that heads toward North America [44], and some has drifted to U.S. West Coast as shown in Figure 1.1(b). These harmful processes pose numerous potential risks to our aquatic environment. It is thus imperative to detect them and monitor their evolution, and alarm the authorities to take preventive actions.

Current approaches to monitoring aquatic environment fall into four basic categories, i.e., manual spotting (e.g., with handhold devices or via patrol boats [3, 9]), *in situ* sensing (e.g., with fixed or buoyed sensors [82]), remote sensing (e.g., balloon-board [46], aircraft [39], and satellite imaging [11, 58]), and AUV-based autonomous sensing [25, 102]. Manual sampling [3, 9] is still a



(a) (b)

Figure 1.1: Representative harmful aquatic processes: (a) HABs on Lake Mendota (top left) and Lake Monona (right bottom) in Wisconsin, 1999 [33] (Photo Credit: University of Wisconsin-Madison and WisconsinView); (b) debris from the Japan tsunami arriving at U.S. West Coast, 2012 [73] (Photo Credit: Scripps Institution of Oceanography).

common practice for small-scale monitoring. However, this method is labor-intensive and difficult to adapt to the dynamics of aquatic environment. An alternative is *in situ* sensing with fixed or buoyed sensors [82]. However, since these sensors cannot move around, it can take a prohibitively large number of them to capture spatially inhomogeneous information. More advanced methods involve remote sensing technologies, e.g., balloon-board [46], aircraft [39], and satellite imaging [11, 58]. The former method is only effective for one-off and short-term monitoring of highly concentrated aquatic processes of interest that have been already detected, and the latter two often have high operational cost and falls short of monitoring resolution. Robotic technologies have been making significant progress over the past couple of decades and applied to aquatic environment monitoring. Autonomous underwater vehicles (AUVs) [25, 102] are notable examples of such technologies, and have been used for various aquatic sensing tasks. However, AUV-based platforms often have high manufacturing costs (over $\$50{,}000$ per unit [83]). In summary, these limitations make remote sensing and AUV-based approaches ill-suited for monitoring spatially scattered and temporally evolving aquatic processes.

The advances in computing, communication, sensing, and actuation technologies have made it

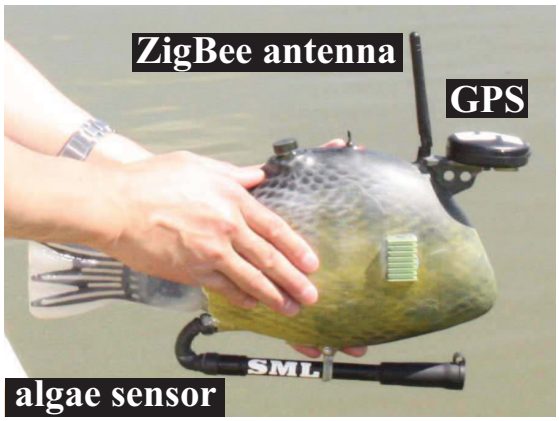


Figure 1.2: Prototype of autonomous robotic fish [96].

possible to create untethered robotic fish with on-board power, control, navigation, wireless communication, and sensing modules, which turn these robots into mobile sensing platforms in aquatic environment. Figure 1.2 shows a prototype of such platforms [96], which costs around $300. Due to the low manufacturing cost, these platforms can be massively deployed to form a mobile sensor network that monitors aquatic processes of interest, providing significantly higher spatial and temporal sensing resolution than the existing monitoring methods. Moreover, a school of robotic fish can coordinate their sensing and movements through wireless communication enabled by the on-board ZigBee radio, to adapt to the dynamics of evolution. In Chapters 3 and 4, we will employ this robotic sensing platform to address diffusion process profiling and spatiotemporal field reconstruction, respectively.

Recently, mobile sensing based on smartphones has received increasing research interest due to their rich computation, communication, and storage resources. Figure 1.3 shows a prototype of smartphone-based robotic sensing platform built with a gliding robotic fish [31] and a Samsung Galaxy Nexus smartphone. The integrated smartphone and robotic fish assemble a promising platform for aquatic environment monitoring due to the following salient advantages. The robotic fish [31] is a low-cost (about $3,000 per unit) aquatic mobile platform with high maneuverability

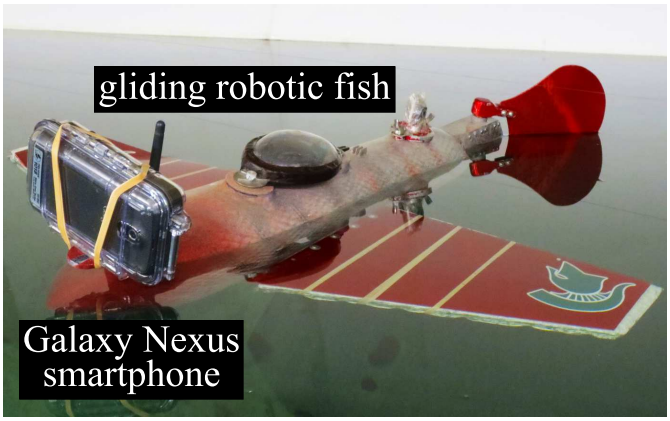


Figure 1.3: Prototype of smartphone-based robotic sensing platform [111] that integrates a Samsung Galaxy Nexus smartphone in a water-proof enclosure with a gliding robotic fish [31].

in rotation and orientation maintenance, enabling it to adapt to the dynamic aquatic environment. Moreover, it has stronger resistance to capsizing than robotic boats (e.g., [23]) due to the enhanced stability from its bionic design. The on-board cameras of smartphone provide an inexpensive yet high-resolution sensing modality for detecting the harmful aquatic processes. For example, HABs, as shown in Figure 1.1(a), can be detected using the phone's built-in cameras based on their distinctive colors. Moreover, in addition to camera, a few other sensors such as accelerometer and gyroscope, which are commonly available on smartphone, can assist the navigation and control of robotic fish. Second, compared with traditional chemical sensors that measure only one location at a time, camera has a wider sensing range and provides richer information about the aquatic process such as color and spatial distribution. Such information is often important for the authorities to conduct hazard analysis and take contingency measures. Third, current smartphones are capable of running advanced computer vision (CV) algorithms for real-time image processing. Lastly, the price of smartphone has been dropping drastically in the past few years. Owing to these features, the smartphone-based robotic sensor represents an energy-efficient, low-cost, yet intelligent mobile sensing platform for aquatic environment monitoring. In Chapters 5 and 6, we will explore the usage of this smartphone-based robotic sensing platform.

1.2 Contribution of This Dissertation

This dissertation is focused on monitoring aquatic environment using the robotic sensors shown in Figures 1.2 and 1.3. Specifically, we explore four representative problems, i.e., diffusion processing profiling in Chapter 3, spatiotemporal field reconstruction in Chapter 4, aquatic debris surveillance in Chapter 5, and water surface monitoring in Chapter 6. We propose several novel algorithms to address the unique challenges in aquatic environment monitoring, which include a near-optimal sensor movement scheduling algorithm for diffusion profiling [108, 109], a rendezvous-based swarm mobility control scheme for field reconstruction [106, 107], CV algorithms for on-water target detection [110, 111], a dynamic task offloading scheme to utilize cloud computing [111], a sensor fusion scheme that integrates multiple sensing modalities [110], etc. In particular, this dissertation makes the following major contributions:

1) We propose an accuracy-aware approach to profiling an aquatic diffusion process with robotic sensor networks. We first derive the analytical profiling accuracy based on the Cramér-Rao bound (CRB). We then formulate a sensor movement scheduling problem for aquatic diffusion profiling, where the CRB-based profiling accuracy is maximized under the constraints on sensor mobility and energy budget. To solve this problem, we develop gradient-ascent-based *greedy* and dynamic-programming-based *radial* movement scheduling algorithms.

2) We propose a novel approach to sampling and reconstructing a spatiotemporal aquatic field using a robotic sensor swarm. This approach features a rendezvous-based mobility control scheme. Specifically, a feedback control algorithm maintains the desirable level of wireless connectivity of the sensor swarm in the presence of significant physical dynamics, and an swarm movement scheduling algorithm guides the selection of rendezvous regions so that the reconstruction accuracy can be maximized.

3) We develop several lightweight CV algorithms, including an image registration algorithm that mitigates the impact of camera shaking and an adaptive background subtraction algorithm that reliably detects debris objects, to address the inherent dynamics in aquatic debris detection. Moreover, we propose a novel approach to dynamically offloading these CV tasks to the cloud. Lastly, we design a robot rotation scheduling algorithm that minimizes the movement energy consumption while maintaining a desired level of debris coverage performance.

4) We develop Samba - an aquatic robot that integrates an Android smartphone with a gliding robotic fish [31] for general water surface monitoring. Moreover, we propose an efficient hybrid image segmentation approach to identifying the area of interest in an image by combining visual and inertial sensing. Lastly, we enhance the robot situation awareness with the fusion of various on-board inertial sensors of smartphone.

1.3 Dissertation Organization

This rest of this dissertation is organized as follows. Chapter 2 reviews the related work in aquatic environment monitoring. Chapter 3 addresses diffusion process profiling. Chapter 4 studies spatiotemporal field reconstruction. Chapter 5 explores aquatic debris surveillance. Chapter 6 presents Samba – an aquatic robot designed for water surface monitoring. Chapter 7 concludes this dissertation.

Chapter 2

Related Work

Monitoring aquatic environment using networked sensor systems has recently received increasing interest. Early work focuses on stationary sensor deployment. In [47], positions of sensors are selected before real deployment to reduce the uncertainty in reconstructing a spatial physical field that follows the Gaussian process. However, the proposed algorithms are computationally intensive and hence can only be executed offline. Recently, mobility has been exploited to enhance the adaptability and sensing capability of sensor systems. In [116], a robotic boat supplements a static sensor network to reduce the error of field reconstruction, where the boat's movement is guided by the measurements of the sensor network. Another study [85] develops active learning schemes for mobile sensor networks, which plan the movements of mobile sensors based on the feedback of previous measurements. Heuristic movement scheduling algorithms proposed in [92] estimate the contours of a physical field. In [114], the movement of mobile sensors is directed to reduce the uncertainties in estimating the field variables at a set of pre-specified locations. The algorithms developed for placing stationary sensors in [47] are extended to schedule the movement of a mobile sensor network in reconstructing a Gaussian process [86]. However, the aforementioned studies do not account for the constraints of low-power robotic sensors, such as the limited motion, computation, and communication capabilities, and usually adopt complicated numerical optimization, making them only applicable to a small number (e.g., 3 in [90]) of powerful robots. In this work, we focus on developing movement scheduling algorithms for moderate- or large-scale mobile networks that are composed of inexpensive robotic sensors shown in Figures 1.2 and 1.3. Moreover,

the previous studies generally focus on the open-loop solutions that often fail to adapt to the highly complex and dynamic aquatic environment. In contrast, we aim to develop practical and adaptive wireless communication, sensing, and movement control approaches for mobile sensor systems in aquatic field reconstruction.

2.1 Diffusion Process Profiling

Most previous work on diffusion process monitoring is based on stationary sensor networks. Several estimation techniques are adopted by these studies, which include *state-space filtering*, *statistical signal processing*, and *geometric trilateration*. The state space approach [81, 112] uses discrete state-space equations to approximate the partial differential equations that govern the diffusion process, and then applies filtering algorithms such as Kalman filter [81, 112] to profile the diffusion process based on noisy measurements. In the statistical signal processing approach, several estimation techniques such as MLE [64, 103] and Bayesian parameter estimation [117] are applied to deal with noisy measurements. For example, in [103], an MLE-based diffusion characterization algorithm is designed based on binary sensor measurements to reduce the communication cost. In [117], the parametric probability distribution of the diffusion profile parameters is passed among sensors and updated with sensor measurements by Bayesian estimation. In geometric trilateration approach [15, 62], the measurement of a sensor is mapped to the distance from the sensor to the diffusion source. The source location can then be estimated by trilateration among multiple sensors. Such an approach incurs low computational complexities, but suffers lower estimation accuracy compared with more advanced approaches such as MLE [15].

2.2 Spatiotemporal Field Reconstruction

Recently, swarm-based systems have been proposed for various sensing applications. Representative examples include RoboBee [18] and SensorFly [74]. These studies mainly focus on hardware design and system issues. In contrast, we address the field reconstruction problem using a robotic sensor swarm. Based on key observations from real data traces of robotic sensors' wireless communication and field measurements, we formulate the swarm connectivity control and movement scheduling problems, and solve them using control- and information-theoretic algorithms. Most previous works on maintaining sensor network connectivity adopt the graph theory [114] and the potential field theory [30], and assume fixed communication range and reliable communication quality. However, several studies have revealed significant stochasticity and irregularity in link quality of low-power wireless sensors [14, 59, 75, 118]. Feedback control has been widely adopted to improve the adaptability of computing systems [1, 35, 54, 56]. Different from these existing solutions, our control-theoretic connectivity maintenance algorithm specifically deals with the dynamics caused by movement of robotic sensor swarm and disturbances from the aquatic environment. Mobility has been used to improve link quality and preserve network connectivity for robotic sensor systems. In [100], each robotic sensor moves in the gradient ascent direction of its received signal strength (RSS). However, the movement scheduling algorithm developed in [100] considers only a single link. Moreover, to obtain an estimate of RSS gradient, the robotic sensor has to explore the local area, which increases energy consumption in movements. In Chapter 4, we propose a feedback-control-based approach that aims to adaptively maintain the network connectivity of a robotic sensor swarm in the presence of various environment and system dynamics. Our control-theoretic algorithm does not require the energy-consuming exploration in local area.

2.3 Visual Sensing Applications

Several research efforts have explored the integration of cameras with low-power wireless sensing platforms. Cyclops [76] integrates a CMOS imager hosted by a MICA2 mote [98] and conducts object detection using a naive background subtraction method. In [102], a low-end camera module is installed on an AUV for navigation. SensEye [49] incorporates a Cyclops camera into a Stargate platform [93] to detect objects at a 128×128 resolution. In [97], a camera module is installed on an XYZ node [57] for gesture recognition at a resolution of 256×64. These camera-based platforms can only conduct simple image processing tasks at low resolutions due to their limited computation capabilities. Recently, mobile sensing based on smartphones has received increasing research interest due to their rich computation, communication, and storage resources. The study in [115] designs a driving safety alert system that can detect dangerous driving behaviors using both front- and rear-facing cameras of a smartphone. In [34], a barcode-based visual communication scheme is implemented on smartphone. In Chapter 5, we design an aquatic debris surveillance robot that utilizes the built-in camera, inertial sensors, and other resources on smartphone. Different from existing vision-based systems, we need to deal with several unique challenges in aquatic debris monitoring, such as camera shaking and sporadic debris arrivals. Moreover, different from the previous approaches that typically focus on static cameras, we exploit the controllable mobility of robot to increase the likelihood of capturing debris objects as well as reduce the miss rate in covering sporadic debris arrivals.

2.4 Inertial Sensing Applications

Inertial sensing has recently been explored in various mobile sensing applications. In [37], a set of inertial features are extracted from a phone's built-in accelerometer and used to identify user's

transportation mode. In [105], accelerometer and gyroscope readings are employed to model vehicle dynamics and estimate the device's position inside the vehicle. In [26], smartphones collaboratively detect an earthquake using on-board accelerometer. Fitbit [27] helps track the activity pattern and sleep quality of the user using acceleration features. The FOCUS system [43] detects shaken and blurred views during video recording based on measured acceleration. In Chapter 6, we adopt inertial sensing as an energy-efficient alternative to visual sensing in compute-intensive image processing tasks.

Chapter 3

Diffusion Process Profiling

3.1 Introduction

Oil spills and other harmful diffusion processes such as chemical or radiation leaks have disastrous impact on public health and ecosystem sustainability. When such a crisis arises, an immediate requirement is to profile the characteristics of the diffusion process, including the location of source, the amount of discharged substance, and how rapidly it spreads in space and evolves over time.

This chapter develops an iterative approach to profiling the diffusion process, including the location of source, the amount of discharged substance, and how rapidly it spreads in space and evolves over time, using the robotic fish [96] shown in Figure 1.2. There are several main challenges, however, in using robotic fish for aquatic sensing. First, due to the constraints on size and energy, they are typically equipped with low-end sensors whose measurements are subject to significant biases and noises. Therefore, they must efficiently collaborate in data processing to achieve satisfactory accuracy in diffusion profiling. Second, practical aquatic mobile platforms are only capable of relatively low-speed movements. Hence, the movements of sensors must be efficiently scheduled to achieve real-time profiling of the diffusion processes that may evolve rapidly over time. Third, due to the high power consumption of locomotion, the distance that robotic sensors move in a profiling process should be minimized to extend the network lifetime.

This chapter makes the following contributions:

1) We propose a novel accuracy-aware approach for aquatic diffusion profiling based on robotic sensor networks. Our approach leverages the mobility of robotic sensors to iteratively profile the spatiotemporally evolving diffusion process.

2) We derive the analytical profiling accuracy of our approach based on the Cramér-Rao bound (CRB). Then we formulate a movement scheduling problem for aquatic diffusion profiling, in which the profiling accuracy is maximized under the constraints on sensor mobility and energy budget. We develop gradient-ascent-based *greedy* and dynamic-programming-based *radial* movement scheduling algorithms to solve the problem.

3) We implement the profiling and movement scheduling algorithms on TelosB motes and evaluate the system overhead. Moreover, we collect real data traces of GPS localization errors, robotic fish movement, and wireless communication for evaluation for evaluation.

3.2 Preliminaries

3.2.1 Diffusion Process Model

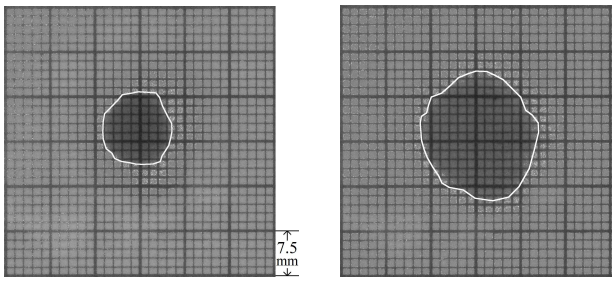
A diffusion process in a static aquatic environment, by which molecules spread from areas of higher concentration to areas of lower concentration, follows Fick's law [60]. In addition to the diffusion, the spread of the discharged substance is also affected by the advection of solvent, e.g., the movement of water caused by the wind. By denoting t as the time elapsed since the discharge of substance and c as the substance concentration, the diffusion-advection model can be described as

$$\frac{\partial c}{\partial t} = D_x \cdot \frac{\partial^2 c}{\partial x^2} + D_y \cdot \frac{\partial^2 c}{\partial y^2} + D_z \cdot \frac{\partial^2 c}{\partial z^2} - u_x \cdot \frac{\partial c}{\partial x} - u_y \cdot \frac{\partial c}{\partial y}, \tag{3.1}$$

where D is the diffusion coefficient, u is the advection speed, and the subscripts of D and u denote the directions (i.e., x-, y-, and z-axis). The diffusion coefficients characterize the speed of diffusion and depend on the species of solvent and discharge substance as well as other environment factors such as temperature. The advection speeds characterize the horizontal solvent movement caused by external forces such as wind and flow. The above Fickian diffusion-advection model has been widely adopted to study the spreading of gaseous substances [103] and buoyant fluid pollutants such as oil slick on the sea [63]. For many buoyant fluid pollutants, the two horizontal diffusion coefficients, i.e., D_x and D_y, are identical, while the vertical diffusion coefficient, i.e., D_z, is insignificant. For example, in a field experiment [24], where diesel oil was discharged into the sea, the estimated D_x is $2{,}000\,\mathrm{cm}^2/\mathrm{s}$ while D_z is only $10\,\mathrm{cm}^2/\mathrm{s}$. Therefore, the vertical diffusion coefficient can be safely ignored and the diffusion can be well characterized by a 2-dimensional process. In this chapter, our study is focused on buoyant fluid pollutants with the diffusion coefficients $D_x = D_y = D$.

Suppose a total of $A\,\mathrm{cm}^3$ of substance is discharged at location (x_s, y_s) and $t = 0$. At time $t > 0$, the original diffusion source is drifted to (x_0, y_0) due to advection, where $x_0 = x_s + u_x t$ and $y_0 = y_s + u_y t$. Hereafter, by *source location* we refer to the source location that has *drifted* from the original position due to advection, unless otherwise specified. Denote $d(x,y)$ (abbreviated to d) as the distance from any location (x,y) to the source location, i.e., $d = \sqrt{(x-x_0)^2 + (y-y_0)^2}$. In the presence of advection, the diffusion is isotropic with respect to the drifted source location [16]. Therefore, the concentration at (x,y) can be denoted as $c(d,t)$. The initial condition for Equation (3.1) is an impulse source, which can be represented by the Dirac delta function, i.e., $c(d,0) = A \cdot \delta(d)$. The closed-form solution to Equation (3.1) is given by [103]:

$$c(d,t) = \alpha \cdot \exp\left(-\beta \cdot d^2\right), \quad d \geq 0, t > 0, \tag{3.2}$$



(a) Elapsed time $t = 3.5$ s (b) Elapsed time $t = 18.1$ s

Figure 3.1: Observations of the diffusion process of Rhodamine-B solution in saline water.

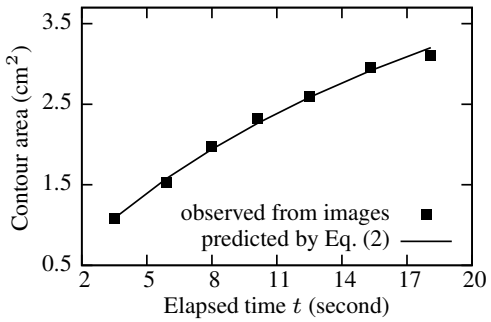


Figure 3.2: Observed and predicted contour area over time.

where $\alpha = A/4\pi Dt$ and $\beta = 1/4Dt$. From Equation (3.2), for a given time instant t, the concentration distribution is described by the Gaussian function that centers at the source location. As time elapses, the concentration distribution becomes flatter. In this chapter, the *diffusion profile* is defined as $\Theta = \{x_0, y_0, \alpha, \beta\}$.

We now validate Equation (3.2) with real lab experiments of Rhodamine-B diffusion. We discharge Rhodamine-B solution in saline water, and periodically capture diffusion process using a digital camera. We assume that the grayscale of a pixel in the captured image linearly increases with the concentration at the corresponding physical location [99]. Therefore, the evolution of diffusion process can be characterized by the expansion of a contour given a certain threshold of grayscale in the captured images. With the measured contour areas along the recorded shooting times, we can estimate D by linear regression. Our methodology of model validation is to compare the contour area observed in images with that predicted by the model in Equation (3.2). The detailed derivations are available in [108, 109]. Figure 3.1 plots the captured images with contours marked in white. Figure 3.2 plots the contour areas observed in images and predicted by Equation (3.2) versus t. We can see that the model in Equation (3.2) well characterizes the diffusion process of Rhodamine-B. Although we only verify the Fickian diffusion model in small scale,

this model has also been validated using data traces of the large-scale spreading of buoyant fluid pollutants, e.g., the oil slick in [63].

3.2.2 Sensor Model

Our approach leverages mobile sensing platforms (e.g., robotic fish [96]) to collaboratively profile an aquatic diffusion process. The nodes form a cluster, and a cluster head is selected to process the measurements from cluster members. The selection of cluster head will be discussed in Section 3.6.2. Moreover, we will extend our approach to address multiple clusters in Section 3.7.2. Many aquatic mobile platforms are battery-powered and hence have limited mobility and energy budget. For example, the movement speed of the robotic fish designed in [96] was about 1.8 to $6\,\mathrm{m/min}$. We assume that the mobile nodes are equipped with pollutant concentration sensors (e.g., the Cyclops-7 series [17]) that can measure the concentrations of crude oil, harmful algae, etc. Lastly, we assume that the sensors are equipped with low-power wireless interfaces (e.g., ZigBee radio) and can communicate with each other on water surface.

The measurements of most sensors are subject to biases and additive random noises from the sensor circuitry and the environment (e.g., wave). Specifically, the reading of sensor i, denoted by z_i, is given by $z_i = c(d_i,t) + b_i + n_i$, where d_i is the distance from sensor i to the diffusion source, b_i and n_i are the bias and random noise for sensor i, respectively. In the presence of constant-speed advection, the source and the sensors will drift with the same speed and therefore they are in the same inertial system. As a result, the concentration at the position of sensor i is given by $c(d_i,t)$. We assume that the noise experienced by sensor i follows the zero-mean normal distribution with variance ς^2, i.e., $n_i \sim \mathcal{N}(0,\varsigma^2)$. We assume that the noises, i.e., $\{n_i|\forall i\}$, are independent across sensors. Such a measurement model has been widely adopted for various chemical sensors [64, 103, 117]. We assume that the biases and noise variance (i.e., $\{b_i|\forall i\}$ and

ς) are known constants. For example, they are often given in the sensor specification provided by the manufacturer. Moreover, they can be measured in offline lab experiments. Specifically, by placing a sensor in the pollutant-free fluid media, the bias and noise variance can be estimated by the sample mean and variance over a number of readings [2].

In this chapter, we adopt a temporal sampling scheme to mitigate the impact of noise. When sensor i measures the concentration, it continuously takes K samples in a short time, and computes the average as its measurement. Hence, the measurement z_i follows the normal distribution, i.e., $z_i \sim \mathcal{N}(c(d_i,t)+b_i,\sigma^2)$, where $\sigma^2 = \varsigma^2/K$.

3.3 Overview of Approach

In this section, we provide an overview of our approach. Our objective is to profile (i.e., estimate Θ) of an aquatic diffusion-advection process using a robotic sensor network. Our approach is designed to meet two key objectives. First, the noisy measurements of sensors are jointly processed to improve the accuracy in profiling the diffusion. Second, sensors can actively move based on current measurements to maximize the profiling accuracy subject to the energy consumption budget. With the estimated profile $\widetilde{\Theta}$, we can learn several important characteristics of the diffusion process of interest.[1] First, we can compute the current concentration contour maps with Equation (3.2). Second, we can estimate the elapsed time since the start of the diffusion and the total amount of discharged substance, with $\widetilde{t} = (4D\widetilde{\beta})^{-1}$ and $\widetilde{A} = \pi\widetilde{\alpha}\widetilde{\beta}^{-1}$. Third, we can estimate the original source location by $\widetilde{x}_s = \widetilde{x}_0 - u_x\widetilde{t}$ and $\widetilde{y}_s = \widetilde{y}_0 - u_y\widetilde{t}$. Moreover, we can predict the evolution of the diffusion in the future, which is often important for emergency management in the cases of harmful substance discharge.

[1] For the clarity of presentation, we denote $\widetilde{x}$ as the estimate of x.

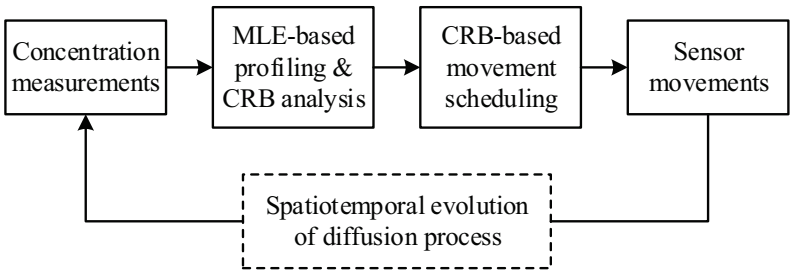


Figure 3.3: The iterative diffusion profiling process.

We assume that the robotic sensors are initially distributed at randomly chosen positions in the deployment region that covers the diffusion source. For example, the sensors can be dropped off from an unmanned aerial vehicle or placed by an aquatic vessel randomly. Note that random and sometimes uniform deployment of sensors around the source location is a good strategy when the characteristics of diffusion process have yet to be determined. This also avoids the massive locomotion energy required to spread sensors for a satisfactory profiling accuracy when the diffusion process evolves. We assume that all sensors know their positions (e.g., through GPS or an in-network localization service) and are time-synchronized. In Section 3.7.3.6, we will evaluate the impact of initial sensor deployment on the profiling accuracy and locomotion energy consumption.

After the initial deployment, sensors begin a diffusion profiling process consisting of multiple *profiling iteration*s. The iterative profiling process is illustrated in Figure 3.3. In a profiling iteration, sensors first simultaneously take concentration measurements and send them to the cluster head. Various existing data collection protocols [53, 113] can be used to collect the measurements. The cluster head then adopts the maximum likelihood estimation (MLE) to estimate Θ from the noisy measurements of all sensors. With the estimated diffusion profile, the cluster head schedules sensor movements such that the expected profiling accuracy in the next profiling iteration is maximized, subject to the limited sensor mobility and energy budget. Finally, the movement schedule including moving orientations and distances is sent to sensors to direct their movements.

Our accuracy-aware diffusion profiling approach features the following novelties. First, it starts with little prior knowledge about the diffusion and progressively learns the profile of the diffusion with improved accuracy along the profiling iterations. As sensors resample the concentration in each iteration, such an iterative profiling strategy allows the network to adapt to the dynamics of the diffusion process while reducing energy consumption of robotic sensors. Moreover, although the profiling accuracy in each iteration is affected by errors in sensor localization and movement control, our approach only schedules short-distance movements for sensors in each iteration and updates their positions in the next iteration, which avoids the accumulation of errors in sensor localization and movement control. Second, we analyze the CRB of the MLE-based diffusion profiling algorithm and propose a novel CRB-based profiling accuracy metric, which is used to direct the movement scheduling of robotic sensors. Third, we propose two novel movement scheduling algorithms, which include a gradient-ascent-based *greedy* algorithm and a dynamic-programming-based *radial* algorithm. The *greedy* algorithm only incurs linear complexity, while the *radial* algorithm can find the near-optimal movement schedule with a higher but still polynomial complexity.

3.4 Profiling Algorithm and Accuracy Analysis

In this section, we first present our MLE-based diffusion profiling algorithm, which estimates the diffusion profile Θ in each profiling iteration. We then analyze the theoretical profiling accuracy based on CRB. The closed-form relationship between the profiling accuracy and the sensors' positions will guide the design of our accuracy-aware sensor movement scheduling algorithms.

3.4.1 MLE-Based Diffusion Profiling Algorithm

Suppose that a total of N aquatic sensors are deployed in the region of interest. To simplify the analysis, we define the bias-removed and normalized observation vector $\mathbf{z}$ as

$$\mathbf{z} = [\frac{z_1 - b_1}{\sigma}, \frac{z_2 - b_2}{\sigma}, \ldots, \frac{z_N - b_N}{\sigma}]^\top.$$

As the sensor biases and noise variance are known, $\mathbf{z}$ can be easily calculated by the cluster head based on the noisy sensor measurements. By denoting $\mathbf{H} = [\sigma^{-1}\mathrm{e}^{-\beta d_1^2}, \ldots, \sigma^{-1}\mathrm{e}^{-\beta d_N^2}]^\top$, $\mathbf{z}$ follows the N-dimensional normal distribution, i.e., $\mathbf{z} \sim \mathcal{N}(\alpha\mathbf{H}, \mathbf{I})$, where $\mathbf{I}$ is an $N \times N$ identity matrix. The log-likelihood of an observation $\mathbf{z}$ given Θ is expressed by [22]:

$$\mathscr{L}(\mathbf{z}|\Theta) = -(\mathbf{z} - \alpha\mathbf{H})^\top(\mathbf{z} - \alpha\mathbf{H}). \tag{3.3}$$

MLE maximizes the log-likelihood given by Equation (3.3). Formally, $\widetilde{\Theta}(\mathbf{z}) = \arg\max_\Theta \mathscr{L}(\mathbf{z}|\Theta)$. This unconstrained optimization problem can be solved by various numerical methods, e.g., the Nelder-Mead algorithm [65].

3.4.2 Cramér-Rao Bound for Diffusion Profiling

CRB provides a theoretical lower bound on the variance of parameter estimators [22], and has been widely adopted to guide the design of estimation algorithms [64], [117]. This section derives the CRB of $\widetilde{\Theta}(\mathbf{z})$. CRB is given by the inverse of the Fisher information matrix (FIM) [22]. For the diffusion profiling, the FIM is defined by $\mathbf{J} = -\mathbb{E}\left[\frac{\partial}{\partial\Theta}\left(\frac{\partial}{\partial\Theta}\mathscr{L}(\mathbf{z}|\Theta)\right)\right] = \alpha^2\frac{\partial\mathbf{H}^\top}{\partial\Theta}\frac{\partial\mathbf{H}}{\partial\Theta}$, where the expectation $\mathbb{E}[\cdot]$ is taken over all possible $\mathbf{z}$. The k^{th} diagonal element of the inverse of $\mathbf{J}$ (denoted by $\mathbf{J}^{-1}_{k,k}$) provides the lower bound on the variance of the k^{th} element of $\widetilde{\Theta}$ (denoted by $\widetilde{\Theta}_k$) [22].

Formally, $\mathrm{Var}(\widetilde{\Theta}_k) \geq \mathbf{J}^{-1}_{k,k}$. The number $\mathbf{J}^{-1}_{k,k}$ is the CRB corresponding to Θ_k, which is denoted as $\mathrm{CRB}(\Theta_k)$ in this chapter. Although the CRB can be easily computed via numerical methods, in order to guide the movements of sensors, we will derive the closed-form CRB.

Even though $\mathbf{J}$ is just a 4×4 matrix, deriving $\mathbf{J}^{-1}$ is challenging, because the N-dimensional joint distribution function in Equation (3.3) leads to high inter-node dependence. To simplify the discussion, we set up a Cartesian coordinate system with the origin at the source location and let (x_i, y_i) denote the coordinates of sensor i. Note that the coordinates of the diffusion source and sensor i in the global coordinate system are (x_0, y_0) and $(x_0 + x_i, y_0 + y_i)$, respectively. We apply matrix calculus to derive the closed-form $\mathbf{J}$ and then derive $\mathbf{J}^{-1}$ by block matrix manipulations. Due to space limitation, the details of the derivations are omitted here and can be found in [108, 109]. To facilitate the representation of CRB, we first define several notations, i.e., $\widehat{x_i}$, $\widehat{y_i}$, $\mathbf{L}_{X_1}$, $\mathbf{L}_{X_2}$, $\mathbf{L}_{Y_1}$, and $\mathbf{L}_{Y_2}$. First, $\widehat{x_i}$ is given by

$$\widehat{x_i} = \frac{\sum_{j=1}^{N} x_j (d_j^2 - d_i^2) \mathrm{e}^{-2\beta d_j^2}}{\sqrt{\sum_{m=1}^{N} \sum_{n=1}^{N} (d_m^2 - d_n^2)^2 \mathrm{e}^{-2\beta (d_m^2 + d_n^2)}}}. \tag{3.4}$$

By replacing x_j in Equation (3.4) with y_j, we can define $\widehat{y_i}$ in a similar manner. Moreover, $\mathbf{L}_{X_1}$, $\mathbf{L}_{Y_1}$ are $1 \times N$ vectors, and $\mathbf{L}_{X_2}$, $\mathbf{L}_{Y_2}$ are $N \times 1$ vectors. The i^{th} elements of them are defined as $\mathbf{L}_{X_1}(i) = \sigma^{-1} \mathrm{e}^{-\beta d_i^2} (x_i + \widehat{x_i})$, $\mathbf{L}_{Y_1}(i) = \sigma^{-1} \mathrm{e}^{-\beta d_i^2} (y_i + \widehat{y_i})$, $\mathbf{L}_{X_2}(i) = \sigma^{-1} \mathrm{e}^{-\beta d_i^2} (x_i - \widehat{x_i})$, and $\mathbf{L}_{Y_2}(i) = \sigma^{-1} \mathrm{e}^{-\beta d_i^2} (y_i - \widehat{y_i})$. Based on the above notations, the CRBs for the estimates of x_0 and y_0 are given

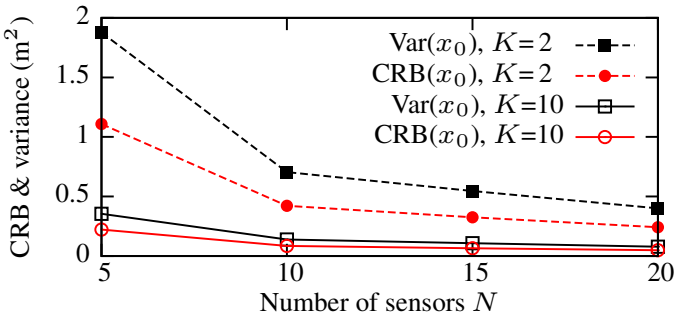


Figure 3.4: Variance of MLE-based profiling algorithm converges to CRB with more sensors. K is the number of samples for computing a measurement.

by

$$\mathrm{CRB}(x_0) = \mathbf{F}_{1,1}^{-1} = \frac{(4\alpha^2\beta^2)^{-1}}{\mathbf{L}_{X_1}\mathbf{L}_{X_2} - \frac{\left(\mathbf{L}_{X_1}\mathbf{L}_{Y_2}+\mathbf{L}_{X_2}\mathbf{L}_{Y_1}\right)^2}{4\mathbf{L}_{Y_1}\mathbf{L}_{Y_2}}}, \tag{3.5}$$

$$\mathrm{CRB}(y_0) = \mathbf{F}_{2,2}^{-1} = \frac{(4\alpha^2\beta^2)^{-1}}{\mathbf{L}_{Y_1}\mathbf{L}_{Y_2} - \frac{\left(\mathbf{L}_{X_1}\mathbf{L}_{Y_2}+\mathbf{L}_{X_2}\mathbf{L}_{Y_1}\right)^2}{4\mathbf{L}_{X_1}\mathbf{L}_{X_2}}}. \tag{3.6}$$

We now discuss how well the CRBs can characterize the MLE-based profiling algorithm discussed in Section 3.4.1. As shown in [38], the variance of a parameter estimated by MLE converges to its CRB when the number of sensors N approaches infinity. We now evaluate the convergence under realistic settings of N. Figure 3.4 shows $\mathrm{Var}(x_0)$ and $\mathrm{CRB}(x_0)$ versus N, where the sensors are randomly deployed in a $200 \times 200\,\mathrm{m}^2$ region. We can see that $\mathrm{Var}(x_0)$ approaches $\mathrm{CRB}(x_0)$ as N increases. Moreover, we evaluate the impact of sensor sampling scheme on the convergence. Note that sensor's signal-to-noise ratio (SNR) increases with the number of samples K. We can see that MLE shows better convergence for larger K. For example, if $K = 10$ and $N \geq 10$, the difference between $\mathrm{Var}(x_0)$ and $\mathrm{CRB}(x_0)$ is insignificant. These results show that the CRB can well characterize the accuracy of the MLE-based profiling algorithm under realistic setting of network size.

3.4.3 Profiling Accuracy Metric

In this section, we propose a novel diffusion profiling accuracy metric based on the CRBs derived in Section 3.4.2, which will be used to guide the movements of sensors in Section 3.5. Several previous works [90] adopt the determinant of the FIM as the accuracy metric, which jointly considers all the parameters. Such a metric requires the parameters to be properly normalized to avoid biases. However, normalizing the parameters with different physical meanings is highly problem-dependent. Moreover, as the closed-form determinant of the FIM is extremely complicated, the resulted sensor movement scheduling has to rely on the numerical methods with high computational complexities [90], which is not suitable for robotic sensors with limited resources. In this chapter, we propose a new profiling accuracy metric, denoted by ω, which is defined according to the sum of reciprocals of $\mathrm{CRB}(x_0)$ and $\mathrm{CRB}(y_0)$. Formally,

$$\omega = \frac{\frac{1}{\mathrm{CRB}(x_0)} + \frac{1}{\mathrm{CRB}(y_0)}}{4\alpha^2\beta^2} = (1-\varepsilon)\left(\mathbf{L}_{X_1}\mathbf{L}_{X_2} + \mathbf{L}_{Y_1}\mathbf{L}_{Y_2}\right), \tag{3.7}$$

where $\varepsilon = \frac{\left(\mathbf{L}_{X_1}\mathbf{L}_{Y_2} + \mathbf{L}_{X_2}\mathbf{L}_{Y_1}\right)^2}{4\mathbf{L}_{X_1}\mathbf{L}_{X_2}\mathbf{L}_{Y_1}\mathbf{L}_{Y_2}}$. By adopting reciprocals, the accuracy analysis can be greatly simplified. Note that as α and β are unknown but fixed in a particular profiling iteration, $4\alpha^2\beta^2$ in the denominator of Equation (3.7) is a scaling factor. Therefore, optimizing $1/\mathrm{CRB}(x_0) + 1/\mathrm{CRB}(y_0)$ is equivalent to optimizing ω. As discussed in Section 3.4.1, we adopt the MLE to estimate Θ. The variance of the MLE result converges to CRB and hence a larger ω indicates more accurate estimation of x_0 and y_0. With the metric ω, the movements of sensors will be directed according to the accuracy of localizing the diffusion source. In the rest of this chapter, the term *profiling accuracy* refers to the metric ω defined in Equation (3.7). Note that our approach can also be applied to focus on the profiling accuracy of the elapsed time t and discharged substance amount A, by applying the same matrix manipulations to obtain $\mathrm{CRB}(\alpha)$ and $\mathrm{CRB}(\beta)$.

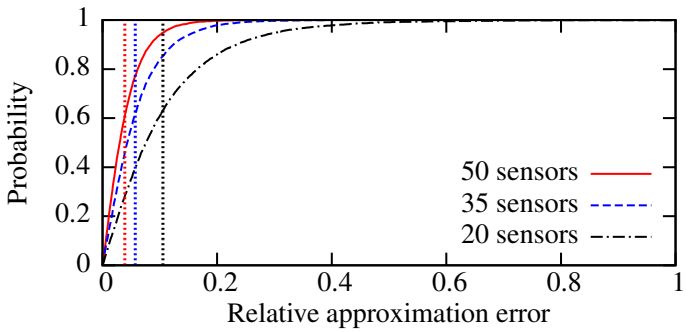


Figure 3.5: CDF and average of relative approximation error for ω.

3.4.4 Approximated Profiling Accuracy

The profiling accuracy metric ω proposed in Section 3.4.3 can characterize the profiling performance. However, its expression given by Equation (3.7) is still too complex to find efficient movement scheduling algorithms. Hence, we derive the approximation to Equation (3.7). If sensors are randomly distributed around the diffusion source, ε is close to zero. By assuming a random sensor distribution and setting $\varepsilon = 0$, the ω can be approximated as:

$$\omega \approx \sum_{i=1}^{N} \omega_i, \quad \omega_i = \sigma^{-2} \mathrm{e}^{-2\beta d_i^2} \left(d_i^2 - \min_{j \in [1,N], j \neq i} d_j^2 \right), \tag{3.8}$$

where ω_i can be regarded as the *contribution* of sensor i to the overall profiling accuracy. As ω_i depends on d_i and the minimum distance to the source from other sensors, Equation (3.8) highly reduces the inter-node dependence compared with Equation (3.7). Note that the approximation error (i.e., $\omega - \sum_{i=1}^{N} \omega_i$) can be easily calculated by the cluster head with the movement schedule.

We now evaluate the above approximation by Monte Carlo method. We define the *relative approximation error* as $|\omega - \sum_{i=1}^{N} \omega_i| / \omega$. In the Monte Carlo simulations, we generate a large number of deployments over a disc region with radius of 100 m. In each deployment, sensors are randomly distributed and we then calculate the relative approximation error. Figure 3.5 shows the cumulative distribution function (CDF) and the average of the relative approximation error given

different number of sensors. We can see that the average relative approximation error is only 10% when 20 sensors are deployed. This approximation assumes that the deployment region is centered at the diffusion source. In Section 3.7.3.5, we will evaluate the case where the deployment region is biased from the diffusion source.

3.5 Diffusion Process Profiling using Robotic Sensors

In this section, we formally formulate the movement scheduling problem. Because of the limited mobility and energy budget of aquatic mobile sensors, the sensor movements must be efficiently scheduled in order to achieve the maximum profiling accuracy. As the power consumption of sensing, computation and radio transmission is significantly less than that of locomotion [96], in this chapter we only consider the locomotion energy. Moreover, as the locomotion energy is approximately proportional to the moving distance [7], we will use moving distance to quantify the locomotion energy consumption. To simplify the motion control of sensors, we assume that a sensor moves straight in each profiling iteration and the moving distance is always multiple of l meters, where l is referred to as *step*. We note that this model is motivated by the locomotion and computation limitation typically seen for aquatic sensing platforms. First, the locomotion of robotic fish is typically driven by closed-loop motion control algorithms, resulting in constant course-correction during movement. Second, each profiling iteration has short time duration. As a result, the assumption of sensor's straight movement in an iteration does not introduce significant errors in the movement scheduling. As the estimation and movement are performed in an iterative manner, we will focus on the movement scheduling in one profiling iteration. Denote $m_i \in \mathbb{Z}^+$ and $\phi_i \in [0, 2\pi)$ as the number of steps and movement orientation of sensor i in a profiling iteration, respectively. Our objective is to maximize the expected profiling accuracy after sensor

movements, subject to the constraints on total energy budget and sensor's individual energy budget. The movement scheduling problem for diffusion profiling is formally formulated as follows. Suppose that a total of M steps can be allocated to sensors and sensor i can move at most L_i meters in a profiling iteration. Find the allocation of steps and movement orientations for all N sensors, i.e., $\{m_i, \phi_i | i \in [1, N]\}$, such that the profiling accuracy ω (defined by Equation (3.7)) after sensor movements is maximized, subject to:

$$\sum_{i=1}^{N} m_i \leq M, \tag{3.9}$$

$$m_i \cdot l \leq L_i, \quad \forall i. \tag{3.10}$$

Equation (3.9) upper-bounds the total locomotion energy in a profiling iteration. Equation (3.10) can be used to constrain the energy consumption of individual sensors. For example, L_i can be specified according to the sensor's residual energy. Moreover, L_i can also be specified to ensure the delay of a profiling iteration. If sensors move at a constant speed of v m/s and a profiling iteration is required to be completed within τ seconds to achieve the desired temporal resolution of profiling, L_i can be set to $L_i = v \cdot \tau$. As discussed in Section 3.3, the cluster head adopts MLE to estimate Θ, and then schedules the movements of sensors such that the expected ω in the next profiling iteration is maximized, subject to the constraints in Equations (3.9) and (3.10). An exhaustive search to the above problem would yield an exponential complexity with respect to N, which is $O((\frac{2\pi}{\phi_0} \cdot \frac{L_i}{l})^N)$ where ϕ_0 is the granularity in searching for the movement orientation. Such a complexity is prohibitively high as the problem needs to be solved in each profiling iteration by the cluster head. In the next section, we will propose an efficient *greedy* algorithm and a near-optimal *radial* algorithm that are feasible to mote-class sensor platforms.

3.6 Sensor Movement Scheduling Algorithms

In this section, we propose an efficient *greedy* movement scheduling algorithm based on gradient ascent and a near-optimal *radial* algorithm based on dynamic programming to solve the problem formulated in Section 3.5. In both scheduling algorithms, sensors move simultaneously according to a movement schedule. In comparison with the strategy where sensors move sequentially, our scheme adapts to the spatiotemporally evolving process, improving the temporal resolution of profiling.

3.6.1 Greedy Movement Scheduling

Gradient ascent is a widely adopted approach to find a local maximum of a utility function. In this chapter, we propose a *greedy* movement scheduling algorithm based on the gradient ascent approach. We first discuss how to determine the movement orientations for the sensors. Since the profiling accuracy ω given by Equation (3.7) is a function of all sensors' positions, we can compute the gradient of ω with respect to the position of sensor i (denoted by $\nabla_i\omega$), which is formally given by $\nabla_i\omega = \left[\frac{\partial\omega}{\partial x_i}, \frac{\partial\omega}{\partial y_i}\right]^\top$. When all sensors except sensor i remain stationary, the metric ω will increase the fastest if sensor i moves in the orientation given by $\nabla_i\omega$. Therefore, in the greedy movement scheduling algorithm, we let $\phi_i = \angle(\nabla_i\omega)$. Note that sensors will move simultaneously when the movement schedule is executed. We now discuss how to allocate the movement steps. The magnitude of $\nabla_i\omega$, denoted by $\|\nabla_i\omega\|$, quantifies the steepness of the metric ω when sensor i moves in the orientation $\angle(\nabla_i\omega)$ while other sensors remain stationary. Therefore, in the *greedy* algorithm, we propose to proportionally allocate the movement steps according to sensor's gradient magnitude. Specifically, m_i is given by $m_i = \min\left\{\left\lfloor \frac{\|\nabla_i\omega\|}{\sum_{i=1}^{N}\|\nabla_i\omega\|}\cdot M\right\rfloor, \left\lfloor\frac{L_i}{l}\right\rfloor\right\}$. Note that the $\left\lfloor\frac{L_i}{l}\right\rfloor$ in the *min* operator satisfies the constraint Equation (3.10). This *greedy* algorithm has

linear complexity, i.e., $O(N)$, which is preferable for the cluster head with limited computational resource.

3.6.2 Radial Movement Scheduling

In this section, we propose a new movement scheduling algorithm based on the approximations discussed in Section 3.4.4. From Equation (3.8), the contribution of sensor i, ω_i, depends on the minimum distance between the cluster head and other sensors. Because of such inter-node dependence, it is difficult to derive the optimal distance for each sensor that maximizes the overall profiling accuracy ω. It can be shown that the problem involves non-linear and non-convex constrained optimization. Several stochastic search algorithms, such as simulated annealing, can find near-optimal solutions. However, these algorithms often have prohibitively high complexities. In our algorithm, we fix the sensor closest to the estimated source location and only schedule the movements of other sensors in each profiling iteration. As the sensor closest to the source receives the highest SNR, moving other sensors will likely yield more performance gain. Moreover, this sensor can serve as the cluster head that receives measurements from other sensors and computes the movement schedule. It is hence desirable to keep it stationary due to its higher energy consumption in computation and communication. We note that the sensor closest to the source may be different in each iteration after sensor movements, resulting in rotation of cluster head among sensors. By fixing the sensor closest to the source, the distance d_i that maximizes the expected ω_i, denoted by d_i^*, can be directly calculated by

$$d_i^* = \sqrt{\frac{1}{2\beta} + \min_{j\in[1,N]} d_j^2}, \quad \forall i \neq \arg\min_{j\in[1,N]} d_j. \tag{3.11}$$

Note that as β is a time-dependent variable, d_i^* also changes with time and hence should be updated in each profiling iteration. Equation (3.11) allows us to easily determine the movement orientation of sensor i. Specifically, if $d_i > d_i^*$, sensor i will move toward the estimated source location; otherwise, sensor i will move in the opposite direction. Formally, by defining $\delta = \text{sgn}(d_i^* - d_i)$, we can express the movement orientation of sensor i as $\phi_i = \angle([\delta \cdot x_i, \delta \cdot y_i]^\top)$.

We now discuss how to allocate the movement steps. In the rest of this section, when we refer to sensor i, we assume sensor i is not the closest to the estimated source location. After sensor i moves m_i steps in the orientation of ϕ_i, its contribution to the overall profiling accuracy is

$$\omega_i(m_i) = \frac{(d_i + \delta \cdot m_i \cdot l)^2 - \min_{j\in[1,N]} d_j^2}{\sigma^2 \cdot \mathrm{e}^{2\beta(d_i+\delta\cdot m_i\cdot l)^2}}, \tag{3.12}$$

where $\min_{j\in[1,N]} d_j^2$ in Equation (3.12) is a constant for sensor i, and β can be predicted based on its current estimate to capture the temporal evolution of the diffusion, i.e., $\beta = (1/\widetilde{\beta} + 4 \cdot D \cdot \tau)^{-1}$. Given the *radial* movement orientations described earlier, the formulated problem is equivalent to maximizing $\sum_i \omega_i(m_i)$ subject to the constraints Equations (3.9) and (3.10), which can be solved by a dynamic programming algorithm as follows.

We number the sensors by $1, 2, \ldots, N-1$, excluding the sensor closest to the estimated source location. Let $\Omega(i,m)$ be the maximum ω when the first i sensors are allocated with m steps. The dynamic programming recursion that computes $\Omega(i,m)$ can be expressed as

$$\Omega(i,m) = \max_{0\le m_i \le \lfloor L_i/l \rfloor} \{\Omega(i-1, m-m_i) + \omega_i(m_i)\}.$$

The initial condition of the above recursion is $\Omega(0,m) = 0$ for $m \in [0,M]$. According to the above equation, at the i^{th} iteration of the recursion, the optimal value of $\Omega(i,m)$ is computed as the

maximum value of $\lfloor L_i/l \rfloor$ cases which have been computed in previous iterations of the recursion. Specifically, for the case where sensor i moves m_i steps, the maximum profiling accuracy ω of the first i sensors allocated with m steps can be computed as $\Omega(i-1, m-m_i) + \omega_i(m_i)$, where $\Omega(i-1, m-m_i)$ is the maximum ω of the first $i-1$ sensors allocated with $m-m_i$ steps. The maximum overall profiling accuracy is given by $\omega^* = \max_{m\in[1,M]} \Omega(N-1, m)$.

We now describe how to construct the movement schedule using the above dynamic programming recursion. The movement schedule of sensor i is represented by a pair (i, m_i). For each $\Omega(i,m)$, we define a movement schedule $S(i,m)$ initialized to be an empty set. The set $S(i,m)$ is filled incrementally in each iteration when $\Omega(i,m)$ is computed. Specifically, in the i^{th} iteration of the recursion, if $\Omega(i-1, m-m_x) + \omega_i(m_x)$ gives the maximum value among all cases, we add a movement schedule (i, m_x) to $S(i,m)$. Formally, $S(i,m) = S(i-1, m-m_x) \cup \{(i, m_x)\}$, where $m_x = \arg\max_{0\le m_i \le \lfloor v\tau/l \rfloor} \{\Omega(i-1, m-m_i) + \omega_i(m_i)\}$. The dynamic programming complexity is $O\left((N-1)M^2\right)$, where N and M are the number of sensors and allocatable movement steps in a profiling iteration, respectively.

3.7 Performance Evaluation

3.7.1 Evaluation Methodology

We evaluate our approach through a combination of hardware experiments and extensive trace-driven simulations. Specifically, we implement the MLE, *greedy* and *radial* algorithms on TelosB mote platform and evaluate their computation overhead. The results are presented in Section 3.7.2, and provide insight into the feasibility of adopting the proposed profiling algorithms on mote-class robotic sensing platforms. Moreover, we evaluate the proposed profiling algorithms in extensive simulations based on real data traces and present the results in Section 3.7.3.

3.7.2 Overhead on Sensor Hardware

We have implemented the MLE and two movement scheduling algorithms in TinyOS 2.1.1 on TelosB platform [98] equipped with an 8 MHz processor. Note that these algorithms are executed on the cluster head. We ported the C implementation of the Nelder-Mead algorithm [65] in GNU Scientific Library (GSL) [32] to TinyOS to solve the optimization problem in MLE (see Section 3.4.1). The porting is non-trivial because dynamic memory allocation and function pointer are extensively used in GSL while these features are not available in TinyOS. Our implementation of MLE requires 19 KB ROM and 1 KB RAM. When 10 sensors are to be scheduled, the two movement scheduling algorithms require 1 KB and 8.8 KB RAM, respectively. Figure 3.6 plots the average execution time of the MLE, *greedy* and *radial* algorithms versus the number of sensors. We note that the complexity of MLE is $O(N)$. For both movement scheduling algorithms, the execution time linearly increases with N, which is consistent with our complexity analysis. The *radial* algorithm takes about 100 seconds to compute the movement schedule in a profiling iteration when $N = 20$. This overhead is reasonable compared with the movement delay of low-speed mobile sensors. The *greedy* algorithm is significantly faster, and hence provides an efficient solution when the timeliness is more important than profiling accuracy. For the *radial* algorithm, 30% execution time is spent on computing a look-up table consisting of each sensor i's contribution, i.e., $\omega_i(m_i)$ in Equation (3.12), given all possible values of m_i. There are several ways to further reduce the computation overhead. First, our current implementation employs extensive floating-point computation. Our previous experience shows that fixed-point arithmetic is significantly more efficient on TelosB motes. Moreover, we can also adopt more powerful sensing platforms as cluster head in the network. For example, the projected execution time on Imote2 [98] equipped with a 416 MHz processor is within 2 seconds for computing the movement schedule for 20 sensors.

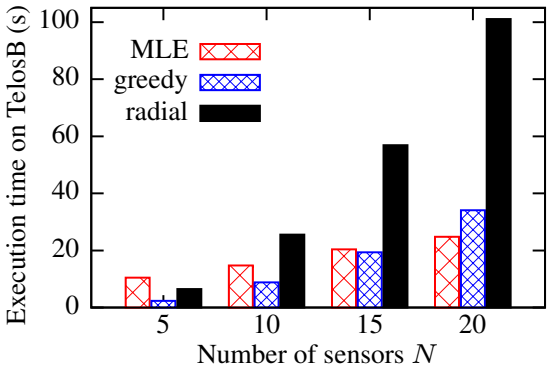


Figure 3.6: Execution time on TelosB versus number of sensors N.

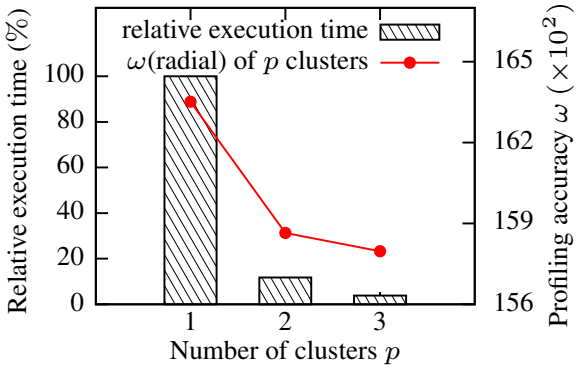


Figure 3.7: Execution time and profiling accuracy versus number of clusters p.

We note that the complexity of *radial* algorithm is $O(N^3)$, which can jeopardize the timeliness of periodical profiling when more sensors are deployed. To reduce the computation delay, in this chapter we adopt a simple clustering method by randomly assigning N nodes to p clusters. The average of the profiling results of all clusters is yielded as the final result. The complexity for a cluster reduces to $O(N^3/p^3)$. Figure 3.7 plots the execution time and profiling accuracy of the *radial* approach when the network is divided into p clusters. The left y-axis is the ratio of execution time for p clusters with respect to the case of a single cluster. We can see that both the execution time and profiling accuracy decrease with p. This is because simply averaging results from all clusters does not fully account for the inter-cluster dependence in the accuracy of dynamic programming. Nevertheless, the *radial* algorithm of 2 and 3 clusters still outperforms the *greedy* algorithm of a single cluster in profiling accuracy.

3.7.3 Trace-Driven Simulations

3.7.3.1 Trace Collection

We collect three sets of data traces, which include GPS localization errors, robotic fish movement, and on-water ZigBee wireless communication. First, the data traces of GPS error are collected

using two Linx GPS modules [55] in outdoor open space. We extract the GPS error by comparing the distance measured by GPS modules with the groundtruth distance. The average GPS error is 2.29 meters. Second, the data traces of movement control are collected with a robotic fish developed in our lab [96] (see Figure 1.2). The movement of robotic fish is driven by a servo motor that is controlled by continuous pulse-width modulation waves. By setting the fish tail beating amplitude and frequency to 23° and 0.9 Hz, the movement speed is $2.5\,\mathrm{m/min}$. We then have the fish swim along a fixed direction in an experimental water tank, and derive the real speed by dividing the moving distance by elapsed time. Third, the data traces of ZigBee communication are collected with two IRIS motes [98] by measuring the packet reception rate (PRR) of a single link. The details of PRR trace collection are omitted here and can be found in Section 4.3.1.

3.7.3.2 Simulation Settings and Methodologies

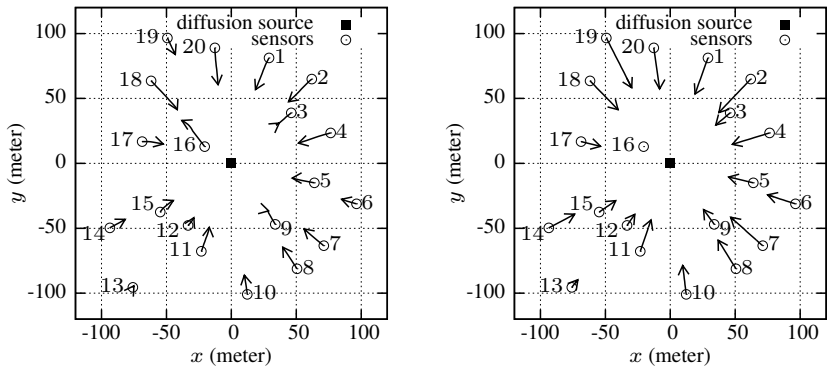
We conduct extensive simulations based on collected data traces to evaluate the effectiveness of our approach. The simulation programs are written in Matlab. As discussed in Section 3.2.2, the effect of constant-speed advection is canceled because the sensors and source location are in the same inertial system. Therefore, we only simulate the diffusion process without advection. The diffusion source is at the origin of the coordinate system, i.e., $x_0 = y_0 = 0$. The sensors are randomly deployed in the square region of $200 \times 200\,\mathrm{m}^2$ centered at the origin. The reading of a sensor is set to be the sum of the concentration calculated from Equation (3.2), the bias b_i, and a random number sampled from the normal distribution $\mathcal{N}(0, \varsigma^2)$. Without loss of generality, we assume that the bias b_i is zero in the simulations. As discussed in Section 3.2.2, in each profiling iteration, a sensor samples K readings and outputs the average of them as the measurement. The amount of discharged substance is set to be $A = 0.7 \times 10^6\,\mathrm{cm}^3$ (i.e., $0.7\,\mathrm{m}^3$) unless otherwise specified. The diffusion coefficient is set to be $D = 5{,}000\,\mathrm{cm}^2/\mathrm{s}$. Note that the settings of A and D are comparable to the

real field experiments reported in [24] where 2 to 5 m^3 of diesel oil were discharged into the sea and the estimated diffusion coefficient ranged from 2,000 cm^2/s to 7,000 cm^2/s. The noise standard deviation is set to be $\varsigma = 1\,cm^3/m^2$, i.e., 1 cm^3 discharged substance per unit area.[2] To easily compare various movement scheduling algorithms, we let the first profiling iteration always start at $t = 1800$ s, i.e., half an hour after the discharge. At $t = 1800$ s, the average received SNR is around $10/1$. The rationale of this setting is that moving sensors too early (i.e., at low SNRs) leads to little improvement on profiling accuracy, resulting in waste of energy. In practice, various approaches can be applied to initiate the profiling process, e.g., by comparing the average measurement to a threshold that ensures good SNRs. Other settings include step length $l = 0.5$ m, profiling iteration duration $\tau = 60$ s, sensor movement speed $v = 2.5\,m/min$ and number of sampling $K = 2$, unless otherwise specified.

We now discuss how the real data traces are used in the simulations. We compare the performance of the *greedy* and *radial* algorithms with and without the data traces of movement and localization errors. If the movement/localization traces are used, the movement speed of a robotic sensor in the simulation is set to be the real speed that is randomly selected from the movement data traces, and the position reading sent to the cluster head is corrupted by a localization error that is randomly selected from the GPS error traces. The simulation results with movement/localization traces presented in Section 3.7.3.3 to 3.7.3.6 are labeled with the prefix "trace-driven". In Section 3.7.3.7, to evaluate the communication overhead of our approach, the PRR of a link in the simulation is set to be the distance-based interpolation of real PRR measurements shown in Figure 4.3.

We compare our approach with two additional baseline algorithms in the evaluation. The first baseline (referred to as *SNR-based*) schedules the movements based on the SNRs received by

[2] As we adopt a 2-dimensional model to characterize the diffusion process, the physical unit of concentration is cm^3/m^2. As observed in the field experiments [24], diesel oil can penetrate down to several meters from the water surface. As a result, the equivalent ς that accounts for the depth dimension ranges from $0.1\,cm^3/m^3$ to $1\,cm^3/m^3$. Our setting is consistent with the noise standard derivation of the crude oil sensor Cyclops-7 [17], which is $0.1\,cm^3/m^3$.



(a) *greedy*: final $\omega = 15400$ (b) *radial*: final $\omega = 16400$

Figure 3.8: Movement trajectories of 20 sensors in the first 15 profiling iterations.

sensors. The SNR received by sensor i (denoted by SNR_i) is defined as $c(d_i,t)/\sigma$, where d_i and t can be computed from the estimated profile $\widetilde{\Theta}$. In the *SNR-based* scheduling algorithm, the sensors always move toward the estimated source location to increase the received SNRs. The movement steps are proportionally allocated according to sensors' SNRs. The rationale behind this heuristic is that the accuracy of MLE increases with SNR. The second baseline (referred to as *annealing*) is based on the simulated annealing. For given movement orientations $\{\phi_i|\forall i\}$, it uses the brutal-force search to find the optimal step allocation under the constraints in Equations (3.9) and (3.10). It then employs a simulated annealing algorithm to search for the optimal movement orientations. However, it has exponential complexity with respect to the number of sensors.

3.7.3.3 Sensor Movement Trajectories

We first visually compare the sensor movement trajectories computed by the *greedy* and *radial* movement scheduling algorithms. A total of 20 sensors are deployed. Figure 3.8 shows the movement trajectories of sensors in the first 15 profiling iterations. For a particular sensor, the circle denotes its initial position, the segments represent its movement trajectory of 15 profiling iterations, and the arrow indicates its movement orientation in the 15^{th} iteration. The sensor with no

segments remains stationary during all 15 profiling iterations. We can see that, with the *greedy* algorithm, several sensors (e.g., sensor 15, 16, and 17) have bent trajectories. This is because the movement orientation of each sensor is to maximize the gradient ascent of ω, hence not necessarily to be aligned along the iterations. In the *radial* algorithm, sensors' trajectories are more straight. This is because the movement orientation is along the direction determined by the current sensor position and the estimated source location that is close to the true source location. Moreover, we find that the *radial* algorithm outperforms the *greedy* algorithm in terms of profiling accuracy after the first 15 iterations. In the *greedy* algorithm, the orientation assignment and movement step allocation are based on the gradient derived from the current positions of sensors. Besides, the *greedy* algorithm does not account for the interdependence of sensors in providing the overall profiling accuracy. As a result, its solution may not lead to the maximum ω after the sensor movements and the temporal evolution of diffusion.

3.7.3.4 Profiling Accuracy

In the second set of simulations, we evaluate the accuracy in estimating the diffusion profile Θ. Total 10 sensors are deployed and our evaluation lasts for 15 profiling iterations. Figure 3.9 plots the profiling accuracy ω (defined in Equation (3.7)) based on the estimated diffusion profile $\widetilde{\Theta}$. The curve labeled with "stationary" is the result if all sensors always remain stationary. Nevertheless, we can see that the profiling accuracy improves over time because of the temporal evolution of the diffusion. For both *greedy* and *radial*, the curves with and without simulated movement control and localization errors almost overlap with each other. As our iterative approach has no accumulated error, small movement control and localization errors have little impact on our approach. The *radial* algorithm outperforms the *greedy* and *SNR-based* algorithms by 16% and 50% in terms of ω at the 15^{th} profiling iteration, respectively. And the accuracy performance of the *radial* algorithm

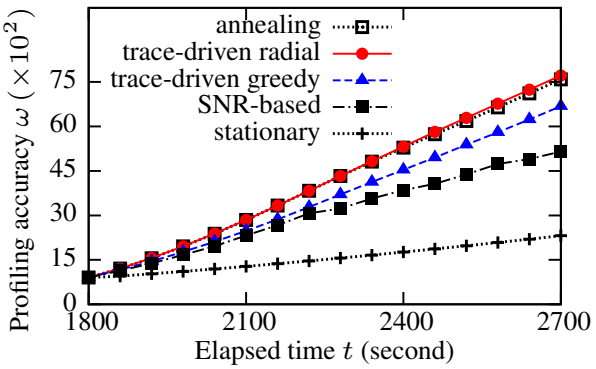


Figure 3.9: Profiling accuracy ω versus elapsed time t.

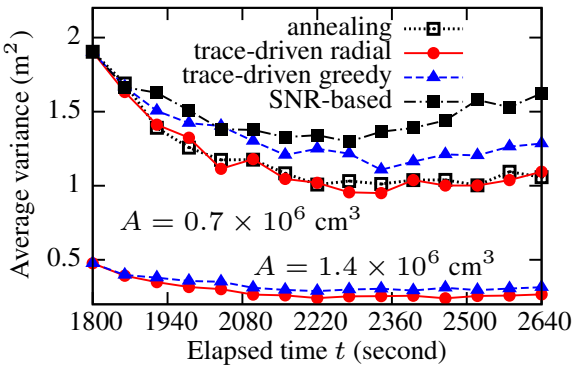


Figure 3.10: Average $\mathrm{Var}(\widetilde{x}_0)$ and $\mathrm{Var}(\widetilde{y}_0)$ versus elapsed time t.

is very close to the *annealing* algorithm that can find the near-optimal solution. However, we note that in each iteration of the *annealing* algorithm, a new look-up table needs to be computed due to changed movement orientations. Hence its execution time highly depends on the number of iterations that can be very large. Therefore, the *annealing* algorithm is infeasible on mote-class platforms.

Figure 3.10 plots the average of $\mathrm{Var}(\widetilde{x}_0)$ and $\mathrm{Var}(\widetilde{y}_0)$ in each profiling iteration under various settings of the discharged substance amount A. In order to evaluate the variances in each profiling iteration, the sensors perform many rounds of MLE, where each round yields a pair of $(\widetilde{x}_0, \widetilde{y}_0)$. The $\mathrm{Var}(\widetilde{x}_0)$ and $\mathrm{Var}(\widetilde{y}_0)$ are calculated from all rounds. From Figure 3.10, we find that the variances may increase (for the *SNR-based* scheduling algorithm) or fluctuate (for other approaches) after several iterations. This is because the variances are time-dependent due to the involving of α and β in $\mathrm{CRB}(x_0)$ and $\mathrm{CRB}(y_0)$. Moreover, we can see that the variances decrease with A. As sensors receive higher SNRs in the case of higher A, our result consists with the intuition that the estimation error decreases with SNR. Compared with the *SNR-based* algorithm, the *radial* algorithm reduces the variance in estimating diffusion source location by 36% for $A = 0.7 \times 10^6\,\mathrm{cm}^3$. Compared with the *greedy* algorithm, the reductions are 12% and 18% for $A = 0.7 \times 10^6$ and $1.4 \times 10^6\,\mathrm{cm}^3$, respectively.

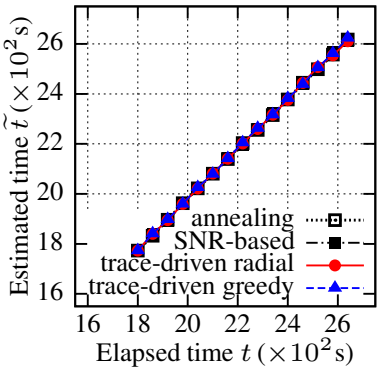


Figure 3.11: Estimated elapsed time $\widetilde{t}$ versus elapsed time t.

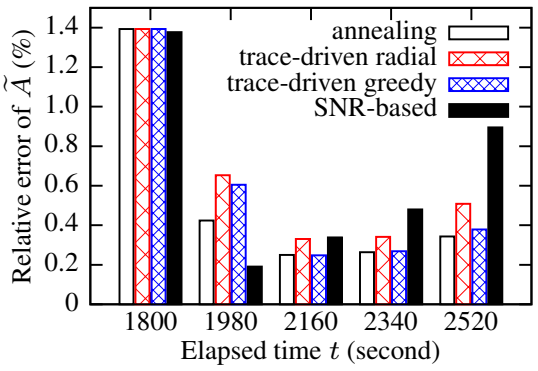


Figure 3.12: Estimated substance amount $\widetilde{A}$ versus elapsed time t.

The MLE algorithm also estimates the initial substance amount A and elapsed time t. Therefore, we additionally evaluate the performance of our movement scheduling algorithms on profiling A and t. Figure 3.11 plots $\widetilde{t}$ versus the groundtruth time. We can see that the elapsed time can be accurately estimated. For example, the relative error of $\widetilde{t}$ for *radial* algorithm is only 1.0%. Figure 3.12 plots the relative error of $\widetilde{A}$, which is calculated as $|(\widetilde{A}-A)/A|$. We can see that both the *radial* and *greedy* algorithms give comparable profiling performance with the *annealing* algorithm. The relative error is less than 1.4%. Moreover, the result shows that the *greedy* algorithm achieves a slightly better profiling performance than the *radial* algorithm. As discussed in Section 3.4.3, the accuracy metric adopted in this chapter is defined to characterize the accuracy of localizing the diffusion source. Therefore, it is possible that *radical* algorithm yields larger error than *greedy* algorithm in profiling A.

3.7.3.5 Sampling Scheme, Source Bias, and Network Density

We characterize the profiling error after 15 iterations by the average of $\mathrm{Var}(\widetilde{x}_0)$ and $\mathrm{Var}(\widetilde{y}_0)$. Except for the evaluations on network density, a total of 10 sensors are deployed. In the temporal sampling scheme presented in Section 3.2.2, a sensor yields the average of K continuous samples as the measurement to reduce noise variance. Figure 3.13 plots the profiling error versus K. We can

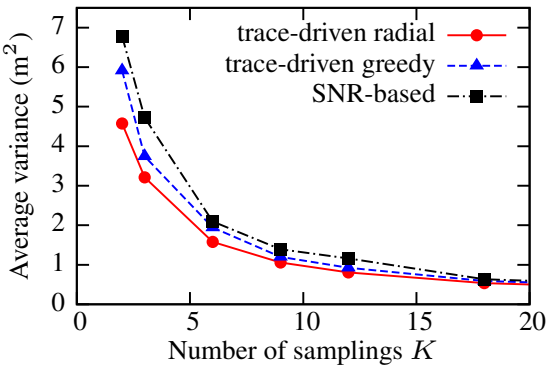


Figure 3.13: Average $\mathrm{Var}(\widetilde{x}_0)$ and $\mathrm{Var}(\widetilde{y}_0)$ versus number of samplings K.

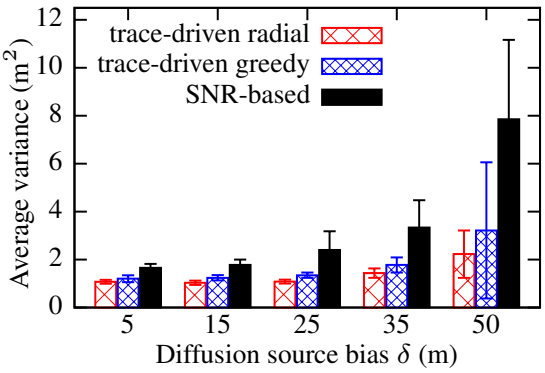


Figure 3.14: Average $\mathrm{Var}(\widetilde{x}_0)$ and $\mathrm{Var}(\widetilde{y}_0)$ versus source location bias δ.

see that the profiling error decreases with K. The relative reductions of profiling error by the *radial* algorithm with respect to the *greedy* and *SNR-based* algorithms are about 18% and 30%, respectively, when K ranges from 2 to 20.

The approximations discussed in Section 3.4.3 assume that the sensors are randomly deployed around the diffusion source. In this set of simulations, we evaluate the impact of source location bias on profiling accuracy. Specifically, the diffusion source appears at $(\delta, 0)$, where δ is referred to as the source location bias. Figure 3.14 plots the profiling error versus δ. To jointly account for the impact of random sensor deployment, for each setting of δ, we deploy a number of networks and show the error bars. We find that the *radial* algorithm is robust to the source location bias. Moreover, we note that the *radial* algorithm is consistently better than other algorithms.

Figure 3.15 plots the profiling error versus the number of sensors. When more sensors are deployed, the profiling error can be reduced. The *radial* algorithm is consistently better than the other algorithms. For all algorithms, the profiling error is reduced by about 40% when the number of sensors increases from 10 to 15. Moreover, the relative reduction of profiling error decreases with the number of sensors.

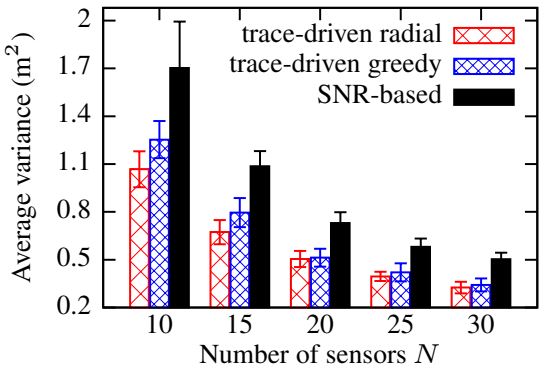


Figure 3.15: Average $\mathrm{Var}(\widetilde{x}_0)$ and $\mathrm{Var}(\widetilde{y}_0)$ versus number of sensors N.

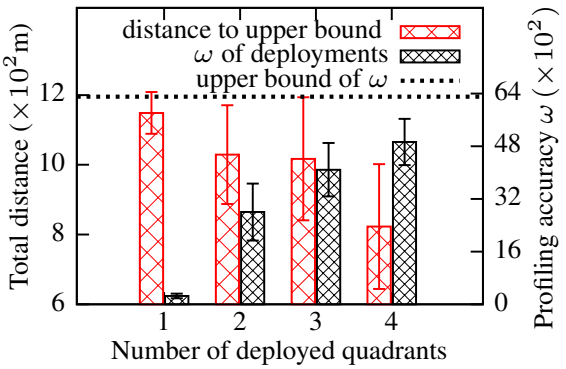


Figure 3.16: Impact of initial deployment on the profiling accuracy.

3.7.3.6 Impact of Sensor Deployment

In this section, we evaluate the impact of initial sensor deployment on the profiling accuracy and energy consumption in locomotion. We fix each d_i and randomly deploy sensors in one, two adjacent, three and four quadrants of the plane originated at the source location, resulting in four sensor deployments. We compute the upper bound of ω, in which sensors' angles with respect to the source location are exhaustively searched to maximize the profiling accuracy. Note that the sensor deployment with maximized profiling accuracy is still an open issue. Figure 3.16 plots the upper bound of ω as well as the profiling accuracy of four sensor deployments. We can see that the profiling accuracy of the four-quadrant deployment is the closest to the upper bound. Figure 3.16 also plots the minimum total distance that the sensors in a deployment have to move to achieve the upper bound ω. We can observe that if sensors are not deployed around the source location, spreading sensors first can significantly improve the profiling accuracy. However, if sensors have limited energy for locomotion, it is more beneficial to deploy sensors around the source to avoid energy-consuming spreading movements.

3.7.3.7 Communication Overhead

In each profiling iteration, the communication overhead is mainly affected by the packet loss caused by the unreliable on-water wireless communication. Hence, we employ the total number of transmissions in collecting all sensor measurements as the evaluation metric. Specifically, we choose the shortest distance path as the routing path from a sensor to the cluster head, where the distance metric of each hop is PRR^{-1}, i.e., the expected number of (re-)transmissions on the hop. The PRR is set to be the distance-based interpolation of real measurements shown in Figure 4.3. When a node transmits packet to the next hop, the packet is delivered with a success probability equal to the PRR. The node re-transmits the packet up to 10 times before it is dropped. In the simulations, 30 sensors are randomly deployed. The packet to the cluster head includes sensor ID, current position and measurement. The packet to each sensor contains the movement schedule that includes movement orientation and distance. Our simulation results show that the number of (re-)transmissions in a profiling iteration has a mean of 158 and a standard deviation of 28. Even if all these transmissions happen sequentially, the delay will be within seconds, because transmitting a TinyOS packet only takes about 10 milliseconds on typical mote-class platforms. This result shows that our approach has low communication overhead under realistic settings.

3.8 Conclusion

This chapter proposes an accuracy-aware profiling approach for aquatic diffusion processes using robotic sensor networks. Our approach features an iterative profiling process where the sensors reposition themselves to progressively improve the profiling accuracy along the iterations. We develop two movement scheduling algorithms, including an efficient *greedy* algorithm and a near-optimal *radial* algorithm. We implement our algorithms on TelosB motes and evaluate them using

real traces of GPS localization errors, robotic fish movement, and ZigBee wireless communication.

Chapter 4

Spatiotemporal Field Reconstruction

4.1 Introduction

This chapter addresses an important problem in aquatic environment monitoring – reconstruction of spatiotemporal field. Many physical and biological phenomena in aquatic environment, including HABs [21], lake surface temperature [114], and plume concentration of chemical substance [20], can be modeled as spatiotemporal aquatic field that usually follows a certain distribution such as the Gaussian process. The reconstructed aquatic field allows one to study fine-grained spatial distribution and temporal evolution of physical and biological phenomena of interest. For example, the reconstructed HAB field is helpful for understanding the development of emerging HABs and guiding authorities to take future preventive actions.

Aquatic sensor networks composed of robotic fish [96] are a typical Cyber-Physical System (CPS) whose efficient operation depends on the tight coupling and coordination between cyber (sensing, communication, and information processing) and physical components (mobility control and environment). Compared with terrestrial sensor networks, there are several unique challenges associated with aquatic sensor networks, including uncontrollable disturbances from the underlying fluid medium (e.g., waves and flows), inherently dynamic profiles of aquatic processes, and significant errors in motion control. Therefore, both sensing and mobility control of robotic fish must account for the spatial variability and temporal evolution of aquatic processes. Moreover, our measurements show that aquatic sensors equipped with ZigBee radio have highly variable link

quality and only about half of the communication range of the terrestrial radio. Such characteristics must be explicitly considered in the design of the network. Finally, the operation of these sensors has to be very energy-efficient due to the limited power supply.

This chapter makes the following contributions:

1) We propose a new approach to the sampling and reconstruction of spatiotemporal aquatic field using a sensor *swarm* composed of inexpensive, low-power, and collaborative robotic sensors. Our approach features a *rendezvous-based* mobility control scheme, where sensors in a swarm gather and sense the environment in a series of carefully chosen rendezvous regions, reducing the overhead of inter-sensor coordination during movement.

2) We design a novel feedback control algorithm that maintains the desirable level of wireless connectivity of a sensor swarm in the presence of significant physical dynamics. Based on a wireless signal propagation model, the control-theoretic algorithm adjusts the radius of rendezvous region adaptively to ensure a bound on the PRR between sensors.

3) We present a new analysis of spatiotemporal field reconstruction accuracy based on mutual information and posterior entropy. Our analytical results are used to guide the selection of rendezvous regions so that the reconstruction accuracy can be maximized subject to the limited sensor mobility.

4.2 Overview of Approach

4.2.1 Background and Challenges

Our objective is to reconstruct an aquatic scalar field that follows the spatiotemporal Gaussian process using a group of robotic sensors. The design of our approach is motivated by the following

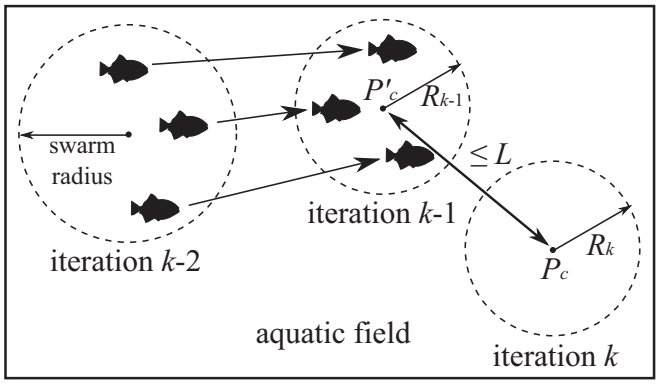


Figure 4.1: Rendezvous-based swarm control scheme. Dashed circles represent the rendezvous circles.

major challenges in reconstructing a spatiotemporal field. First, the physical and biological phenomena of interest often affect large spatial areas. For example, HABs can spread over the water area of a dozen to tens of square kilometers. However, the number of robotic sensors available in practice is often small (e.g., a few dozens). In addition, as the robotic sensors in aquatic environment often have short communication ranges, the area that networked robotic sensor system can sample at any given time is limited. Second, because of the complex environment dynamics (e.g., wave and wind) and the limited motion capabilities of the robotic sensors, accurate movement control of an aquatic sensor system is often challenging. Third, the link quality and network connectivity of robotic sensors are highly dynamic due to physical uncertainties. The resulted data loss can significantly affect the accuracy of field reconstruction.

4.2.2 Approach Overview

A simple approach to reconstructing the field using robotic sensors is to send sensors to regions that evenly divide the whole aquatic field and each sensor only samples its own region. Because the aquatic process typically covers a large area as discussed in Section 4.2.1, under this simple approach, the sensors would not be able to communicate with each other. Therefore, this non-

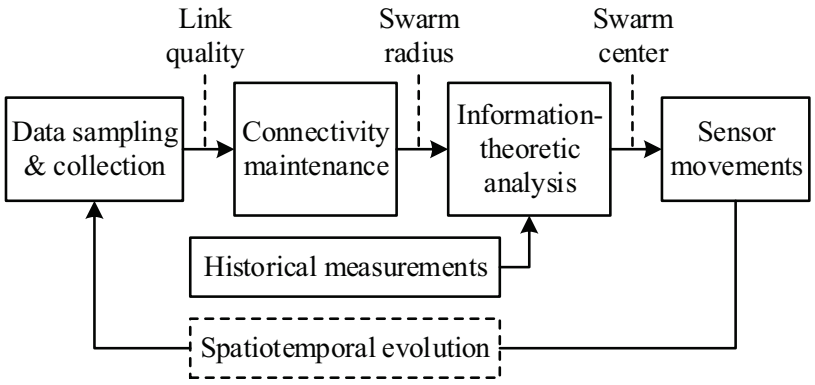


Figure 4.2: The iterative sampling process of a robotic sensor swarm.

collaborative approach has the following two drawbacks. First, each sensor can only reconstruct the field based on its own measurements, and the field reconstruction based on all sensor measurements cannot be performed until sensors complete their sampling and gather at some location. Second, the accuracy of the whole field reconstruction would be significantly undermined if some sensors experience failures.

To address the challenges discussed in Section 4.2.1, we adopt a novel *rendezvous-based swarm* scheme as illustrated in Figure 4.1. We assume that all sensors know their positions and are time-synchronized, e.g., through GPS or in-network localization/synchronization services. The robotic sensor system iteratively samples the aquatic field. As shown in Figure 4.2, in each sampling iteration, robotic sensors move into a rendezvous circle, form a swarm and sample the environment. In the swarm, a sensor serves as the swarm head, which collects the measurements of other sensors via wireless communications as well as schedules the movements of sensors in the next sampling iteration. To simplify the data collection process and reduce the communication overhead, the swarm adopts a single-hop star network topology centered at the swarm head. In our approach, the movement scheduling at the swarm head is executed as follows:

1) The swarm head first assesses the quality of network connectivity based on the received data and then determines the radius of the rendezvous circle (referred to as *swarm radius*) in the

next sampling iteration, such that the network connectivity in the next sampling iteration can achieve a desirable level.

2) Given the projected swarm radius, the swarm head conducts information-theoretic analysis to select the location of the next rendezvous circle, in order to maximize the improvement of the field reconstruction accuracy.

3) The swarm head generates random target positions within the next rendezvous circle and assigns the positions to each sensor to minimize the total movement distance. The target positions are finally sent to the sensors. Under this random target position approach, small motion control errors can be tolerated as long as the final positions of sensors fall within the rendezvous circle. Moreover, as proved in Section 4.4.5, under this approach, there is no crossing between sensors' moving paths and hence the robotic sensors would not collide.

After receiving target position in the next sampling iteration, each sensor straightly moves toward its destination to minimize the energy consumption of locomotion. To initiate the above process, the swarm is initially dropped at a venue within the region affected by the physical/biological process of interest. Note that to balance the energy consumption of sensors, the swarm head role can rotate among all sensors. The communication overhead of our approach is low because sensors coordinate with each other only when they gather in a rendezvous circle. In summary, our swarm scheme allows the robotic sensors to efficiently collaborate in sensing a large dynamic aquatic field and avoid heavy coordination overhead. Therefore, it is practical and energy-efficient for low-power aquatic robotic platforms [96, 116].

Our approach has the following two key novelties:

First, it features control-theoretic connectivity maintenance. Data loss of wireless communication can significantly affect the quality of sensing. A key goal of our system is to ensure that

the swarm head reliably receives the measurements from all sensors. However, this is challenging because the on-water wireless links have highly dynamic quality due to the impact of fluid medium and changing positions of sensors during movement. We develop a control-theoretic algorithm to maintain desirable connectivity of a sensor swarm in the presence of these dynamics by adaptively adjusting the swarm radius. Specifically, the swarm head first estimates the quality of network connectivity based on the average of PRRs of all links. As the swarm average PRR generally decreases with the swarm radius, the swarm head calculates a new swarm radius based on a wireless signal propagation model and the current swarm average PRR, such that the expected connectivity in the next sampling iteration can be maintained at a desirable level. A control problem is formulated to address this procedure and its solution gives an adaptive algorithm for tuning the swarm radius.

Second, it features information-theoretic movement scheduling. Due to limited power supply and high power consumption in locomotion, the sensor swarm must efficiently schedule the movement of sensors to sample the field. Specifically, the swarm head must find the location of the next rendezvous circle subject to energy budget, such that the improvement of the field reconstruction accuracy can be maximized with the newly obtained sensor measurements. In this chapter, we employ information-theoretic analysis to guide the selection of rendezvous circle locations. Moreover, two information metrics (i.e., mutual information and posterior entropy) with different computational complexities can be integrated with our analysis, which hence allow the system designer to choose desirable trade-offs between the system overhead and reconstruction accuracy.

4.3 Swarm Connectivity Maintenance

The wireless connectivity between a robotic sensor and the swarm head is affected by various environment and system dynamics, which include the stochastic fluctuation of the on-water wireless

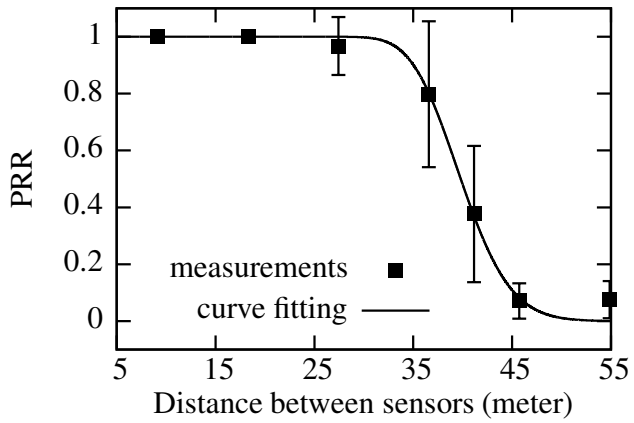


Figure 4.3: PRR measurements versus distance between two sensors. The error bar represents standard deviation.

links, the errors of localization and motion control, and the uncertain distance between moving sensors. In this section, we first study the on-water wireless link dynamics based on real data traces collected on a lake. We then analyze the swarm connectivity. Finally, we formulate the connectivity maintenance as a feedback control problem, which aims to maintain the swarm connectivity at a desired level by adjusting the swarm radius based on the quality of all links measured at runtime.

4.3.1 On-Water Wireless Link Dynamics

We first motivate our approach using PRR traces of on-water ZigBee wireless link. Figure 4.3 plots the PRR measured by two IRIS motes versus distance in an experiment conducted on the wavy water surface of Lake Lansing, Michigan, on a windy day. Specifically, we placed the two IRIS motes about 12 cm above the water surface and measured the PRR versus the distance between the two motes. Each PRR measurement was calculated from 50 packets transmitted within one second. From Figure 4.3, we have the following two important observations. First, the on-water wireless communication has a limited reliable communication range, which is about 35 m for a typical ZigBee radio. According to our experience, the communication range of IRIS mote

on water surface decreases by about 50% compared to that on land. Second, the PRR shows significant variance, especially in the transition range from 25 m to 40 m. It is mostly caused by the radio and environment dynamics [118]. The wireless link in wavy water environment is more dynamic than that in calm water environment, due to the multipath effect and fading. Such highly dynamic communication quality can lead to increased communication cost in the field sampling and even loss of sensors due to disconnected network. Therefore, it is critical to maintain satisfactory connectivity under radio and environment dynamics.

4.3.2 Modeling Swarm Connectivity

As discussed in Section 4.2.2, the sensor swarm forms a network with single-hop star topology in a rendezvous circle. Compared with multi-hop topology, the single-hop topology of sensor swarm incurs significantly lower overhead in communication and network formation/maintenance. Suppose the reliable on-water communication range of a typical ZigBee radio is 35 m. Under the single-hop star topology centered at the swarm head, a sensor swarm can spread over an area of up to 3,800 m^2. We adopt the average PRR of the links between the swarm head and all sensors as the metric of swarm connectivity. This metric quantifies not only the average connectivity of the swarm but also the communication cost in collecting sensor measurements in a sampling iteration. In this section, we first derive the expression for the average PRR given swarm radius, which allows us to adaptively control the swarm connectivity by adjusting the swarm radius. We then verify the closed-form expression using real data traces.

4.3.2.1 Model Derivation

Let P_t (in dBm) denote the power of the wireless signal transmitted by a sensor, and $PL(d_0)$ (in dBm) denote the path loss at reference distance d_0. The signal power at the receiver that is d

meters from the transmitter is $P_r(d) = P_t - PL(d_0) - 10\alpha \log_{10}(d/d_0)$ [78], where α is the path loss exponent that typically ranges from 2.0 to 4.0. We assume that the noise power (denoted by P_n) in dBm follows the zero-mean normal distribution with variance ξ^2 [78]. The signal-to-noise ratio (SNR) at distance d is given by $\text{SNR} = P_r(d) - P_n$. We assume that a packet can be successfully received if the SNR is greater than a threshold denoted by η [45]. Hence, the PRR of a single link can be derived as

$$\text{PRR}(d) = \frac{1}{2} + \frac{1}{2} \cdot \text{erf}(a_1 \log_{10} d + a_2), \tag{4.1}$$

where $a_1 = -5\sqrt{2}$, $a_2 = \frac{P_t - PL(d_0) - \eta}{\sqrt{2}\xi} + 5\sqrt{2}\log_{10} d_0$, and $\text{erf}(\cdot)$ is the error function.

Based on the single-link PRR model given in Equation (4.1), we now derive the average PRR over all sensors that are randomly distributed within the rendezvous circle. Our analysis shows that it is difficult to derive the closed-form formula for the average PRR. We propose an approximate formula as follows. The expectation of the distance between any sensor and the swarm head (denoted by $\mathbb{E}[d]$), which are two random points in the rendezvous circle, is a linear function of the swarm radius (denoted by R), specifically, $\mathbb{E}[d] = 128R/45\pi$. Based on this observation and Equation (4.1), we approximate the average PRR over all sensors (denoted by $\overline{\text{PRR}}(R)$) by

$$\overline{\text{PRR}}(R) \simeq (1-c) + c \cdot \text{erf}(c_1 \log_{10} R + c_2), \tag{4.2}$$

where c_1 ($c_1 < 0$), c_2 ($c_2 > 0$), and c ($0 < c < 0.5$) are three coefficients. Although Equation (4.2) is an approximate model, the feedback-based connectivity maintenance algorithm can tolerate minor inaccuracy in system modeling.

4.3.2.2 Model Validation

We now use the collected PRR traces of on-water wireless communication (see Section 4.3.1) to verify the above models. We start from the link PRR model given in Equation (4.1). The least square fitting of the average of the PRR measurements versus distance is given by

$$\mathrm{PRR}(d) = \frac{1}{2} + \frac{1}{2} \cdot \mathrm{erf}(-7.096 \log_{10} d + 26.14),$$

which is plotted in Figure 4.3. We can see that the fitted value for a_1 (i.e., -7.096) is very close to its theoretical value (i.e., $-5\sqrt{2} = -7.0711$). Moreover, the fitted curve well matches the average of the PRR measurements. Therefore, the model in Equation (4.1) can characterize the average performance of on-water link PRR. Although Equation (4.1) only captures the expected PRR, the control-theoretic connectivity maintenance algorithm presented in Section 4.3.3 accounts for the variance of PRR measurements.

We then conduct Monte Carlo simulations to verify the accuracy of swarm average PRR model given in Equation (4.2), and determine the values of the three coefficients. Specifically, for a given R, we generate a large number (20,000) of random placements of 10 sensors in the rendezvous circle. In the simulations, the PRR of each link is set to be the distance-based interpolation of real PRR measurements obtained in the aforementioned on-water experiment. Figure 4.4 shows the error bar of the swarm average PRR, where the variances are caused by the random sensor placements and estimation inaccuracy as well as the inherent stochasticity of wireless link. We then fit the curve defined by Equation (4.2) with the simulation results, as shown in Figure 4.4. From the figure, we can see that the approximate model for the swarm average PRR is fairly accurate. The fitted value for the coefficient c_1, c_2, and c are -1.201, 4.879, and 0.4783, respectively. These values are also adopted in the performance evaluation in Section 4.5.

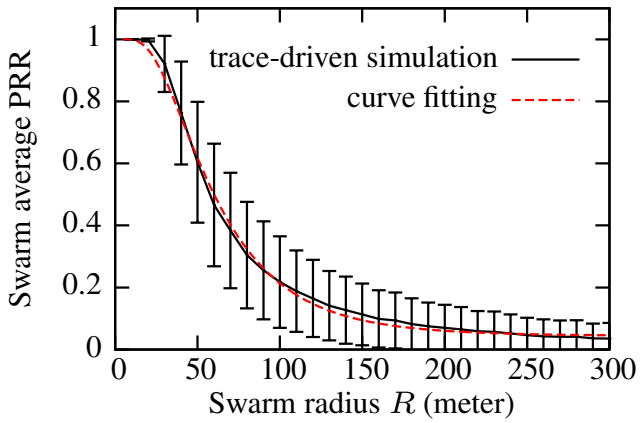


Figure 4.4: Swarm average PRR versus swarm radius. The error bar represents the standard deviation.

4.3.3 Swarm Connectivity Control

Our objective is to maintain the swarm average PRR at a desired level (denoted by δ) in the presence of various environment and system dynamics. From Figure 4.4, the swarm average PRR decreases with the swarm radius. However, the amount of information sampled by the sensors often increases with the swarm radius. Therefore, there is a trade-off between the amount of information obtained by the swarm and its connectivity. To avoid the loss of sensors that can have catastrophic consequence to the swarm, we ensure that the swarm is a well connected network in each rendezvous circle by setting a relatively high δ, e.g., 0.8 to 0.9. In this section, we first analyze the control laws based on the connectivity model in Equation (4.2) and then develop the connectivity maintenance algorithm.

The block diagram of the feedback control loop is shown in Figure 4.5. We denote $G_c(z)$, $G_p(z)$ and $H(z)$ as the transfer functions of the connectivity maintenance algorithm, the sensor swarm system and the feedback, which are expressed in z-transform representation. The z-transform provides a compact representation for time varying functions, where z represents a time shift operation. We refer interested reader to [68] for the details of z-transform and [1] for a few representative applications of discrete-time control theory to networking and computing systems. As shown in

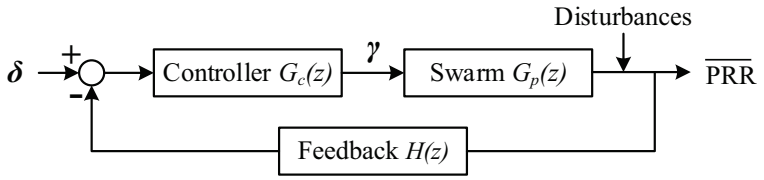


Figure 4.5: The closed loop for swarm connectivity control.

Figure 4.5, the desired PRR level δ is the *reference*, and the $\overline{\text{PRR}}(R)$ is the *controlled variable*. As $\overline{\text{PRR}}(R)$ is a nonlinear function of R (see Equation (4.2)), we define $\gamma = \text{erf}(c_1 \log_{10} R + c_2)$ as the *control input* to simplify the controller design. As a result, we have the swarm average PRR expressed as $\overline{\text{PRR}}(\gamma) \simeq (1-c) + c \cdot \gamma$. As this time-domain expression does not contain time shift, its z-transform is simply $G_p(z) = c$ [68]. In each sampling iteration, to ensure that the swarm head receives the measurements from all sensors, a sensor retransmits the lost packet until it receives an acknowledgement from the swarm head. At the end of each sampling iteration, the swarm head estimates the $\overline{\text{PRR}}(R)$ as $\frac{1}{N}\sum_{i=1}^{N}\frac{1}{\text{CTX}_i}$, where N is the number of sensors in the sensor swarm, and CTX_i is the number of (re-)transmissions of sensor i in the current sampling iteration. Such a passive estimation approach avoids transmitting a large number of measurement packets for estimating PRRs. Then, the swarm head updates γ based on the estimated $\overline{\text{PRR}}(R)$, and sets R in the next sampling iteration accordingly. As the feedback will take effect in the next iteration, $H(z) = z^{-1}$, which represents a delay of one iteration. Since the system is of zero order, a first-order controller is sufficient to achieve the stability and convergence [68]. Hence, we let $G_c(z) = \frac{\alpha}{1-\beta \cdot z^{-1}}$, where $\alpha > 0$ and $\beta > 0$. The settings of α and β need to ensure the system stability, convergence and robustness. Following the standard method for analyzing stability and convergence [68], the stability and convergence condition can be obtained as $\beta = 1$ and $0 < \alpha < 2/c$. The detailed analysis can be found in [106, 107].

In this chapter, we model three uncertainties that substantially affect the $\overline{\text{PRR}}(R)$ as the disturbances in the control loop shown in Figure 4.5. First, as shown in Figure 4.3, the PRR measure-

ments exhibit variance especially in the transition range from 25 m to 40 m. Second, the swarm topology changes with the random sensor positions, hence also causes variance to the $\overline{\text{PRR}}(R)$. Third, although the estimated PRR based on (re-)transmission number is unbiased, it has variance because of the limited number of samples. The error bars in Figure 4.4 show the overall standard deviation versus the swarm radius. From the figure, we find that in order to keep a satisfactory swarm average PRR around 0.8, the standard deviation is 0.12. We now discuss how to design $G_c(z)$ to reduce the impact of such random disturbances. From control theory [68], to minimize the effects of disturbance on the controlled variable $\overline{\text{PRR}}(R)$, the gain of $G_c(z)G_p(z)H(z)$ should be made as large as possible. By jointly considering the stability and convergence condition, we set $\alpha = 2b/c$ where b is a relatively large value within $[0,1]$. In the experiments conducted in this chapter, b is set to be 0.9.

Implementing $G_c(z)$ in the time domain gives the connectivity maintenance algorithm. According to Figure 4.5, we have $G_c(z) = \gamma(z)/(\delta - H(z)\overline{\text{PRR}})$. From $H(z)$ and $G_c(z)$, the control input can be expressed as $\gamma(z) = z^{-1}\gamma(z) + 2bc^{-1}(\delta - z^{-1}\overline{\text{PRR}})$, and its time-domain implementation is $\gamma_k = \gamma_{k-1} + 2bc^{-1}(\delta - \overline{\text{PRR}}_{k-1})$, where k is the index of sampling iteration. The swarm radius to be set in the k^{th} sampling iteration is given by $R_k = 10^{(\text{erf}^{-1}(\gamma_k)-c_2)/c_1}$.

4.4 Information-Theoretic Movement Scheduling

In this section, we first briefly introduce the Gaussian process model that characterizes many physical/biological phenomena, and present the field reconstruction algorithm. We then present the information-theoretic analysis for selecting the location of rendezvous circle in the next sampling iteration, which aims to maximize the accuracy of field reconstruction.

4.4.1 Physical Field

4.4.1.1 Spatiotemporal Gaussian Process Model

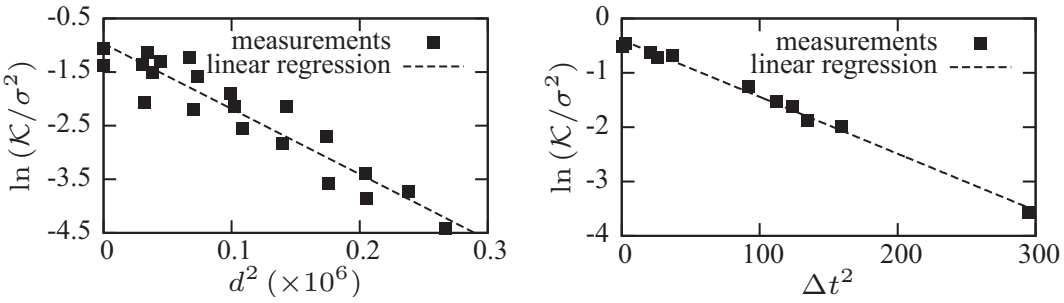
We assume that the monitored physical phenomenon follows the spatiotemporal Gaussian process [79]. Let $Z(p,t)$ denote the field variable at point $p \in \mathbb{R}^2$ and time $t \in [0,+\infty]$. For example, the surface phytoplankton population density is an important field variable of HABs. A Gaussian process can be fully characterized by the *mean function*, denoted by $\mathscr{M}(p,t)$, and the *covariance function*, denoted by $\mathscr{K}((p,t),(p',t'))$, where (p,t) and (p',t') are two time-space coordinates. In this chapter, we adopt the following covariance function that has been widely adopted [21,47,114]:

$$\mathscr{K}(d,\Delta t) = \sigma^2 \cdot \exp\left(-\frac{d^2}{2\varsigma_s^2}\right) \cdot \exp\left(-\frac{\Delta t^2}{2\varsigma_t^2}\right), \tag{4.3}$$

where $d = \| p - p' \|_{\ell_2}$, $\Delta t = |t - t'|$, σ^2 is the prior variance of any field variable, ς_s and ς_t are the spatial and temporal *kernel bandwidth*s, respectively. Therefore, the covariance function can be rewritten as $\mathscr{K}(d,\Delta t)$. The vector composed of the field variables at N time-space coordinates $\{(p_i,t_i) \,|\, i \in [1,N]\}$, denoted by $\mathbf{Z}$, follows the multivariate Gaussian distribution, i.e., $\mathbf{Z} \sim \mathscr{N}(\mathbf{m},\Sigma)$, where $\mathbf{m}$ and Σ are the mean vector and covariance matrix. Specifically, $\mathbf{m} = [\mathscr{M}(p_1,t_1),\ldots,\mathscr{M}(p_N,t_N)]$ and the $(i,j)^{\text{th}}$ entry of Σ is given by $\mathscr{K}(\| p_i - p_j \|_{\ell_2}, |t_i - t_j|)$. Sensor measurements can be corrupted by noises from the sensor circuitry and environment [114]. The reading at time-space coordinates (p,t), denoted by $R(p,t)$, is given by $R(p,t) = Z(p,t) + W$, where W is a zero-mean Gaussian noise with variance of σ_w^2.

4.4.1.2 Model Verification

We now verify the Gaussian process model using real temperature traces collected on Lake Fulmor, California [66]. The temperature readings on the lake surface were collected by 8 robotic



(a) $\ln(\mathcal{K}/\sigma^2)$ versus d^2 (b) $\ln(\mathcal{K}/\sigma^2)$ versus Δt^2

Figure 4.6: $\ln(\mathscr{K}/\sigma^2)$ versus d^2 and Δt^2.

boats over several hours. Applying logarithm to the covariance function $\mathscr{K}(d,\Delta t)$ yields $-2\cdot\ln\mathscr{K}(d,\Delta t)/\sigma^2 = d^2/\varsigma_s^2 + \Delta t^2/\varsigma_t^2$. Therefore, the quantities $\ln(\mathscr{K}/\sigma^2)$, d^2, and Δt^2 are expected to exhibit linear relationships. Figure 4.6 plots $\ln(\mathscr{K}/\sigma^2)$ versus d^2 and Δt^2, respectively. We can observe from the figure that the quantities exhibit linear relationships with small variations caused by the random noise W. The hyperparameters are estimated as $\varsigma_s = 6.42$ and $\varsigma_t = 7.15$. Therefore, the adopted $\mathscr{K}(d,\Delta t)$ well characterizes the spatiotemporal covariance of the water surface temperatures.

4.4.2 Field Reconstruction using a Robotic Sensor Swarm

In this section, we present how to reconstruct the field using measurements collected by the sensor swarm. To facilitate the expression, we define $\mathbf{H}$ as a row vector composed of all measurements, i.e., $\mathbf{H} = [R(p_1,t_1),\ldots,R(p_N,t_N)]$, $\mathbf{m}$ as a row vector composed of the corresponding prior mean values, and T as the time duration of each sampling iteration. Therefore, each $t_i\,(i \in [1,N])$ is always multiple of T. The $\mathbf{H_c}$ is a $3\times N$ matrix, where each column is the time-space coordinates of the corresponding measurement in $\mathbf{H}$. The objective of reconstructing a Gaussian process field is to estimate the posterior mean and variance at any time-space coordinates (p,t) given $\mathbf{H}$, which

are denoted by $\mathbb{E}[Z|\mathbf{H}]$ and $\mathrm{Var}[Z|\mathbf{H}]$. The estimates are given by [77, 79]:

$$\mathbb{E}[Z|\mathbf{H}] = \mathscr{M}(p,t) + \widetilde{\Sigma}[(p,t),\mathbf{H_c}] \cdot \widetilde{\Sigma}^{-1}[\mathbf{H_c}] \cdot (\mathbf{H} - \mathbf{m})^{\top}, \tag{4.4}$$

$$\mathrm{Var}[Z|\mathbf{H}] = \sigma^2 - \widetilde{\Sigma}[(p,t),\mathbf{H_c}] \cdot \widetilde{\Sigma}^{-1}[\mathbf{H_c}] \cdot \widetilde{\Sigma}^{\top}[(p,t),\mathbf{H_c}], \tag{4.5}$$

where $\widetilde{\Sigma}$ is a matrix calculated from the covariance matrix Σ of the field variables at $\mathbf{H_c}$. Specifically, the $(i,j)^{\text{th}}$ entry of $\widetilde{\Sigma}$ is given by $\widetilde{\Sigma}_{ij} = \Sigma_{ij} + \theta_{ij}\frac{\sigma_w^2}{\sigma}$, where $\theta_{ij} = 1$ if $i = j$, and otherwise $\theta_{ij} = 0$. There are three interesting observations from Equations (4.4) and (4.5). First, because of the spatiotemporal correlation, the posterior variance (i.e., the uncertainty) is reduced given the measurements $\mathbf{H}$. Second, from Equation (4.5), the posterior variance does not depend on the prior and posterior means. As our movement scheduling algorithm aims to reduce the variance, it does not need the knowledge of means. Third, the dimension of $\widetilde{\Sigma}$ increases along the accumulation of sensor measurement. As a result, the high dimensional Σ poses substantial computation overhead to calculate its inversion in Equations (4.4) and (4.5) on resource-constrained robotic sensors.

From the above three observations, an important design of our approach is to separate the following two tasks:

The first task is sensor movement scheduling. This task is executed on the swarm head in each sampling iteration, which aims to reduce the variance given in Equation (4.5). Section 4.4.3 to Section 4.4.5 will present the details of our sensor movement scheduling algorithms. In particular, as the sensor movement scheduling involves calculating the inversion of $\widetilde{\Sigma}$ in Equation (4.5) on the swarm head, in Section 4.4.4, we propose two measurement truncation schemes that can significantly reduce the computation overhead.

The second task is field reconstruction. This task computes Equations (4.4) and (4.5) based

on collected measurements. It can be executed on either on the swarm head if it has sufficient computation capability, or a remote data processing center after measurements are fetched back.

4.4.3 Information-Theoretic Swarm Center Selection

We now discuss the selection of the center of the next rendezvous circle (referred to as *swarm center*), which aims to improve the accuracy of the field reconstruction algorithm (i.e., Equations (4.4) and (4.5)).

4.4.3.1 Problem Formulation

Suppose that the swarm has N robotic sensors and will schedule the sensor movements for the next (i.e., the k^{th}) sampling iteration. Let $\mathbf{V}$ denote the region to be reconstructed and the time of reconstruction, and $\mathbf{S}$ denote the set of target time-space coordinates for all sensors.[1] Hence, $\mathbf{S}$ can be represented as $(\{p_1, p_2, \ldots, p_N\}, kT)$, where p_i is the target position of sensor i. Let p'_c and p_c denote the swarm center in the $(k-1)^{\text{th}}$ and k^{th} sampling iteration, and R_k is the scheduled swarm radius for the k^{th} iteration by the connectivity maintenance algorithm. The optimal solution of $\mathbf{S}$ maximizes the following information-theoretic metric:

$$\Omega(\mathbf{S}) = \mathbb{H}[\mathbf{V} \setminus \mathbf{S} \,|\, \mathbf{H_c}] - \mathbb{H}[\mathbf{V} \setminus \mathbf{S} \,|\, \mathbf{H_c} \cup \mathbf{S}], \tag{4.6}$$

subject to

$$\begin{aligned} &\| p'_c - p_c \|_{\ell_2} \leq L; \\ &\| p_i - p_c \|_{\ell_2} \leq R_k, \forall i \in [1, N]; \end{aligned} \tag{4.7}$$

[1] To simplify the presentation, $\mathbf{V}$ refers to both the set of field variables and the corresponding time-space coordinates. So do $\mathbf{S}$ and $\mathbf{H_c}$.

where the $\mathbb{H}[\cdot]$ denotes entropy and quantifies the uncertainty. In Equation (4.6), the term $\mathbf{V} \setminus \mathbf{S}$ represents the set of ungauged sites in the current iteration, and the term $\mathbf{H_c} \cup \mathbf{S}$ represents the set of visited time-space coordinates after the current iteration. Therefore, $\mathbb{H}[\mathbf{V} \setminus \mathbf{S} \,|\, \mathbf{H_c}]$ represents the uncertainty at the ungauged sites (i.e., $\mathbf{V} \setminus \mathbf{S}$) given the historically visited positions, and $\mathbb{H}[\mathbf{V} \setminus \mathbf{S} \,|\, \mathbf{H_c} \cup \mathbf{S}]$ represents the uncertainty at the ungauged sites after additionally sampling the field at $\mathbf{S}$. As a result, the above problem aims to maximize the drop of entropy at the ungauged sites after the current iteration by sampling the field variables at $\mathbf{S}$ given the historical measurements at $\mathbf{H_c}$. The above problem formulation adopts the drop of entropy as the performance metric, which is defined by Equation (4.6). The constraint in the first part of Equation (4.7) specifies the reachable area of the swarm due to limited sensor movement speed. For example, we can set $L = v \cdot T$, where v is the maximum speed of the robotic sensors. The constraint in the second part of Equation (4.7) ensures the scheduled swarm radius. These constraints are also illustrated in Figure 4.1.

Note that the posterior entropy for the ungauged sites [104] is another widely adopted performance metric in field reconstruction studies. We now identify the relationship between posterior entropy and our metric defined in Equation (4.6). Suppose the current iteration is the k^{th} iteration of the sampling process. By cumulating Equation (4.6) of each iteration, we can approximate the accumulative entropy reduction (denoted as $\sum_{i=1}^{k} \Omega(\mathbf{S}_i)$) as: $\sum_{i=1}^{k} \Omega(\mathbf{S}_i) \approx \mathbb{H}[\mathbf{V} \setminus \mathbf{S}_1] - \mathbb{H}[\mathbf{V} \setminus \mathbf{S}_k \,|\, \mathbf{H_c} \cup \mathbf{S}_k]$, where $\mathbf{S}_i$ is the set of sampling positions in the i^{th} iteration and $\mathbf{H_c} = \mathbf{S}_1 \cup \mathbf{S}_2 \ldots \cup \mathbf{S}_{k-1}$ represents all the gauged sites till the k^{th} iteration. In particular, the term $\mathbb{H}[\mathbf{V} \setminus \mathbf{S}_k \,|\, \mathbf{H_c} \cup \mathbf{S}_k]$ denotes the posterior entropy after the k^{th} iteration. Note that $\mathbb{H}[\mathbf{V} \setminus \mathbf{S}_1]$ is a constant. Therefore, there is a simple linear relationship between the posterior entropy and the accumulated entropy reduction. From this relationship, the minimum posterior entropy is achieved when the entropy reduction in each iteration is maximized.

4.4.3.2 Swarm Center Selection Algorithm

A similar problem without the condition $\mathbf{H_c}$ and the constraints in Equation (4.7) has been proven to be NP-hard [48]. Hence, the above problem has prohibitively high complexity that is not practical for robotic sensing platforms. In this chapter, we propose a heuristic approach that approximates the whole swarm by its center, which is selected from a set of discrete candidate points. By such an approximation, we avoid the complex inter-point dependence given by Equation (4.3), hence largely reduce the computation overhead. We will evaluate the performance of this approximation in Section 4.5.1.8. Under the proposed heuristic approach, we adopt *mutual information* (MI) and *posterior entropy* (PE) to quantify the information reward. As these two metrics differ in computation overhead and the resulted reconstruction accuracy, they allow the system designer to choose desirable trade-off between the overhead and accuracy subject to the budget of computation resources of robotic sensors.

We first discuss the MI-based metric. The MI of a random variable X given a set of random variables $\mathbf{Y}$ can be expressed as $\mathbb{I}[X;\mathbf{Y}] = \mathbb{H}[X] - \mathbb{H}[X \mid \mathbf{Y}]$, where

$$\begin{aligned} \mathbb{H}[X \mid \mathbf{Y}] &= \frac{1}{2}\log\left(2\pi \mathrm{e} \cdot \mathrm{Var}[X \mid \mathbf{Y}]\right), \\ \mathrm{Var}[X \mid \mathbf{Y}] &= \mathrm{Var}[X] - \Sigma[X,\mathbf{Y}] \cdot \Sigma^{-1}[\mathbf{Y}] \cdot \Sigma^{\top}[X,\mathbf{Y}]. \end{aligned} \tag{4.8}$$

The $\Sigma[X,\mathbf{Y}]$ is a row vector composed of the covariances of X with each variable in $\mathbf{Y}$, and $\Sigma^{-1}[\mathbf{Y}]$ is the inverse of the covariance matrix of $\mathbf{Y}$. Given available measurements at $\mathbf{H_c}$, the MI-based information reward for the swarm centered at position p_c, denoted by $\Omega_{\mathrm{MI}}(p_c, kT)$, is defined as

$$\Omega_{\mathrm{MI}}(p_c,kT) = \mathbb{I}[\mathbf{V} \setminus (p_c,kT); (p_c,kT) \mid \mathbf{H_c}] = \mathbb{H}[(p_c,kT) \mid \mathbf{H_c}] - \mathbb{H}[(p_c,kT) \mid \mathbf{V} \cup \mathbf{H_c} \setminus (p_c,kT)].$$

The above information reward metric characterizes the drop of uncertainty about the region other than p_c given all historical measurements if the swarm is centered at p_c in the next iteration. The swarm center selection is hence to maximize Ω_{MI}, subject to the constraints in Equation (4.7).

The complexity for computing Ω_{MI} for a certain p_c is $O(|\mathbf{V}|^3)$. However, as the aquatic phenomenon of interest (e.g., HABs) often affects a large spatial area, computing Ω_{MI} can incur high overhead. To reduce the computation overhead, we propose another information reward metric based on PE:

$$\Omega_{\mathrm{PE}}(p_c, kT) = \mathbb{H}[(p_c, kT) \,|\, \mathbf{H_c}].$$

Different from Ω_{MI}, Ω_{PE} characterizes the uncertainty drop at the swarm center p_c in the next iteration given the historical measurements. For each certain p_c, the complexity of computing Ω_{PE} is $O(|\mathbf{H_c}|^3)$, which is much smaller than that of Ω_{MI}. Although such a metric does not necessarily lead to the maximum uncertainty drop for the ungauged sites, it can reduce the computation overhead by only considering the most uncertain positions.

4.4.4 Truncating Historical Measurements

Both the metrics Ω_{MI} and Ω_{PE} involve storing and inverting the covariance matrix $\Sigma[\mathbf{H_c}]$ when computing Equation (4.8). This imposes substantial challenges to the robotic sensing platforms with limited computation resources. For example, a TelosB mote equipped with 10 KB RAM can store at most a 50×50 covariance matrix. Moreover, matrix inversion is a computation-intensive operation with at least cubic complexity with respect to the number of historical measurements. To develop practical information-theoretic movement scheduling algorithms for robotic sensors, we propose two schemes for truncating the historical measurements. Both schemes select K measurements to compose the covariance matrix.

The first scheme selects K historical measurements with the largest covariances regarding a candidate swarm center. This scheme is referred to as *cov-trunc*. The rationale of *cov-trunc* is as follows. As we only use a subset of historical measurements, the conditional variance in Equation (4.8) will increase. The *cov-trunc* scheme maximizes each element in $\Sigma[X, \mathbf{Y}]$, and hence can efficiently suppress the undesired increase of the conditional variance caused by the truncation. The drawback of *cov-trunc* is that it needs to truncate the historical measurements for each candidate swarm center when maximizing Ω_{MI} and Ω_{PE}. As a result, a matrix inversion operation is needed for each candidate swarm center, which results in high computation overhead for the swarm head. To address this issue, we propose another truncating scheme, referred to as *time-trunc*. The *time-trunc* selects the most recent K historical measurements. As the most recent sampling positions are generally in the proximity of the swarm in the next iteration, *time-trunc* can well approximate *cov-trunc* even though it ignores the spatial correlation. The *time-trunc* scheme has the following two advantages. First, it only needs a matrix inversion operation for each sampling iteration. Second, the swarm head only needs to maintain a first-in-first-out historical measurement buffer with size of K. This buffer can be easily migrated in the swarm head rotation process for the purpose of balancing energy consumption. However, we note that the performance of these truncation schemes depends on the properties of the underlining aquatic processes, such as the kernel bandwidths (i.e., ς_s and ς_t) and the affected region (i.e., $\mathbf{V}$). In Section 4.5.1.6, we will evaluate the impact of historical measurements truncation on reconstruction accuracy.

4.4.5 Sensor Movement Scheduling

As discussed in Section 4.2, once the swarm center and radius are determined, the swarm head randomly selects N positions (denoted by $\mathbf{p}'$) in the rendezvous circle. We let $\mathbf{p}$ denote the current positions of robotic sensors. To prolong the lifetime of the robotic sensor swarm, we find the

element mapping from $\mathbf{p}$ to $\mathbf{p}'$, such that the sum of sensors' movement distances is minimized. Under this movement scheduling scheme, there is no crossing between sensors' moving paths. The proof can be found in [107]. Therefore, our movement scheduling scheme is collision-free. This element mapping problem can be solved by existing algorithms such as Munkres assignment algorithm [12] with a complexity of $O(N^3)$. Once the mapping is found, the swarm head sends the target position to each robotic sensor, which then moves toward the target position. While moving toward the target position, the robotic sensor can adopt a feedback-based motion control algorithm that adaptively corrects the motion errors based on the localization result, using a potential function approach [6]. The motion control of robotic fish is beyond the scope of this chapter and the details can be found in [6].

4.5 Performance Evaluation

We evaluate the performance of the proposed algorithms by trace-driven simulations and implementation on hardware. First, we evaluate the connectivity maintenance and the swarm movement scheduling algorithms using extensive simulations based on real data traces of water surface temperature field [66] and on-water ZigBee wireless communication. Second, we implement one of the proposed swarm movement scheduling algorithms on TelosB sensing platform and evaluate its overhead. The results provide insights into the feasibility of adopting advanced information-theoretic movement scheduling algorithms on mote-class robotic sensing platforms.

4.5.1 Trace-Driven Simulations

4.5.1.1 Simulation Methodology and Settings

In the simulations, 10 robotic sensors are used to reconstruct a scalar field in a square region. The hyperparameters of the Gaussian process are set to be $[\sigma^2, \varsigma_s, \varsigma_t] = [9, 6, 8]$, unless otherwise specified. Note that these settings are consistent with [86, 116] and obtained from real on-water temperature traces [66]. Initially, the robotic sensors are randomly deployed in a small region with radius of 10 m. In each sampling iteration, the PRR of each link is set to be the distance-based interpolation of real on-water PRR traces measured by two IRIS motes on Lake Lansing, Michigan (see Section 4.3.2). Other settings include: desired swarm connectivity level $\delta = 0.9$, sampling iteration duration $T = 5$ min, sensor movement speed $v = 0.2$ m/s, and $L = v \times T = 60$ m.

4.5.1.2 Swarm Connectivity Maintenance and Communication Overhead

We first compare our connectivity maintenance algorithm with a heuristic baseline algorithm. The heuristic algorithm adopts the Kalman filter to update the coefficient c in Equation (4.2) based on the recently estimated $\overline{\text{PRR}}(R)$. The next swarm radius is then obtained by solving Equation (4.2). Recall that our approach assigns a fixed value to c and tunes the swarm radius directly. Figure 4.7 plots the $\overline{\text{PRR}}(R)$ in the first 10 sampling iterations. The range of swarm radius after 10 iterations is $[24, 36]$. The error bars, calculated from 20 runs, are caused by the various disturbances discussed in Section 4.3.3. We can see that the swarm average PRR controlled by our algorithm quickly converges to the desired connectivity level. In contrast, the heuristic algorithm diverges from the reference. This is because the Kalman filter does not tune the swarm radius directly, and incorrectly updates the coefficient c in the control cycle. To evaluate the response of our algorithm to the sudden changes of the wireless link quality, we artificially reduce the PRR measurements by

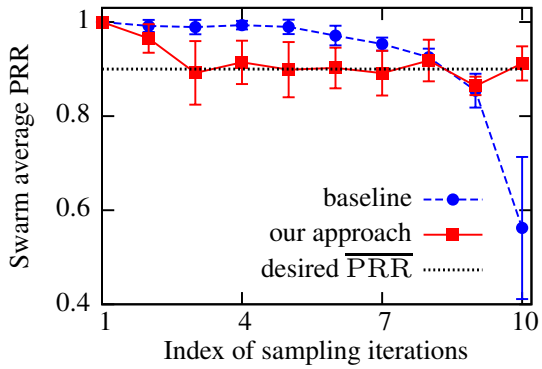


Figure 4.7: Swarm average PRR versus sampling iteration. The error bar represents the standard deviation.

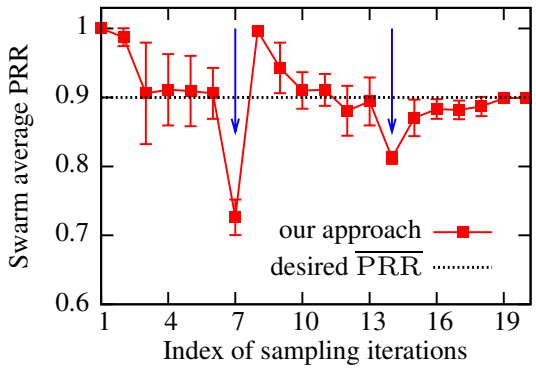
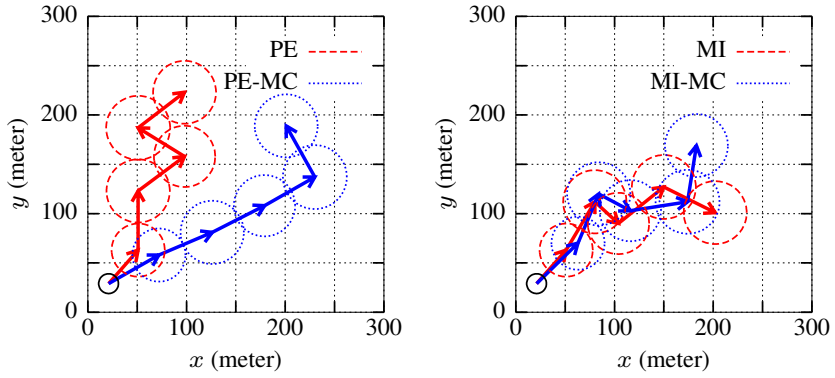


Figure 4.8: Impulsive and step responses (at the 7^{th} and 14^{th} iterations) of our algorithm.

20% only in the 7^{th} iteration (i.e., the left arrow in Figure 4.8) and continuously reduce the PRR measurements by 10% after the 14^{th} iteration (i.e., the right arrow in Figure 4.8). For both types of changes, our algorithm can converge within a few iterations.

In each sampling iteration, the communication overhead is mainly caused by the packet loss. Hence, we employ the total number of transmissions in collecting all sensor measurements as the evaluation metric. When a node transmits a packet to the swarm head, the packet is delivered with a success probability equal to the PRR. The node re-transmits the packet up to 20 times before it is dropped. The packet to the swarm head includes sensor ID, spatiotemporal coordinates and measurement. The packet to the sensor contains the target position in the next rendezvous circle. Consider a typical sampling iteration, e.g., the 4^{th} iteration in Figure 4.8 where a swarm radius around 32 m yields a swarm average PRR about 0.9. Our simulation results show that for a swarm consists of 10 nodes, the number of transmitted packets (for two-way communications) has a mean of 38 and a standard deviation of 8. Even if all these packets are transmitted sequentially, the delay will be within a second, because transmitting a TinyOS packet only takes about 10 milliseconds on typical mote-class sensor platforms. Therefore, our approach has low communication overhead.



(a) PE and PE-MC (b) MI and MI-MC

Figure 4.9: Trajectories of a robotic sensor swarm with 10 sensors in the first 6 sampling iterations in the reconstruction of a $300 \times 300\,\mathrm{m}^2$ field.

4.5.1.3 Effectiveness of Swarm Center Selection

We now compare the two swarm center selection approaches presented in Section 4.4.3.2 (referred to as MI and PE) with three other baseline approaches. The first baseline (referred to as MI-MC) finds the next swarm center according to the metric $\Omega(\mathbf{S})$ in Equation (4.6), where $\mathbf{S}$ is a set of random sensor placements within the rendezvous circle. For each candidate p_c, 100 random sensor placements are generated (i.e., Monte Carlo trials) and the average Ω is used as the information reward relating to p_c. The second baseline (referred to as PE-MC) is similar to the MI-MC, except that the metric is given by $\mathbb{H}(\mathbf{S}\,|\,\mathbf{H_c})$. These two Monte Carlo baselines give the near-optimal swarm centers regarding the MI and PE metrics, respectively. However, due to the high computation overhead of Monte Carlo method, these two baselines are not suitable for mote-class sensor platforms. A random walk approach is employed as the third baseline (referred to as RW). Specifically, the swarm head selects a random position as p_c subject to the constraints in Equation (4.7).

We first show the swarm trajectories scheduled by various approaches. Figure 4.9 plots the trajectories of a sensor swarm in the first 6 sampling iterations. Note that the swarm radius is controlled by the connectivity maintenance algorithm. We can see that, for all approaches, two consecutive rendezvous circles can overlap. This is because the correlation of the Gaussian process

exists in both spatial and temporal domains, moving to a farther location does not necessarily increase the overall information reward. Note that, if only spatial correlation is considered, the swarm will move to the farthest unexplored areas. From Figure 4.9(a), PE and PE-MC output different trajectories. As the next swarm location is affected by historical sensor positions which were randomly generated, the trajectories can be different under different approaches, and even different under the same approach in different simulation runs. However, they follow the similar trend of spreading out in the field.

We then compare the effectiveness of various approaches based on the criterion Ω given in Equation (4.6), which quantifies the drop of uncertainty at the ungauged sites at current time. Note that in each sampling iteration, the position of the rendezvous circle is determined by the specified information reward metric, and the radius is chosen to maintain the swarm connectivity. Figure 4.10 plots Ω versus the index of sampling iteration. The error bar represents the standard deviation over multiple simulation runs. We can see that Ω increases over time as more measurements are taken. From the figure, we find that the MI and MI-MC outperforms the PE by 10% and 19%, respectively, in the 5^{th} sampling iteration. However, they have much higher computation overhead than PE. Specifically, MI and MI-MC take about 20 and 8000 times of the execution time of PE, respectively. The RW approach yields the worst accuracy. Moreover, the gap between our approach and the corresponding Monte-Carlo-based baseline (e.g., PE and PE-MC) gives the performance loss caused by approximating the rendezvous circle with the swarm head. Due to the large number of Monte Carlo trials, the baseline approaches achieve the better performance with substantially heavier computation overhead that is infeasible on mote-class platforms.

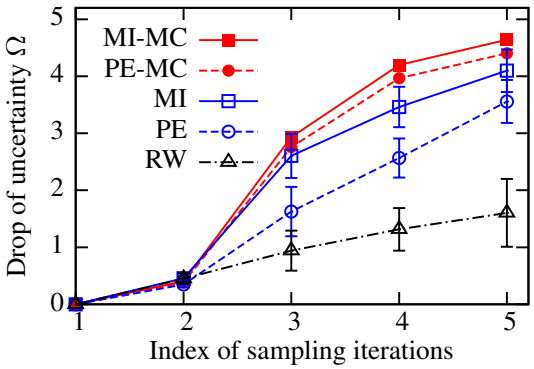


Figure 4.10: Information reward versus sampling iteration.

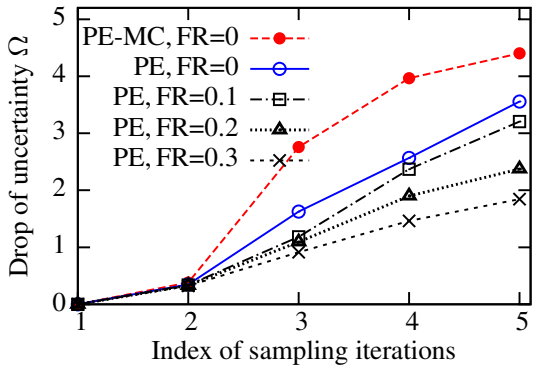


Figure 4.11: Impact of sensing failure on information reward (FR shorts for failure rate).

4.5.1.4 Impact of Sensing Failure

We now evaluate the impact of *sensing failure* on reconstruction performance. Sensing failure means that the robotic sensing platform cannot sample the field temporarily. In this set of simulations, we take the PE approach as an example and introduce random sensing failures. Specifically, each sensor has a sensing failure rate (10%, 20%, and 30%) in a sampling iteration. For sensors that experience sensing failure, their sampling positions will not be used in computing the drop of uncertainty Ω and scheduling the swarm movement in the next iteration. We adopt the PE-MC and PE approaches with zero failure rate as baselines, in which the PE-MC approach gives the near-optimal swarm center regarding the PE metric. For each failure rate, we conduct 6 runs of simulations. The average information rewards are plotted in Figure 4.11. We can observe that our approach can achieve comparable reconstruction performance in the presence of relatively low sensing failure rate (e.g., 10%). As the sensing failure rate increases, the reconstruction performance drops. This is because the decreased sampling diversity leads to inaccuracy in swarm position selection. Note that, in addition to sensing failure, sensors are also subject to *hardware failure*, *motion and control failure*. In particular, hardware failure means that the robotic sensing platform completely fails and the swarm will lose the failed node. Since a swarm consists of a

limited number of nodes, the robotic sensing platform should be designed to have a low hardware failure rate to ensure long-term monitoring. Motion and control failure is caused by errors in swarm size control and sensor motion control such that the node cannot communicate with the swarm head. We have specifically presented two recovery mechanisms in [106, 107] to address such failures.

4.5.1.5 Effectiveness of Random Position Selection

In this section, we analyze the effectiveness of random sensor position selection regarding information reward. In our approach, the position of each sensor in the next sampling iteration is randomly selected by the swarm head within the rendezvous circle. We note that the proposed MI/PE-based metrics can also be used to guide the sensor position selection. Specifically, the swarm head sequentially selects each sensor's position based on the MI/PE-based metrics, such that each added sensor maximizes the information reward metric. We refer to the PE-based sequential sensor position selection approach as PE-Seq. This set of simulations only evaluate the PE-based approaches, which help us understand the impact of random sensor position selection scheme. To make a fair comparison, we set the swarm radius of PE-Seq identical to that of PE in each iteration. Figure 4.12 plots Ω versus the index of sampling iteration for PE-MC, PE-Seq, and PE, respectively. From the figure, we can see that the random sensor position selection (i.e., PE) gives comparable performance with the metric-guided sensor position selection (i.e., PE-Seq). The slightly better performance of PE-Seq is achieved at the cost of intensive computation overhead in determining each sensor's position. Due to the sequential execution process, this scheme cannot be executed on low-power sensing platforms at runtime. Therefore, within a rendezvous circle, the random placement of sensors does not significantly affect the information reward. Hence, our approach that combines metric-based swarm center position selection and in-swarm random sen-

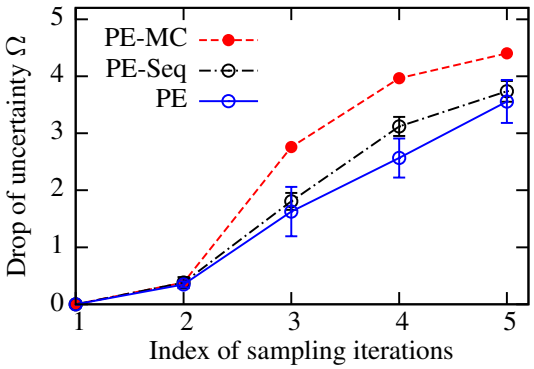


Figure 4.12: Impact of sensor position selection on the information reward.

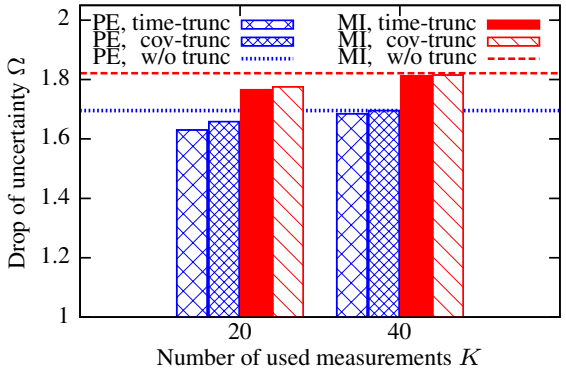


Figure 4.13: Information reward in the 15th iteration versus number of used measurements K.

sor position selection, not only is an effective solution in terms of information reward, but also simplifies the motion control of the sensor swarm as well as reduces the computation overhead of swarm head.

4.5.1.6 Impact of Historical Measurements Truncation

In this set of simulations, we compare the performance of various combinations of the MI/PE-based metrics and the two truncation schemes presented in Section 4.4.4. The number of used historical measurements, i.e., K, is set to be 20 or 40. The robotic sensor swarm is deployed in a $1000 \times 1000\,\mathrm{m}^2$ square region. Figure 4.13 plots the performance criterion Ω at the 15th sampling iteration under various settings. Without the truncation scheme, all historical measurements are used and hence a greater Ω is achieved. Moreover, we can see that Ω increases with K. An interesting observation is that the truncation schemes with $K = 40$ yield almost the same performance obtained by using all historical measurements. In addition, the *time-trunc* and *cov-trunc* schemes have comparable performance. Therefore, the *time-trunc* scheme with a small K can achieve satisfactory performance.

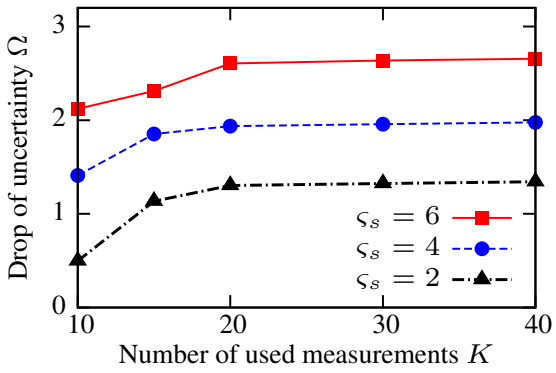


Figure 4.14: Information reward versus number of used measurements K under various ς_s.

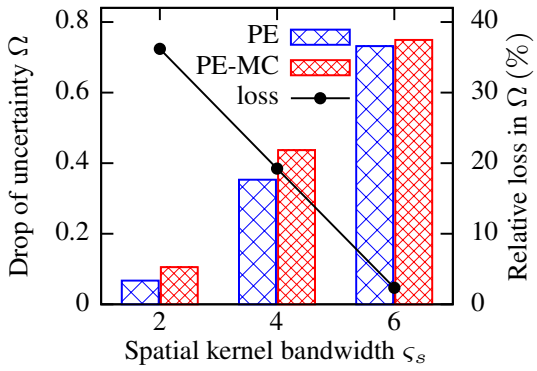


Figure 4.15: Impact of kernel bandwidth on the approximation performance.

4.5.1.7 Impact of Kernel Bandwidth

The kernel bandwidths are important hyperparameters of the Gaussian process. We focus on the impact of the spatial kernel bandwidth ς_s while keeping the temporal kernel bandwidth ς_t fixed. Figure 4.14 plots performance criterion Ω versus K, under various settings of ς_s. We can observe that Ω increases with ς_s. This is because the stronger spatial correlation introduced by the larger ς_s can lead to a greater posterior entropy drop at the ungauged sites. Moreover, we can see from the figure that the performance becomes saturated after K is greater than 20 for various settings of ς_s.

4.5.1.8 Approximation Performance

In this section, by taking the PE approach as an example, we evaluate the performance of our heuristic that approximates the whole swarm by its center. As discussed in Section 4.5.1.3, the PE-MC approach gives the near-optimal swarm center regarding the PE metric. We assess the approximation performance in terms of the relative loss in Ω, which is calculated as $(\Omega_{\text{PE-MC}} - \Omega_{\text{PE}})/\Omega_{\text{PE-MC}}$. In the first set of simulations, we consider the impact of kernel bandwidth. Specifically, we vary the spatial kernel bandwidth ς_s while keeping the temporal kernel bandwidth ς_t and the swarm radius R fixed. The results are plotted in Figure 4.15. We can observe that the relative

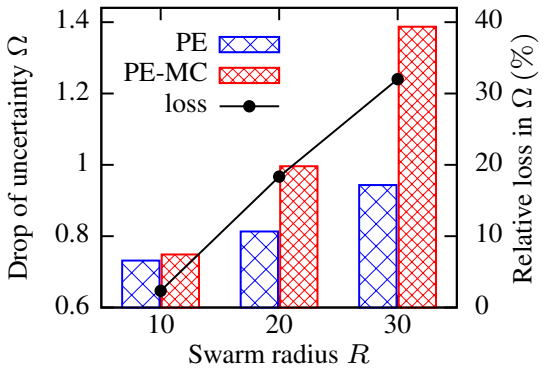


Figure 4.16: Impact of swarm radius on the approximation performance.

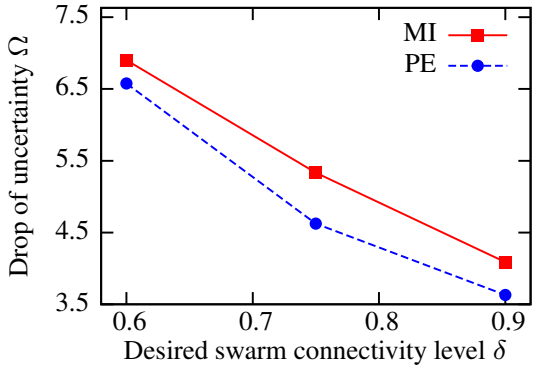
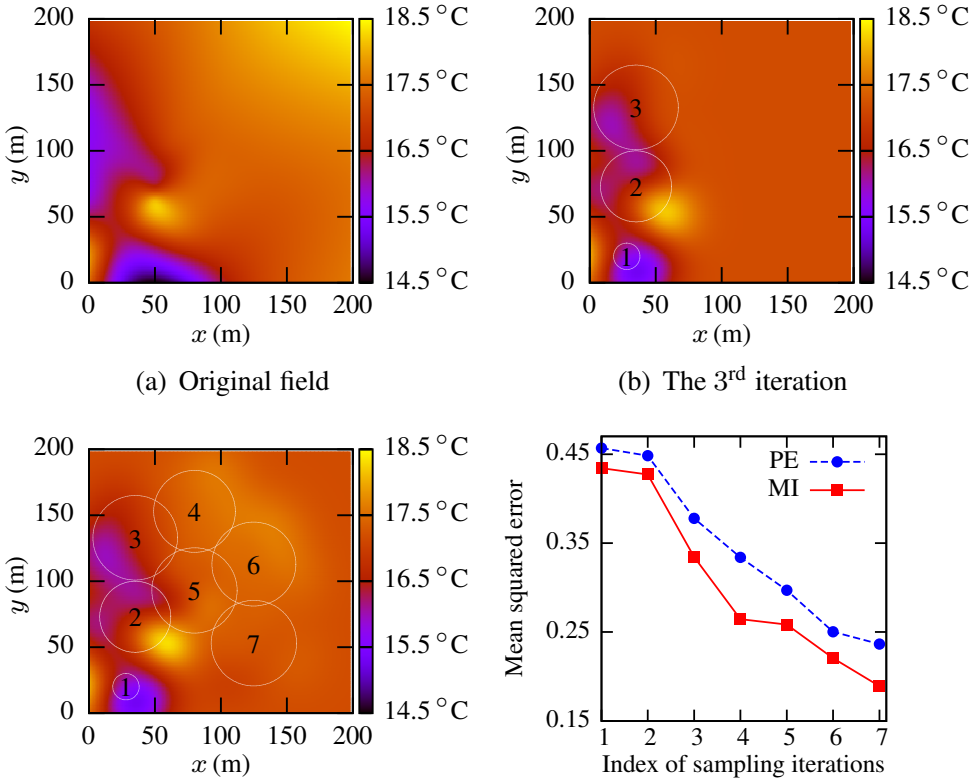


Figure 4.17: Information reward versus desired swarm connectivity level.

loss decreases with the kernel bandwidth. This result is consistent with the intuition that, when nearby positions are more correlated (i.e., a larger kernel bandwidth), the swarm center is a better representation of nearby positions. In the second set of simulations, we consider the impact of swarm radius. Specifically, we vary the swarm radius while keeping the kernel bandwidths fixed. Figure 4.16 shows the impact of swarm radius on the relative loss. From the figure, we can see that the relative loss increases with the swarm radius. This result is consistent with the intuition that, the swarm center can better represent the whole swarm when the swarm size is smaller.

4.5.1.9 Information Reward versus Swarm Connectivity

As discussed in Section 4.3.3, there is a trade-off between the amount of information obtained by the sensor swarm and its connectivity level. In this set of simulations, we quantitatively evaluate the trade-off. Specifically, we vary the desired swarm connectivity level, i.e., δ, and compare the resulted information rewards. Other settings are consistent with those presented in Section 4.5.1.1. For each δ, the control-theoretic algorithm adaptively tunes the radius of rendezvous circle to maintain the swarm connectivity. According to Equation (4.2), a larger δ generally requires a smaller rendezvous circle, which will result in less information reward. Figure 4.17 plots the achieved information reward after 5 sampling iterations versus the desired swarm connectivity



(a) Original field (b) The 3rd iteration

(c) The 7th iteration (d) Reconstruction error

Figure 4.18: Temperature field reconstruction using a robotic sensor swarm. The numbers in the circles represent the sequence of the rendezvous circles.

level. The decreasing relationship between drop of uncertainty and desired swarm connectivity level shown in Figure 4.17 verifies the trade-off.

4.5.1.10 Accuracy of Field Reconstruction

In this set of simulations, we reconstruct a field using 10 robotic sensors. We first generate a temperature field based on the temperature data [66] collected at 8 locations on the surface of Lake Fulmor, California, which has an area of about 3 acres. We have verified that temperature data follows the spatiotemporal Gaussian process in Section 4.4.1. However, the data at 8 locations are not sufficient to drive the simulations. Therefore, we use an existing tool [29] to fit a $200 \times 200\,\mathrm{m}^2$ ($\simeq 10\,\mathrm{acres}$) Gaussian process field based on the traces, as shown in Figure 4.18(a). For

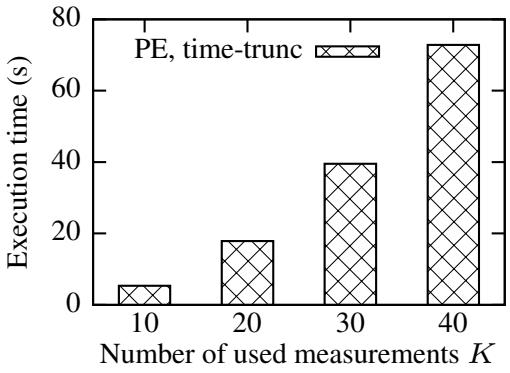


Figure 4.19: Execution time of PE *time-trunc* scheduling versus number of used measurements K.

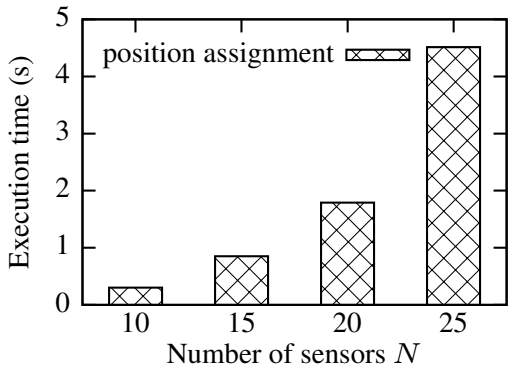


Figure 4.20: Execution time of sensor position assignment versus number of sensors N.

the ease of illustration, the field does not change with time, although our approach can deal with temporal evolution of the field. The movement of the swarm is scheduled by PE without truncation. Sensor measurements in the simulation are corrupted by zero-mean Gaussian noise with variance of 0.15 [48, 114]. In the reconstruction, the mean function $\mathscr{M}(p,t)$ is set to be a fixed value of the average temperature in Figure 4.18(a). Figure 4.18(b) and Figure 4.18(c) show the reconstructed field after the 3$^{\text{rd}}$ and 7$^{\text{th}}$ sampling iteration, as well as the trajectories of the swarm. Figure 4.18(d) plots the mean squared error (MSE) of the reconstructed temperature filed versus the index of sampling iteration. For comparison, we also include the MSE under MI approach. Due to space limitation, the corresponding reconstructed fields and swarm trajectories are omitted here. From Figure 4.18, we can see that the reconstruction accuracy is improved along with the movement of the swarm.

4.5.2 Overhead on Sensor Hardware

We have implemented the PE-based *time-trunc* swarm center selection algorithm and the sensor movement scheduling algorithm in TinyOS 2.1 on TelosB platform. We ported the C implementation of Cholesky decomposition algorithm in GNU Scientific Library [32] to TinyOS to invert

matrix in the swarm center selection algorithm. We also implement the Munkres algorithm in TinyOS to schedule each sensor's movement. Figure 4.19 and Figure 4.20 plot the execution times of the two algorithms in one sampling iteration, respectively. We can see that the PE-based *time-trunc* algorithm takes about one minute when 40 historical measurements are used. The Munkres algorithm for position assignment only takes 4.5 seconds when 25 robotic sensors are deployed. As our current implementation employs extensive floating-point computation, the above processing delays can be further reduced by using fixed-point arithmetic. Nevertheless, a delay of about one minute is acceptable since the duration of each iteration can be much longer than that. Note that the MI metric and the *cov-trunc* scheme result in very long processing delays on TelosB platform because of large search space and repeated matrix inversion operations.

4.6 Conclusion

This chapter proposes a cyber-physical approach to spatiotemporal aquatic field reconstruction using inexpensive, low-power, mobile sensor swarms. Our approach features a rendezvous-based mobility control scheme where a sensor swarm collaborates to sense the environment in a series of carefully chosen rendezvous regions. We design a novel feedback control algorithm that maintains the desirable level of wireless connectivity of a sensor swarm. We present new information-theoretic analysis to guide the selection of rendezvous regions such that the field reconstruction accuracy is maximized. We evaluate our approach by extensive trace-driven simulations and implementation on real sensor hardware. Our results show that the connectivity of robotic sensors can be maintained robustly in the presence of significant physical uncertainties. Moreover, despite the limited mobility, a sensor swarm can accurately reconstruct large dynamic aquatic fields, which

validates the effectiveness of our information-theoretic movement scheduling algorithm. Lastly, our mechanisms incur low overhead on resource-constrained sensor motes.

Chapter 5

Aquatic Debris Surveillance

5.1 Introduction

Recently, aquatic debris has emerged to be a grave environmental issue. Figure 1.1(b) shows the debris from Japan tsunami arriving at U.S. West Coast, 2012. Inland waters also face severe threats from debris. Over 15 scenic lakes in New Jersey still suffer debris resulted from Hurricane Sandy after one year of cleaning [40]. The debris fields pose numerous potential risks to aquatic ecosystems, marine life, human health, and water transport. For example, the debris has led to a loss of up to 4 to 10 million crabs a year in Louisiana [101], and caused damages like propeller entanglement to 58% fishing boats in an Oregon port [13]. It is thus imperative to monitor the debris arrivals and alarm the authorities to take preventive actions for the potential risks.

This chapter presents SOAR (SmartphOne-based Aquatic Robot), a low-cost vision-based debris surveillance robot system that integrates an off-the-shelf smartphone and a robotic fish platform. Figure 1.3 shows a prototype of SOAR built with a gliding robotic fish developed in our previous work [31] and a Samsung Galaxy Nexus smartphone. Various salient advantages of smartphone and gliding robotic fish make the integration of them a promising platform for debris monitoring. First, recent smartphones are powerful enough to execute advanced CV algorithms to process the images captured by the camera to detect debris objects. Meanwhile, the price of smartphones has been dropping drastically in the last five years. Many low-end Android phones (e.g., LG Optimus Net with 800 MHz CPU and 2 GB memory) cost only about $100 [52]. Second,

besides visual sensing, various built-in sensing modalities such as GPS and accelerometer can be used to facilitate the navigation and control of the robot and enable situation awareness to improve the debris detection performance. Third, the long-range communication capability of smartphone makes it possible to leverage the cloud to increase robot's intelligence and reduce energy consumption by offloading intensive computation. Lastly, as a commercial off-the-shelf platform, smartphone provides an integrated sensing system and diverse system configurations, which can meet the requirements of a wide spectrum of embedded applications. Moreover, it offers user-friendly programming environment and extensive library support, which greatly accelerates the development process. The gliding robotic fish, which is a low-cost aquatic mobile platform with high maneuverability in rotation and orientation maintenance, provides SOAR the mobility to adapt to the dynamics of debris and water environment. Owing to these features, SOAR represents an unprecedented *vision*-based, *cloud*-enabled, *low-cost*, yet *intelligent* aquatic mobile sensing platform for debris monitoring.

However, the design of SOAR still faces several unique challenges associated with aquatic debris monitoring. First, due to the impact of waves, SOAR cannot acquire a stable camera view, thereby making it highly difficult to reliably recognize the debris objects. A possible solution is image registration [42] that aligns multiple images into a common coordinate system. However, water environment often lack detectable features such as sharp corners that are commonly used for image registration. Second, SOAR is powered by small batteries due to the constraints on the form factor and cost budget, while both aquatic movement of the robot and image processing on the smartphone incur high energy consumption. Lastly, debris arrivals are often sporadic in a large geographic region [9, 19], making them highly difficult to be captured using smartphone cameras that typically have limited field of view.

This chapter makes the following contributions:

1) We develop several lightweight CV algorithms to address the inherent dynamics in aquatic debris detection, which include an image registration algorithm for extracting the horizon line above water and using it to register the images to mitigate the impact of camera shaking, and an adaptive background subtraction algorithm for reliable detection of debris objects.

2) We propose a novel approach to dynamically offloading the computation-intensive CV tasks to the cloud. The offloading decisions are made to minimize the system energy consumption based on *in situ* measurements of wireless link speed and robot acceleration.

3) We analyze the coverage for sporadic and uncertain debris arrivals based on geometric models. Using the analytical debris arriving probability, we design a robot rotation scheduling algorithm that minimizes the movement energy consumption while maintaining a desired level of debris coverage performance.

5.2 Overview of SOAR

SOAR consists of an off-the-shelf Android smartphone and a gliding robotic fish. The smartphone is loaded with an app that implements the CV, movement scheduling, and cloud communication algorithms. The gliding robotic fish is capable of moving in water by beating its tail that is driven by a servo motor. The motor is manipulated by a programmable control board, which can communicate with the smartphone through either a USB cable or short-range wireless links such as ZigBee. Various closed-loop motion control algorithms based on smartphone's built-in inertial sensor readings can be implemented on either fish control board or smartphone.

SOAR is designed to operate on water surface and monitor floating debris in nearshore aquatic environment, such as public recreational beaches, where wireless (cellular or WiFi) coverage is

available. We focus on monitoring static or slow-moving on-water objects, and filter out other objects such as boats and swimmers based on the estimated speed. When a long shoreline needs to be monitored, multiple SOAR nodes can be deployed dispersedly to form barrier coverage. In this case, the number of needed nodes is the ratio of the length of the monitored shoreline to the coverage range of the smartphone's built-in camera. In this chapter, we focus on the design of debris detection and mobility scheduling algorithms running on a single SOAR node. The sensing results of multiple nodes can be sent back to a central server via the long-range communication interface on smartphones for fusion and human inspection. SOAR has a limited sensing area due to the angular view of the built-in camera on smartphone[1], which makes it difficult to capture the debris arrivals that are likely sporadic [9, 19]. Mobility can be exploited to address this challenge. The gliding robotic fish is capable of both rotating and moving forward. As a rotation can be achieved by beating the fish tail once, it consumes much less energy than moving forward that requires continuous tail beats. Thus, this chapter exploits the rotation mobility of the robotic fish and assumes that the SOAR remains relatively stationary in water. In still water or slow water current, feedback motion control can maintain SOAR's station. In the presence of fast water current, an anchor can be used together with motion control to reduce the energy consumption for maintaining station.

After the initial deployment, SOAR begins a debris surveillance process consisting of multiple *monitoring round*s. In each round, SOAR executes a rotation schedule, which includes the camera orientation and an associated monitoring time interval. Specifically, SOAR rotates to the desired orientation at the beginning of a round and starts to take images at a certain rate, which is determined by a sleep/wake duty cycle. For each image, SOAR uses several CV algorithms to detect the

[1]Extra optical components like fisheye lens can be used to broaden the camera view. However, the integration of these components to SOAR will complicate the system design. In particular, additional complex and energy-consuming image processing algorithms (e.g., distortion rectification) are often needed.

existence of debris objects in real time. Between two image captures, SOAR sleeps to save energy. At the end of a round, SOAR computes the rotation schedule for the next round based on the detection results to ensure a desired level of debris coverage. SOAR is designed to achieve long-term (up to a few months) autonomous debris monitoring. In addition to duty cycling, it adopts a novel offloading approach to leveraging the cloud for battery power conservation. Specifically, SOAR comprises the following two major information processing components.

First, it features real-time debris detection. This component aims to extract debris objects from the taken images. It consists of three lightweight image processing modules, i.e., *image registration*, *background subtraction*, and *debris identification*, which can effectively deal with various environment and system dynamics such as shaking and internal noise of the camera. Specifically, SOAR first registers each frame by exploiting the unique features in aquatic environment, e.g., the coastline for inland waters and the horizon line for marine scenarios. Then, background subtraction in HSV color space is performed on the registered frame to identify the foreground pixel candidates. Finally, the foreground is passed to debris identification for noise removal and debris recognition. At runtime, SOAR minimizes the battery power consumption by determining if the above image processing tasks should be locally executed or entirely/partially offloaded to the cloud depending on the current network condition, e.g., the cellular network availability and link speed.

Second, it features coverage-based rotation scheduling. On the completion of a monitoring round, SOAR analyzes the debris coverage performance based on the estimated debris movement orientation and the surveillance history. It then adaptively configures the camera orientation and monitoring time interval for the next round. Because of the limited energy supply and power-consuming movement in water environment, SOAR must efficiently adjust its orientation while maintaining a desired level of debris coverage performance. To this end, we propose a scheduling

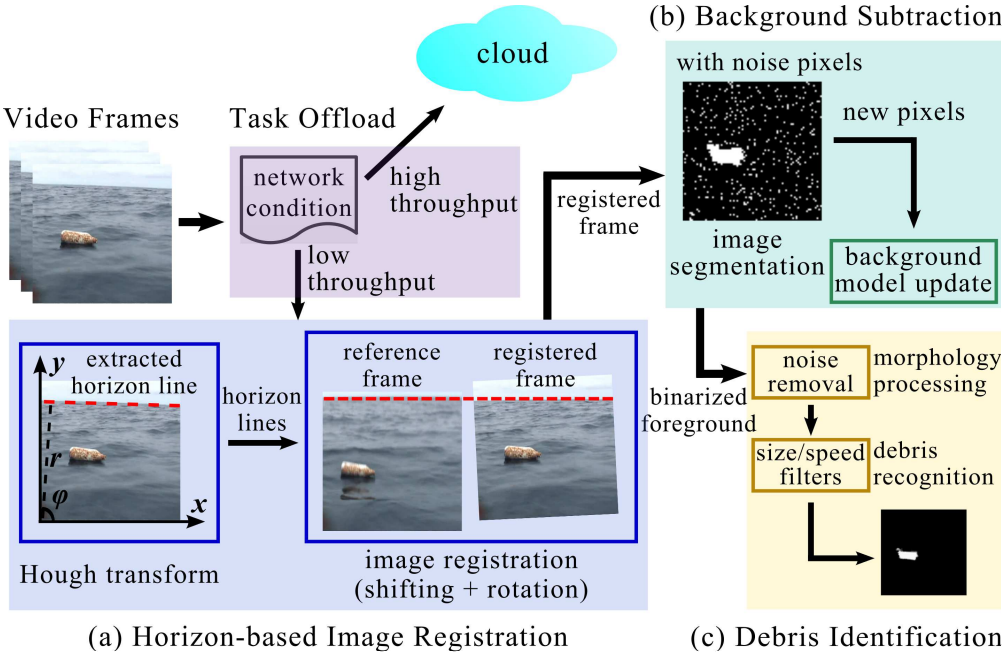


Figure 5.1: Real-time debris objects detection.

algorithm that minimizes the rotation energy consumption in a round by dynamically configuring the rotation schedule, subject to a specified upper bound on miss coverage rate for debris arrivals.

5.3 Real-Time Debris Detection

The image processing pipeline of SOAR is illustrated in Figure 5.1. Although it is based on a collection of elementary CV algorithms, it is non-trivial to optimize these computation-intensive synergistic algorithms for smartphones given the limited resources and stringent requirement on system lifetime. Specifically, SOAR consists of the following image processing components. The *image registration* aligns consecutive frames to mitigate the impact of camera shaking caused by waves. In this chapter, we focus on the marine environment as an example, although our techniques can adapt to other environment. In a marine scenario, the horizon line can be used as a reference to register the frames. To deal with the high computation overhead of horizon line extraction, SOAR offloads a portion of computation to the cloud based on the network link speed and shaking levels

indicated by the smartphone's inertial sensor readings. The registered frames are then compared with a background model to extract the foreground. Lastly, the *debris identification* removes the salt-and-pepper noises from the foreground image and then identifies the debris objects.

5.3.1 Horizon-Based Image Registration

Image registration is the process of aligning images taken at different time instants into one coordinate system [42]. In debris detection, image registration is necessary to mitigate the impact of camera shaking caused by waves, such that subsequent pixel-wise processing can be executed properly. Registration is performed by establishing correspondence between images based on their distinguishable features. However, a key challenge is that there are few detectable image features in typical water environment that can guide the image registration. A novelty of our approach is to leverage the horizon line, which segments the sky and water areas, for image registration, as shown in Figure 5.1(a).

We employ Hough transform [41] to extract the horizon line. Hough transform has been widely used to identify the positions of lines in an image. We assume that the majority of an image is either sky or water area, which are separated by the horizon line. As the sky and water areas typically have distinct colors [19], Hough transform is able to extract the horizon line accurately. For each frame, we first convert it to a grayscale image and detect edges using a Sobel operator [42]. The Sobel operator detects an edge point according to the local intensity of gradient magnitude. Hough transform then finds the horizon line through a voting process based on the number of edge points that each candidate line passes through. The result of Hough transform is a line expressed as $r = x \cdot \cos\phi + y \cdot \sin\phi$ in the Cartesian coordinate system originated at the bottom-left corner of an image. The horizon line is parameterized by r and ϕ, which are the distance from the horizon line

to the origin and the angle between the horizon line's orthogonal direction and x-axis, respectively. An illustration of the extracted horizon line is shown in Figure 5.1(a).

Based on the extracted horizon line, we register each video frame to mitigate the impact of camera shaking. Specifically, for two consecutive frames, we register the successor frame according to the registered predecessor by aligning their extracted horizon lines. Let Δy denote the vertical shifting at the midpoint of the horizon lines, and η denote the angle that the horizon line rotates in these two frames. We assume that the midpoints of the horizon lines in the two frames correspond to the same point in the world plane. Such an assumption is motivated by the observation that the closed-loop motion control algorithms can maintain the robot's orientation and position by adaptive course-correction. Therefore, Δy and η define the *affine transform* between these two consecutive frames. First, we shift the successor frame to align the midpoint of its horizon line with that of the registered predecessor frame by $x' = x$ and $y' = y + \Delta y$, where (x, y) are the coordinates of a pixel in the unregistered frame, and (x', y') are the corresponding coordinates of (x, y) after shifting. Then, we rotate the frame by

$$\begin{bmatrix} x_r \\ y_r \end{bmatrix} = \begin{bmatrix} \cos\eta & \sin\eta & x_0(1-\cos\eta) - y_0\sin\eta \\ -\sin\eta & \cos\eta & x_0\sin\eta - y_0(1-\cos\eta) \end{bmatrix} \cdot \begin{bmatrix} x' & y' & 1 \end{bmatrix}^\top,$$

where (x_r, y_r) denote the coordinates of the underlying pixel in the reference frame, and (x_0, y_0) denote the coordinates of horizon line midpoint in the successor frame after shifting and serve as the center of rotation. For those pixels without their correspondents in the original unregistered frame due to the rotation, we adopt bilinear interpolation to fill the color data (i.e., RGB) for them.

5.3.2 Background Subtraction

To reduce the energy consumption in image processing, we adopt a lightweight background subtraction approach to detecting the foreground debris objects. We first convert the representation of an image to HSV (Hue, Saturation, and Value) model. In HSV, hue represents the color, saturation is the dominance of hue, and value indicates the lightness of the color. The HSV representation is robust to illumination changes and hence more effective in interpreting color features in the presence of reflection in water environment.

The background of a pixel is represented by a Gaussian mixture model (GMM) [42]. The GMM comprises K three-dimensional Gaussians, where each Gaussian characterizes the three channels of HSV color space. When a new frame is available, each pixel is compared with its background model. If the HSV vector of a pixel in the new frame does not fall within a certain range from any mean vector of the K Gaussians, this pixel is considered a foreground pixel candidate; otherwise, it is classified as background. Therefore, the color difference between the foreground and background affects the classification accuracy. In our implementation, the range is chosen to be 2.5 times of the standard deviation of the corresponding Gaussian. Under this setting, the image segmentation can tolerate minor inaccuracy introduced by noises and accommodate certain environmental condition changes.

Any labeled pixel will be used to update the GMM. In GMM, each Gaussian has a weight that characterizes the fraction of historical pixels supporting it. If none of the existing K Gaussians match this pixel (i.e., the pixel has been classified as foreground), the distribution with the lowest weight is replaced by a new Gaussian, which is assigned with a large variance, a small weight, and a mean vector equal to this newly arrived pixel; otherwise, the weight, mean, and variance vectors of the matched Gaussian are updated based on the new pixel value and a user-specified learning

rate. The details of GMM update can be found in [94]. This scheme allows the GMM to adapt to environmental condition changes, enhancing the robustness of background subtraction.

We now discuss the impact of the number of Gaussians (i.e., K) in the GMM on background substraction performance. As shown in [94], the background model with $K = 1$ can only deal with static background. Therefore, more Gaussians are needed in aquatic debris detection to account for the dynamics from camera shaking, reflection, and noises. A larger K can enrich the information maintained by the GMM and hence improve its robustness. However, it also imposes additional computation overhead for the smartphone. In Section 5.6.1.3, we will evaluate the trade-off between system overhead and detection performance through experiments, which guides the setting of K. We note that it may require a large K to describe the environment with more complex and dynamic background. Our approach can be easily extended to employ existing online algorithms that maintain a GMM with variable K adaptive to background changes, such as the reversible jump Markov chain Monte Carlo method [80] and its variant [95] that has been used for video processing.

5.3.3 Debris Identification

The binarized foreground image often contains randomly distributed noise pixels, as depicted in Figure 5.1(b). Because the background subtraction is conducted in a pixel-wise manner, the labeling of foreground and background can be affected by camera noise, resulting in false foreground pixels. To deal with these noise pixels, we adopt the opening operation in image morphology [42]. The opening operation, which consists of erosion followed by dilation, eliminates the noise pixels through erosion while preserving the true foreground by dilation. After the noise removal, we employ region growing to identify the debris objects from the foreground image. It uses the fore-

ground pixels as the initial seed points and forms connected regions that represent candidate debris objects by merging nearby foreground pixels.

We note that the extracted foreground candidate objects may contain objects that can move actively, e.g., boats and swimmers. SOAR adopts a threshold-based approach to filter out these non-debris objects. Specifically, the robot estimates the object movement speed based on the pinhole camera projection model (see Section 5.4.4). When the estimated speed is higher than a threshold, SOAR will save the image and periodically transmit back to a central server for human inspection. The threshold on speed is chosen to exclude the non-debris objects with active mobility. Similar heuristics based on object moving orientations can be applied to improve the accuracy of debris recognition. We use another threshold-based method to remove the small objects. The small object size in the image usually indicates a false alarm or a distant debris object. Ignoring them does not affect the system accuracy since distant real debris objects will likely be detected when they approach closer to the camera. Note that the shape of debris object has little impact on the debris detection performance, as the false negatives in detection are mainly caused by the high color similarity between foreground and background and the long distance between object and camera.

5.3.4 Dynamic Task Offloading

A key advantage of smartphone-based robots lies in their capability of leveraging the abundant resources of the cloud. To prolong the smartphone battery lifetime, SOAR dynamically offloads the entire/partial image processing to the cloud when there is network coverage. As the typical coverage of a cellular tower is up to several miles [89], cellular networks can be available in nearshore water surface. The offloading decision mainly depends on two factors: 1) the overhead of image processing algorithms, e.g., the power consumption, when they are executed locally;

and 2) the wireless network condition, e.g., the uploading speed, which determines the energy consumption for uploading images to the cloud. In our design, SOAR has three offloading schemes, i.e., *cloud*, *local*, and *hybrid* processing, and dynamically chooses a scheme with the lowest energy consumption. The energy consumption of these schemes are analyzed as follows.

We first analyze the energy consumption of the *local* and *cloud* processing schemes, where all the processing on the original frame is conducted either in the cloud or on the phone. When a new frame is available, SOAR checks the network (cellular or WiFi) link speed and estimates the delay to upload this frame. The energy consumption for uploading the entire image is given by $e_{\text{cloud}} = p_c(\gamma) \cdot s/\gamma$, where s is the frame size, γ is the measured link speed, and $p_c(\gamma)$ is the uploading power consumption under the link speed γ. Alternatively, the frame can be processed on the phone. Let t_l denote the delay to process a frame locally, including *image registration*, *background subtraction*, and *debris identification*. Our measurements show that these modules have similar CPU power consumption, which is denoted by p_l. The energy drain for processing a frame on the phone is thus given by $e_{\text{local}} = p_l \cdot t_l$.

In addition to the above two options, we propose a hybrid solution that offloads the Hough transform in *image registration* to the cloud and conducts the rest of processing locally. Such a design is motivated by the observation that the Hough transform incurs nearly 70% processing overhead for a frame (see Section 5.6.1.1). Moreover, as the energy consumption for offloading is largely determined by the uploading volume, we propose to upload a rectangular part of the original frame, which contains the horizon line. As the camera is shaking, the selection is based on the horizon line in the reference frame and the accelerometer readings. Specifically, we adopt the accumulated linear vertical acceleration (denoted by $\sum a_z$) over the time duration from the predecessor reference frame to the current frame to quantify the camera shaking. In theory, if $\sum a_z < 0$, the horizon line will shift upward in the current frame; otherwise, it will shift downward.

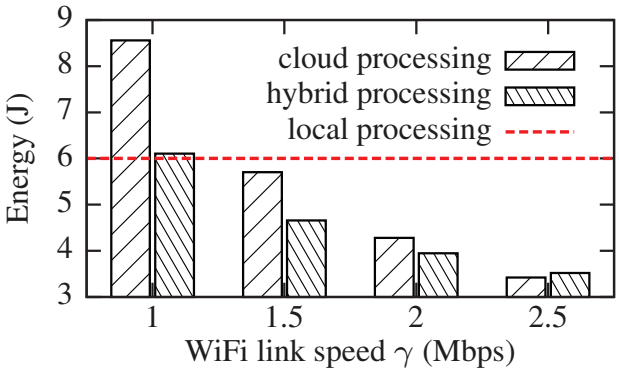


Figure 5.2: Energy consumption per frame on Samsung Galaxy Nexus.

Therefore, from the sign of $\sum a_z$, we can estimate whether the horizon line in the current frame is in the upper or lower part divided by the horizon line in the reference frame. We verify this hypothesis using 20 video sequences with each consisting of 30 frames. The results show that this hypothesis holds for 576 frames out of all 600 frames. In our hybrid scheme, the rectangular part to be uploaded has the original frame width and a height (denoted by h_c) from the horizon line midpoint in the unregistered reference frame to either width depending on the sign of $\sum a_z$. Let h_0 denote the original frame height, and t_h denote the delay to conduct Hough transform locally. The energy consumption for hybrid processing is given by $e_{\text{hybrid}} = \frac{h_c}{h_0} e_{\text{cloud}} + \frac{t_l - t_h}{t_l} e_{\text{local}}$. Note that this formula ignores the low-probability cases where the above hypothesis does not hold. In these cases, the cloud will fail to identify the horizon line and the original frame will be processed locally.

By comparing e_{local}, e_{hybrid}, and e_{cloud}, SOAR chooses a scheme with the lowest energy consumption. All the parameters except the height h_c in hybrid scheme and the link speed γ can be obtained using offline measurements. In our current prototype, we use WiFi to upload video frames to the cloud, although the implementation can be easily extended to cellular network. We measure the power consumption of a Samsung Galaxy Nexus using an Agilent 34411A multimeter when the smartphone is uploading video frames under various WiFi link speed settings and

locally processing the frames. Figure 5.2 plots the energy consumption per frame under the three schemes. Note that for the hybrid scheme, we set $h_c/h_0 = 50\%$. We can observe that when the link speed is high, it is preferable to offload the entire/partial image processing to the cloud for energy conservation.

5.4 Coverage-Based Rotation Scheduling

In this section, we first introduce camera sensing and debris arrival models, and analyze the effectiveness of covering debris arrivals. We then present a rotation scheduling algorithm that aims to minimize the rotation energy consumption while maintaining a desired coverage rate for debris arrivals.

5.4.1 Camera and Debris Models

The *field of view* (FOV) is a widely used concept to characterize a camera's sensing area, in which any object with dimensions larger than pre-defined thresholds can be reliably detected from the taken images. A camera's FOV is typically represented by a sector originated at the camera's pinhole with an angular view α and a radius R [42], where α is hardware-dependent and R can be measured through experiments for a specific application. For example, in pedestrian detection, the camera's FOV has a radius of up to 40 meters [28]. For the objects within the FOV, image sensing is insensitive to their dimensions as long as they are larger than a certain size. For example, our on-water experiments show that any floating object with a cross-sectional area over $0.28\,\mathrm{m}^2$ (e.g., a five gallon water bottle) can be reliably detected at 40 meters away from a smartphone's camera. We note that FOV is a conservative approximation to the camera's sensing area because the objects outside FOV may also be detected by the camera. This approximation simplifies the analysis of the

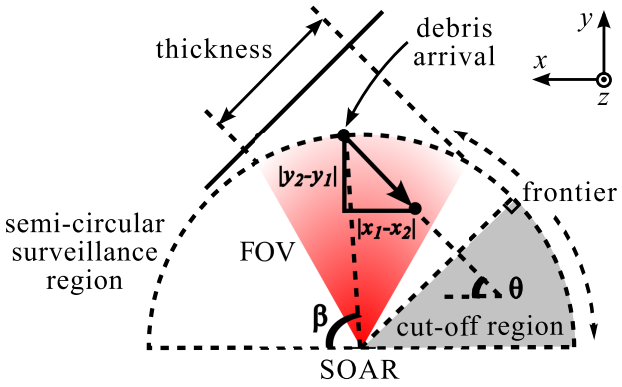


Figure 5.3: Illustration of FOV, surveillance region, thickness, debris arriving angle β, debris movement orientation θ and its estimation, and the cut-off region.

coverage of debris arrivals. In Section 5.4.2, we will show that the rotation scheduling algorithm does not depend on the value of R.

Since we focus on the nearshore debris arrival monitoring, the *surveillance region* for SOAR is defined as the semi-circular area originated at the robot with a radius of R. We define the camera orientation by its optical axis. Because of the limited angular coverage of FOV, SOAR needs to constantly adjust its orientation to maintain a desired coverage level for debris arrivals over the defined surveillance region. In this chapter, we assume that the adjustments of camera orientation only occur at time instants that are multiples of a fixed time interval unit, which is referred to as *slot*.

In the nearshore water environment, debris objects often passively drift with water current that moves straightly in a fixed direction [19]. We set up a Cartesian coordinate system with x-axis, y-axis, and z-axis, which are parallel, horizontally perpendicular, and vertically perpendicular to the shoreline, respectively. We assume that the positions of debris objects on the water surface at a given time instant follow the Poisson point process, which has been verified in previous studies [9, 19]. Our analysis (omitted due to space limit) shows that the number of debris objects that arrive at the surveillance region *frontier*, as illustrated in Figure 5.3, follows the Poisson process. A Poisson

process is characterized by an *arrival rate*, denoted by λ, and a *time interval*, denoted by τ. The λ is the expected number of arriving debris objects at the surveillance region during a slot, and τ is the number of slots. The probability that there is at least one object arriving at the surveillance region during the interval τ is

$$P_a = 1 - \mathrm{e}^{-\lambda\tau}. \tag{5.1}$$

As the occurrences of debris objects are rare events, the Poisson process has a small λ. In Section 5.4.4, we will describe an approach to estimating λ based on the historical detection results and envision a cloud-based approach that leverages the publicly available information about aquatic debris.

5.4.2 Debris Arrival Coverage

In the rotation scheduling, an arriving debris object is *covered* by SOAR if the arrival position at the surveillance region frontier falls into the camera's FOV. A rotation schedule aims to cover as many newly arriving debris objects as possible. Accordingly, our rotation scheduling algorithm controls the orientation of SOAR based on the statistics of the historical debris arrival observations. Therefore, it is important to model the arrivals of debris objects at the surveillance region. We note that whether a debris object covered by the camera's FOV can be eventually extracted from the taken images is determined by the performance of the CV algorithms (see Section 5.3). However, it is desirable to maximize the coverage rate, which improves the overall debris monitoring performance. We define the coverage rate for debris arrivals provided by SOAR as the ratio of the number of trajectories hitting camera's FOV frontier to that hitting surveillance region frontier. However, one immediate challenge is that lines are uncountable. We employ *thickness* in geometric probability [88] to measure the set of trajectories. Specifically, the thickness of an arc

frontier, denoted by $\mathscr{T}$, is defined as the length of its projection to the line perpendicular to debris movement orientation, as illustrated in Figure 5.3.

The thickness allows us to directly evaluate the debris coverage performance. We first analyze the probability of *miss coverage* based on thickness. Let θ denote the *debris movement orientation*, which is the angle between the debris' trajectory and x-axis. Let β denote the *debris arriving angle*, which is the angle between the radius from debris' arrival position at the surveillance region frontier and x-axis. Both θ and β are illustrated in Figure 5.3. To simplify the discussion, we discretize the possible arriving angles by 1°, and define *arrival frontier* as the 1° arc centered at the arrival position. Let $\mathscr{T}(\beta)$ and $\mathscr{T}_0$ denote the thickness of arrival frontier and the thickness of the whole surveillance region frontier, respectively. Based on the geometric relationship, we have $\mathscr{T}(\beta) = 2R\sin 0.5^\circ \cos(\beta-\theta)$ and $\mathscr{T}_0 = (1+\sin\theta)R$. Therefore, the probability that a debris object hits the arrival frontier defined by β conditioned that its trajectory intersects with the surveillance region frontier, is given by $g(\beta,\theta) = \mathscr{T}(\beta)/\mathscr{T}_0$, which is independent of R. As the probability that at least one debris object arrives at the surveillance region is P_a given in Equation (5.1), the probability that at least one debris object reaches the arrival frontier defined by β (denoted by $P_a(\beta)$) is $P_a(\beta) = P_a \cdot \mathscr{T}(\beta)/\mathscr{T}_0 = P_a \cdot g(\beta,\theta)$. We note that if β is not covered by the FOV during an interval τ, $P_a(\beta)$ represents the probability of missing debris arrivals. We then derive the miss coverage rate, which will be used to schedule the rotation of SOAR. Let $t_r(\beta)$ denote the end time instant of the most recent slot when β was covered by the camera's FOV, and t denote the end time instant of interval $[t_r(\beta),t]$, during which the arrival frontier defined by β remains uncovered. The *miss coverage rate* at arriving angle β and time t, denoted by $\omega(\beta,t\,|\,t_r(\beta),\theta)$, is defined as the probability of missing debris arrivals during interval $[t_r(\beta),t]$.

Formally,

$$\omega(\beta, t \,|\, t_r(\beta), \theta) = P_a(\beta) \,|_{\,\tau = t - t_r(\beta)} = (1 - \exp(-\lambda\,(t - t_r(\beta)))) \cdot g(\beta, \theta). \tag{5.2}$$

Note that the miss coverage rate does not depend on the specific value of R, but the ratio of thicknesses, i.e., $g(\beta, \theta)$. Thus, the coverage-based rotation scheduling algorithm presented in Section 5.4.3 will not rely on R.

5.4.3 Coverage-Based Rotation Scheduling

Because of the limited angular coverage and energy budget, the rotation of SOAR needs to be carefully scheduled to achieve the desired coverage rate for debris arrivals. In this section, we formulate the problem of coverage-based robot rotation scheduling. Our objective is to schedule the next camera orientation and corresponding monitoring interval to minimize the overall energy consumption, subject to an upper-bounded miss coverage rate at all arrival angles.

As the rotation energy consumption is approximately proportional to orientation change [51], we adopt the changed orientation to characterize the consumed energy in rotation. Let $\widetilde{\theta}$ denote the estimated debris movement orientation (see Section 5.4.4). Suppose the current camera orientation and time instant are β_0' and t'. The next rotation schedule, including the camera orientation β_0 and end time instant of monitoring interval t, minimizes the average rotation rate $\overline{v}$:

$$\overline{v} = \frac{|\beta_0 - \beta_0'|}{t - t'}, \tag{5.3}$$

subject to

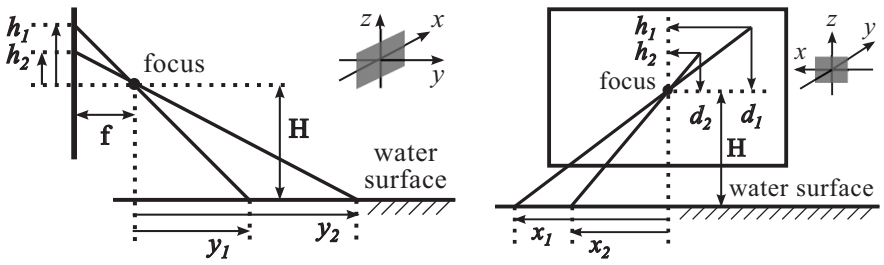
$$\omega(\beta, t \,|\, t_r(\beta), \widetilde{\theta}) < \xi, \ \forall\, \beta \in [0, \beta_0 - \alpha/2] \cup [\beta_0 + \alpha/2, \pi], \tag{5.4}$$

where ξ is the maximally tolerable miss coverage rate. The constraint in Equation (5.4) upper-bounds the miss coverage rate ω at each uncovered β. Note that for β within the camera's FOV, i.e., $\beta \in [\beta_0 - \alpha/2, \beta_0 + \alpha/2]$, it has zero miss coverage rate. The above problem can be efficiently solved by searching a set of discrete candidate orientations. During surveillance, SOAR keeps a map of ω at each β, and updates it before each new scheduling to account for system dynamics, which includes the error in orientation adjustment, the updated $\widetilde{\theta}$ (see Section 5.4.4) and λ. With the updated map of ω, SOAR adaptively schedules the next orientation and the associated monitoring interval.

Equation (5.4) ensures an upper bound on ω for uncovered arriving angles. Given a certain θ, there always exists a *cut-off region* as illustrated in Figure 5.3. Specifically, the arrival position of a debris object will never fall into the frontier of this cut-off region. To avoid the unnecessary rotation, SOAR can exclude this cut-off region. Our analysis shows that $\overline{v}$ is reduced by about 25% after excluding this cut-off region when $\theta = \pi/3$.

5.4.4 Debris Movement and Arrival Estimation

From Equation (5.2), the miss coverage rate ω depends on the debris movement orientation θ. Before deployment, SOAR can be configured with a coarsely estimated θ from prior knowledge about the water movement direction in the deployment region. Once SOAR detects a debris object, θ can be accurately estimated based on the pinhole camera projection model and the positions of the object in images. Figure 5.4 shows the pinhole projection model. Specifically, Figure 5.4(a) illustrates how real-world distance along y-axis is projected to vertical axis h in the image. It can be obtained that $|y_1 - y_2| = fH \cdot |1/h_2 - 1/h_1|$, where y_1 and y_2 are distances between SOAR and the debris object in two frames; h_1 and h_2 are the vertical pixel distance equivalents of y_1 and y_2; f is the camera focal length; and H is the mounting height of smartphone on SOAR. Similarly,



(a) Pinhole projection y-z (b) Pinhole projection x-z

Figure 5.4: Pinhole camera projection model.

Figure 5.4(b) illustrates how real-world distance along x-axis corresponds to horizontal axis d. The following relationship holds $|x_1 - x_2| = H \cdot |d_2/h_2 - d_1/h_1|$, where x_1, x_2, d_1, and d_2 are similar measures to y_1, y_2, h_1, and h_2. Based on the geometric relation shown in Figure 5.3, the estimated debris movement orientation $\widetilde{\theta}$ is given by

$$\widetilde{\theta} = \arctan\left|\frac{y_1 - y_2}{x_1 - x_2}\right| = \arctan\left(f \cdot \left|\frac{h_1 - h_2}{d_2 h_1 - d_1 h_2}\right|\right) + \theta_r, \tag{5.5}$$

where θ_r is the heading direction of SOAR and can be obtained from the built-in digital compass of smartphone. We will evaluate the accuracy of $\widetilde{\theta}$ in Section 5.6.1.2. Moreover, the object movement speed can be estimated by $(|x_1 - x_2|^2 + |y_1 - y_2|^2)^{1/2}/|t_1 - t_2|$, where t_1 and t_2 are the time instants when the object is at $(x_1, y_1, 0)$ and $(x_2, y_2, 0)$, respectively.

We then describe two approaches to estimating the debris arrival rate λ. First, λ can be estimated based on the historical detection results by SOAR. Suppose the robot detects n debris objects in the rotation schedule specified by β_0 and t. As discussed in Section 5.4.2, the probability that an arriving debris object is covered by the camera's FOV conditioned that it arrives at the surveillance region is given by the ratio of their thicknesses, i.e., $\mathscr{T}_{\text{FOV}}/\mathscr{T}_0$. Hence, the expectation of debris arrivals at the whole surveillance region during one slot, i.e., λ, can be estimated by $n\mathscr{T}_0/(P_d\mathscr{T}_{\text{FOV}}(t - t'))$, where P_d is the lower bound on detection probability for the FOV and $t - t'$

counts the monitoring slots. Second, the long-range communication capability of SOAR makes it possible to exploit the available web resources to estimate λ. For example, the Marine Debris Clearinghouse [67] is a representative online tool for aquatic debris tracking. Based on satellite images, on-site reports, and aquatic field simulations, it can estimate the intensity of incoming debris over large areas.

5.5 SOAR Prototype

We have built a proof-of-concept prototype of SOAR for evaluation. The vision-based debris detection algorithm presented in Section 5.3 is fully implemented on smartphone platforms running Android 4.3 Jelly Bean. System evaluation is mainly conducted on two representative handsets, a Samsung Galaxy Nexus (referred to as *Galaxy*) and a Google Nexus 4 (referred to as *Nexus4*). The implementation takes about 1.99 MB storage on the phone, and requires about 10.2 MB RAM when the frame resolution is set to be 720×480. When a new frame is available, SOAR checks the current WiFi condition using Android API `WifiInfo.getLinkSpeed()`. Based on the measured link speed and horizon line position, it determines whether this newly arrived frame is locally processed or entirely/partially uploaded to the cloud, following the scheme proposed in Section 5.3.4. Our initial implementation of image processing modules in Java incurs extensive delays on current Android system. To boost the frame processing performance, we use OpenCV libraries and interface them with Java using Java Native Interface on Android. In particular, we adopt OpenCV's native implementation of the Hough transform, which is more efficient than other implementations in our tests.

We integrate the smartphone to a gliding robotic fish developed in our previous work. The fish platform weights 9 kilogram and represents a hybrid of underwater gliders and robotic fish with

advantages of both. It is equipped with a ZigBee radio for wireless communication, two 75 W · h on-board batteries, and a circuit board for mobility control, localization, and navigation. On the control board, we implement a closed-loop proportional-integral-derivative (PID) controller that adaptively configures the tail beat based on the discrepancy between the scheduled and actual orientations. In our current implementation, we use a host computer to relay the communication between the smartphone (using WiFi) and the fish (using ZigBee). In the future, we will establish direct connection between them, and migrate the PID controller to the smartphone to reduce the physical size and cost of the fish control board. Figure 1.3 depicts our prototype system that integrates a Galaxy in a water-proof enclosure with a gliding robotic fish.

5.6 Performance Evaluation

We evaluate SOAR through testbed experiments and simulations based on data traces collected from the prototype. The testbed experiments validate the feasibility of SOAR by evaluating the computation overhead, effectiveness of each module, and the overall performance of a fully integrated SOAR. The simulations extensively evaluate the performance of SOAR working under wide ranges for parameter settings.

5.6.1 Testbed Experiments

We evaluate SOAR in controlled lab experiments to fully understand its performance. We conduct extensive experiments using our prototype in a 15 feet $\times$ 10 feet water tank in our lab. Along one side of the tank, we vertically place a piece of white foam board above the water surface to imitate the sky area. The line where the foam board intersects with the water surface produces an artificial horizon line. In the experiments, we set the camera frame rate to be 0.25 fps. The setting is based on

the fact that debris usually has a slow drifting speed and hence does not require a rapidly updated background model. Moreover, a low frame rate helps prolong the battery lifetime. The GMM comprises 3 three-dimensional Gaussians (i.e., $K = 3$), unless otherwise specified. We test the debris detection performance under various environmental conditions and experimental settings, which include different camera shaking levels, with and without registration, and different settings of GMM. For each scenario, we conduct 9 runs of experiments. For each run, we calculate the detection probability as the ratio of frames with correct detections to the total frames with debris, and the false alarm rate as the ratio of frames with false detections to the total frames without debris.

5.6.1.1 Overhead on Smartphone Hardware

We first evaluate the overhead of vision-based detection algorithms on smartphone platforms. Specifically, we measure the computation delay of each image processing module, i.e., *image registration*, *background subtraction*, and *debris identification* on Galaxy and Nexus4, respectively. Galaxy has an 1.2 GHz dual-core processor and 1 GB memory, and Nexus4 has an 1.5 GHz quad-core processor and 2 GB memory. They are representative mid- and high-end mainstream smartphone platforms. The computation delay is measured as the elapsed time using Android API `System.nanoTime()`. The results are plotted in Figure 5.5. We can see that *background subtraction* takes the least time, followed by *debris identification* combined with debris movement orientation estimation. *Image registration* incurs the longest delay. Breakdown shows that this long delay is mostly caused by the Hough transform. The overall computation delay is within 3.7 and 3.3 seconds on Galaxy and Nexus4, respectively, which well meet the real-time requirement of debris monitoring as debris arrivals are typically sporadic [9, 19].

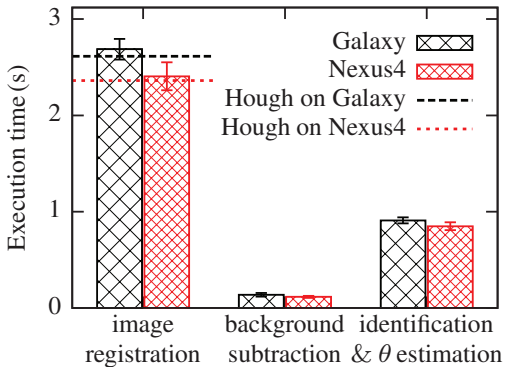


Figure 5.5: Execution time of image processing modules on smartphone platforms.

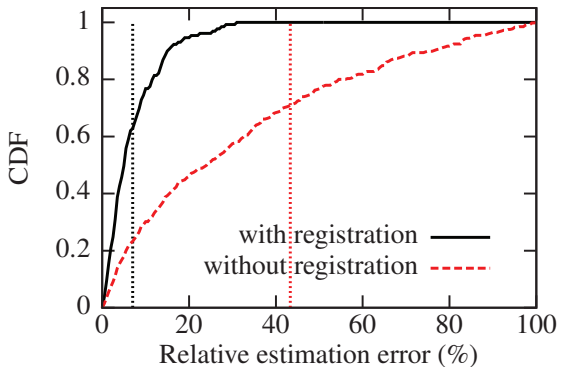


Figure 5.6: CDF and average of relative estimation error for θ.

5.6.1.2 Accuracy of Debris Movement Orientation Estimation

We then evaluate the debris movement orientation estimation presented in Section 5.4.4. Initially, the debris movement orientation θ is unknown to SOAR. After a debris object is successfully detected in two frames, SOAR can estimate θ based on Equation (5.5). In the experiments, an object (a can) is fastened to a rope. We drag the object using the rope to simulate the movement of debris object. We define the *relative estimation error* as $|\theta - \widetilde{\theta}|/\theta$, where $\widetilde{\theta}$ is the estimated orientation from Equation (5.5) and θ is the ground truth obtained by using a protractor. Figure 5.6 plots the CDF and the average of the relative estimation error for the approaches with and without image registration. We can see that the approach with registration can accurately estimate θ, with an average relative estimation error of only about 7%. In contrast, the approach without registration results in an average relative estimation error of around 43%.

5.6.1.3 Impact of Mixed Gaussians

We now evaluate the impact of the number of mixed Gaussians (i.e., K) on detection performance. Section 5.3.2 discusses the trade-off between the detection performance and system overhead caused by K. The detection probability and execution time on Galaxy are plotted in Figure 5.7.

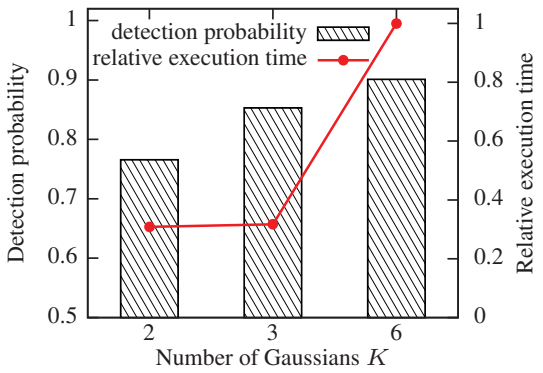


Figure 5.7: Impact of the number of mixed Gaussians K.

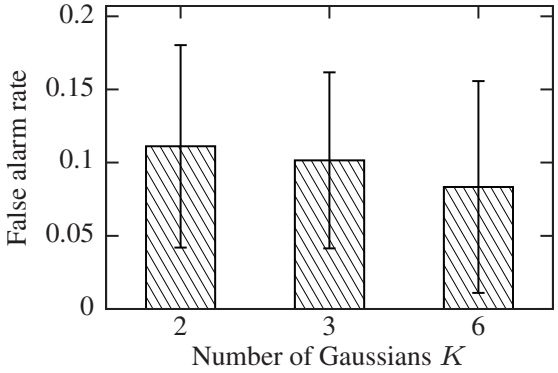
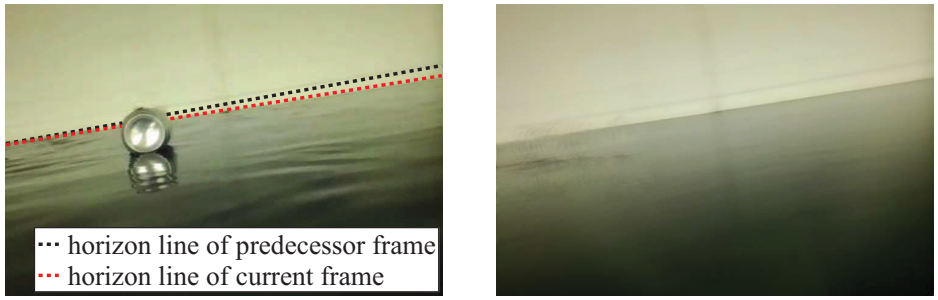


Figure 5.8: False alarm rate versus number of mixed Gaussians K.

We can see that the detection probability increases with K. Moreover, Figure 5.8 evaluates the false alarm rate versus K, where the error bar represents the standard deviation. It can be observed that the false alarm rate decreases with K. A larger K imposes heavier computation overhead in both image segmentation and background model update. When the GMM adopts 2 Gaussians, the computation delay is about 30% of that with 6 Gaussians. We also find that $K = 3$ achieves a satisfactory trade-off between detection performance and computation delay. Therefore, we set $K = 3$ in other testbed experiments.

5.6.1.4 Effectiveness of Image Registration

As discussed in Section 5.3.2, the background subtraction is conducted in a pixel-wise manner, hence its performance is sensitive to camera shaking. Figure 5.9 shows a sample of background subtraction. Specifically, Figure 5.9(a) is the original frame where the red and black dashed lines represent the extracted horizon lines for this frame and the registered predecessor frame, respectively. Figure 5.9(b) shows the background model, where each pixel is the mean vector of the Gaussian with the largest weight in the GMM. Figure 5.9(c) is the result of background subtraction without image registration. Figure 5.9(d) is the result with image registration before subtraction. We can see that our horizon-based registration effectively mitigates the impact of camera shaking,



(a) Original frame (b) Gaussian with the largest weight

(c) Without registration (d) With registration

Figure 5.9: Sample background subtraction outputs for approaches with and without horizon-based image registration.

and hence the detection algorithm can more accurately pinpoint the foreground object location in the image. Figure 5.10 plots the detection probability for approaches with and without image registration under different camera shaking levels. In the experiments, we generate different levels of waves by controlling a feed pump connected to the tank. The $\overline{a}_z$ reported in Figure 5.10 is the average linear vertical acceleration (i.e., excluding gravity) measured by the built-in accelerometer on smartphone, and hence characterizes the camera shaking levels. We can see that the image registration not only improves the average detection performance, but also decreases the variance in detection probability in the presence of camera shaking. It effectively mitigates the impact of shaking, leading to a smaller degradation in detection probability as camera shaking increases.

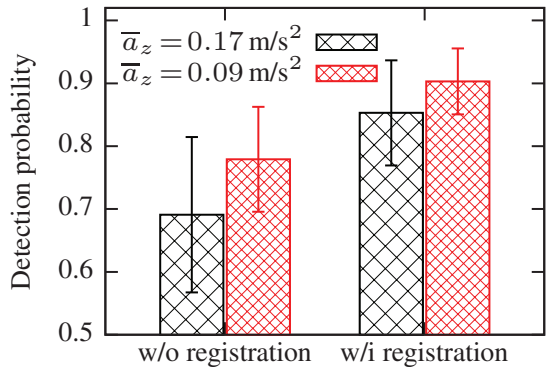


Figure 5.10: Impact of image registration and shaking level.

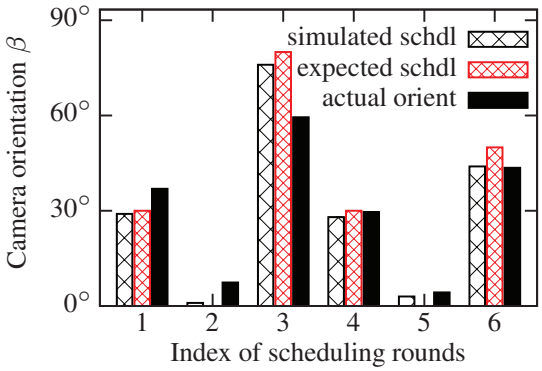


Figure 5.11: Simulated, expected, and actual orientations in the integrated evaluation.

5.6.1.5 Integrated Evaluation

In this set of experiments, all modules of SOAR (vision-based debris detection, θ estimation, rotation scheduling, and PID-controlled orientation adjustment) are integrated and evaluated. Similar to previous experiments, we drag a can to simulate a debris object. The debris movement orientation θ is 0.156π, which is unknown to the system before deployment. We set the slot duration as 1 minute. At time $t = 0$, SOAR is deployed perpendicularly to the tank length. At time $t = 1$ min, it starts the first monitoring round as discussed in Section 5.2. In this experiment, SOAR achieves 83.3% detection probability and 5.8% average relative θ estimation error. We further study the performance of rotation scheduling and orientation adjustment of SOAR. It is unlikely for SOAR to rotate to the exact scheduled orientation due to complex fluid dynamics and compass inaccuracy. This orientation adjustment error and minor inaccuracy in θ estimation affect the rotation scheduling for the next round. For evaluation, we compare the actual rotations of SOAR with the real-time scheduled rotations during this experiment (referred to as *expected schedule*), and those in an ideal simulation (referred to as *simulated schedule*) where we assume both orientation adjustment and θ estimation are accurate and thus feed the scheduling algorithm with ground truth. We note that the differences between the actual rotations and expected schedule characterize the

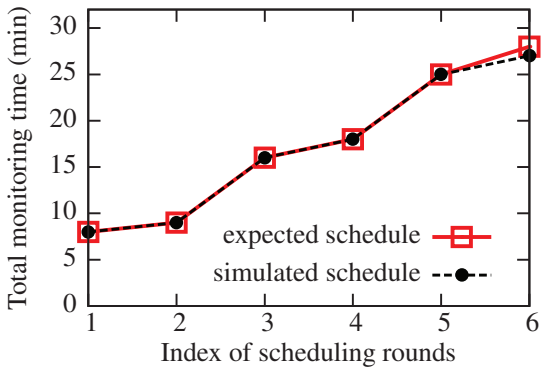


Figure 5.12: Simulated and actual monitoring intervals in the integrated evaluation.

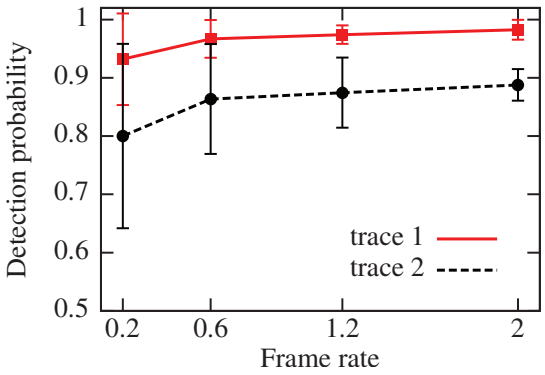


Figure 5.13: Impact of frame rate on debris detection probability.

performance of the PID-controlled orientation adjustment. The deviations between the expected schedule and simulated schedule indicate the robustness of the rotation scheduling algorithm to the control and estimation errors. The results are shown in Figure 5.11 and Figure 5.12. We find that the orientation adjustment errors vary for different expected orientations due to different levels of fluid obstruction. For example, in the 3rd scheduling round, SOAR is subject to a higher level of fluid obstruction when it targets an expected orientation perpendicular to the water movement direction. Our PID controller can generally maintain an orientation adjustment error lower than 15°. Moreover, as shown in Figure 5.12, the expected monitoring intervals well follow the simulated schedule because the temporal scheduling is mainly determined by the debris arriving intensity over time.

5.6.2 Trace-Driven Simulations

To more extensively evaluate SOAR under realistic settings, we conduct simulations driven by real data traces collected using our prototype in the water tank. The data include error traces of SOAR orientation adjustments and video traces of debris arrivals. First, the error traces of SOAR orientation adjustments are collected using our prototype system. The camera orientation

is represented by the heading direction of SOAR. We collect the error traces by measuring the discrepancy between the desired orientation and actual heading direction. Second, we use our prototype to collect two distinct sets of video traces of an arriving bottle. In our prototype, the phone is mounted about 3 inches above the water surface. Trace 1 is collected in calm environment, and has 1495 frames in total. Trace 2, with a total of 1275 frames, is collected in the presence of persistent waves generated by the feed pump. To provide ground truth data, the foreground debris object in each frame is manually labeled.

In the simulations, SOAR is deployed to monitor debris objects arriving in a semi-circular surveillance region. The arrival rate λ is set to be 9, unless otherwise specified. The debris movement orientation is $\theta = \pi/3$, and we assume that θ is known to the robot. We set the FOV angular coverage α to be $5\pi/18$ based on our measurements of Galaxy. Initially, the robot is deployed perpendicular to the shoreline. It is allowed to adjust its orientation after the first slot. For each orientation adjustment, the actual direction is set according to the collected traces, which is thus subject to discrepancies from the desired orientation.

5.6.2.1 Impact of Frame Rate

Figure 5.13 plots the detection probability versus frame rate. We can observe that the detection probability increases with frame rate. This is because a higher frame rate enables the GMM to be more timely updated to capture the environmental condition changes. However, the improvement gained by increasing frame rate is fairly limited. The reasons are two-fold. First, debris objects usually drift slowly with water current. The GMM can thus be updated with a low rate. Second, our horizon-based image registration effectively mitigates the impact of camera shaking, which is the major affecting factor for debris detection. Hence, a low frame rate can achieve satisfactory

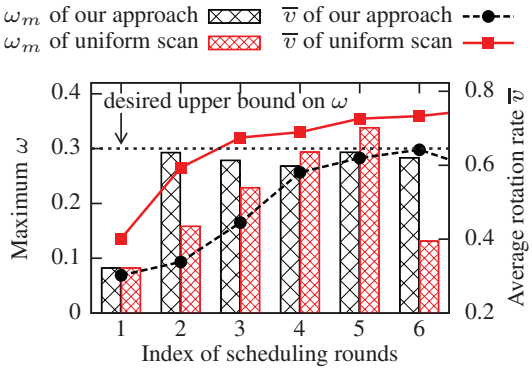


Figure 5.14: Maximum ω and average rotation rate $\overline{v}$ versus scheduling round.

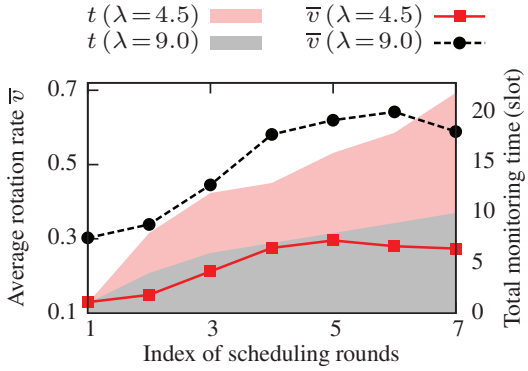


Figure 5.15: Impact of arrival rate λ on rotation rate $\overline{v}$ and scheduling frequency.

debris detection performance. Moreover, in terms of system overhead, a low frame rate is desirable for smartphone platforms that have constraints on resources and energy supply.

5.6.2.2 Coverage Effectiveness

We compare the coverage effectiveness of our approach with a heuristic baseline approach that uniformly scans the surveillance region. Specifically, the baseline approach evenly partitions the semi-circular surveillance region into $\lceil \pi/\alpha \rceil$ sub-regions. In this scheme, the robot sequentially scans the sub-regions by making an orientation adjustment each slot. For our approach, the desired upper bound on miss coverage rate is set to be 0.3. We evaluate the coverage effectiveness by examining the maximum miss coverage rate (denoted by ω_m) among all arriving angles. The ω_m characterizes the worst-case debris coverage performance for a rotation schedule. Figure 5.14 plots the ω_m versus index of scheduling rounds. We can observe that our approach can guarantee the upper bound on miss coverage rate, since it adaptively allocates surveillance slots based on the ω at each β, while the uniform scan cannot bound the ω_m. Moreover, we evaluate the average rotation rate $\overline{v}$, where a larger $\overline{v}$ indicates higher power consumption. Figure 5.14 also plots the $\overline{v}$ versus index of scheduling rounds. As the uniform scan approach continuously adjusts the orientation, it consumes more power than our approach.

5.6.2.3 Impact of Arrival Rate

Figure 5.15 plots the average rotation rate $\overline{v}$ versus index of scheduling rounds under different settings of debris arrival rate λ. We can see that the robot has a higher $\overline{v}$ when λ is higher. This is consistent with the intuition that the robot needs to rotate faster when debris arrives more frequently. Figure 5.15 also plots the total monitoring time versus index of scheduling rounds. We can see that for a higher λ, the rotation scheduling has to be conducted more frequently to meet the upper-bounded miss coverage rate ω, as ω at the uncovered arriving angles increases with λ. The robot is scheduled to rotate less frequently under a lower λ, resulting in a lower $\overline{v}$ and a longer monitoring interval at each orientation. Overall, the results in Figure 5.15 demonstrate that our approach can adaptively control the robot rotation to achieve the desired coverage performance while minimizing the energy consumption.

5.6.2.4 Impact of Estimation Errors

This set of simulations evaluate the impact of estimation errors of debris movement orientation θ on debris coverage performance. Let $\varepsilon(\cdot)$ denote the relative estimation error with respect to ground truth. Figure 5.16 plots the maximum miss coverage rate ω_m versus index of scheduling rounds under various estimation error levels for θ. We note that the estimation error in θ affects the scheduling of camera orientation, as θ determines the distribution of debris arriving probability at the frontier of surveillance region. From the figure, we can see that our rotation scheduling algorithm generally maintains ω_m below the desired upper bound of 0.3, as long as the relative estimation errors for θ is below 15%. As shown in Section 5.6.1, $\varepsilon(\theta)$ is only about 7%. Thus, SOAR can tolerate practical inaccuracy in estimating θ. Moreover, our analysis (omitted due to

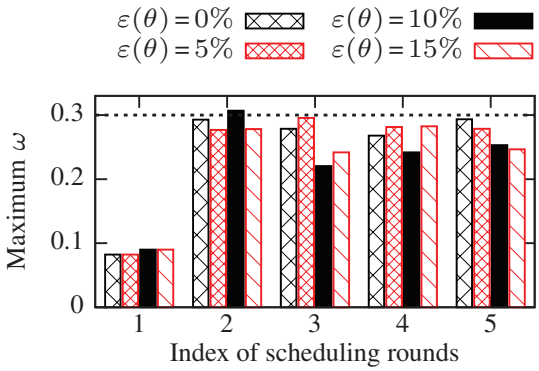


Figure 5.16: Impact of estimation error for debris movement orientation θ.

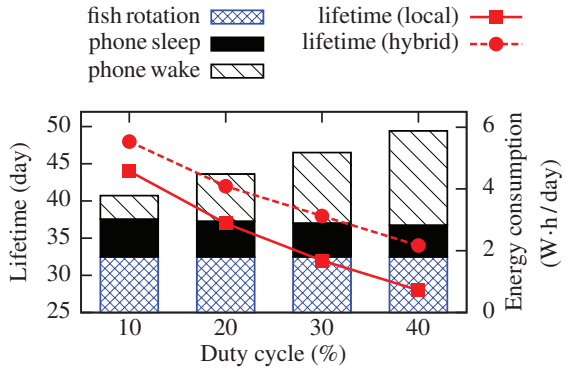


Figure 5.17: Projected lifetime and daily energy consumption breakdown of SOAR.

space limit) validates that SOAR shows similar tolerance to the inaccuracy in estimating debris arrival rate λ.

5.6.2.5 Projected Lifetime

Finally, we evaluate the lifetime of SOAR based on its power consumption profile. The major energy consumption of SOAR is due to the smartphone during the wake periods and the fish rotation. According to our results from the integrated evaluation (see Figure 5.12), a monitoring round lasts for 5 minutes averagely, and SOAR can rotate to the scheduled orientation within 15 seconds. We can thus calculate the upper bound on daily (12 hours of daytime) energy consumption for fish rotation as $(12 \times 60/5) \times (15/3600) \times p_r \,\mathrm{W \cdot h}$, where p_r is the battery power consumption for fish rotation. The energy drain on smartphone can be calculated using offline power consumption measurements and the duty cycle. The total battery capacity of SOAR is $170\,\mathrm{W \cdot h}$, including a backup $13.5\,\mathrm{W \cdot h}$ and two main $75\,\mathrm{W \cdot h}$ batteries on the fish, and a $6.48\,\mathrm{W \cdot h}$ battery on the smartphone. Figure 5.17 plots the projected lifetime under various duty cycle settings, when all the CV tasks are conducted on the smartphone. The duty cycle is defined as the ratio of wake time to the total time. As expected, the lifetime decreases with duty cycle. Note that a low duty cycle will decrease the temporal sensing granularity of SOAR. For example, during a 5 minutes monitoring interval,

it can capture and process about 20 and 50 frames under 20% and 50% duty cycles, respectively. The breakdown of SOAR daily energy consumption is also shown in Figure 5.17. We find that the majority of energy is consumed by the wake periods and fish rotation. Moreover, Figure 5.17 also plots the projected lifetime when smartphone runs the *hybrid* scheme under a link speed of 2 Mbps. The *hybrid* scheme reduces the power consumption for smartphone during the wake periods by offloading the intensive Hough transform to the cloud. It can be seen that the *hybrid* scheme leads to 9.1% to 21.5% improvement on lifetime under different duty cycles.

5.7 Conclusion

This chapter presents SOAR – a new vision-based robotic sensor system designed for aquatic debris monitoring. SOAR integrates an off-the-shelf Android smartphone and a gliding robotic fish. The vision-based debris detection algorithms of SOAR effectively deal with various dynamics such as camera shaking and reflection. A rotation scheduling algorithm adaptively guides the rotation of SOAR to capture the images of arriving debris objects. Moreover, SOAR dynamically offloads the entire/partial image processing to the cloud for energy conservation.

Chapter 6

Water Surface Monitoring

6.1 Introduction

This chapter presents *Samba* (Smartphone-based aquatic monitoring robotic platform), a robot system that is based on SOAR (see Chapter 5) but has more exciting features for water surface monitoring. Specifically, Samba and SOAR fundamentally differ from each other in *four* aspects. First, Samba achieves much higher energy efficiency by integrating inertial sensing with visual sensing through a learning-based scheme. According to our experiments, Samba consumes 97% less energy than SOAR in processing an image frame. Second, the generic design of Samba enables the monitoring of either moving or static aquatic processes such as oil spill and HABs; however, SOAR detects moving debris objects only while treating all the static targets as background. Moreover, SOAR identifies debris objects using a pixel-based approach, while Samba detects target aquatic processes based on their color features without modeling each pixel. As a result, compared with SOAR, Samba drastically decreases the overhead of robot mobility control, as it does not rely on robot mobility to maintain the pixel-level correspondence across frames in dynamic aquatic environment. Lastly, Samba adopts a feedback-control-based algorithm that adapts the robot's movement based on the detected dynamics of the target, while the movement of SOAR is primarily driven by prior knowledge such as the arrival model of the debris objects.

To monitor aquatic processes on the water surface, we need to address several major challenges. First, aquatic processes are often scattered as patches over large geographic regions and spatiotem-

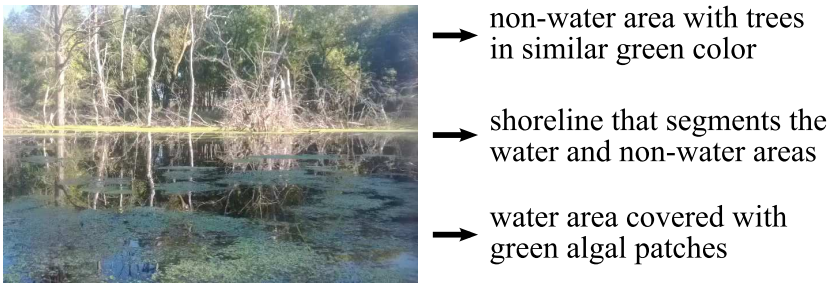


Figure 6.1: A sample image captured by the Samba prototype in an inland lake, where the water and non-water areas are separated by a shoreline; the trees exhibit similar color with the algal patches in the water area.

porally evolving [11, 71], which create challenges for fine-grained monitoring. Continuous visual sensing can track their evolution, which, however, imposes significant energy and computation overhead to smartphone. Second, phone's built-in cameras have limited field of view. Although the controllable mobility of robot can help improve the sensing coverage, aquatic locomotion may incur high energy consumption. Lastly, existing vision-based detection algorithms using background subtraction [84] perform poorly on the images captured in the aquatic environment. For example, the patches of the target aquatic process present in the camera view often block the background, making it difficult to differentiate between the foreground and the real background. Moreover, the target detection may be greatly affected by various dynamics such as blurred images caused by waves, highly variable illuminance, and complex non-water area in the image. These uncertainties can lead to excessive false alarms. Figure 6.1 shows a sample image captured by a Samba prototype in an inland lake, where the trees on the shore exhibit similar green color with the algal patches on the water, potentially resulting in false detections.

This chapter makes the following contributions:

1) We propose an inertial-sensing-assisted image segmentation approach to identifying the water area in the image. By focusing on the water area, Samba can reduce the false alarms caused by the non-water area. A key novelty of this approach is to leverage the energy-efficient inertial

sensing to replace the compute-intensive algorithms for visual feature extraction used in image segmentation. Specifically, Samba first learns the mapping models that project inertial sensor readings to visual features. It then uses these models and real-time inertial sensor readings to estimate the visual features (e.g., the shoreline in Figure 6.1) for image segmentation without actually extracting them from the images.

2) We propose several lightweight and robust CV algorithms to detect harmful aquatic processes in dynamic environment. Our vision-based detection algorithms, consisting of back projection and patch identification, detect the existence of a target process based on its distinctive color features. The algorithms are specifically designed to address the dynamics in aquatic monitoring such as the highly variable illuminance and camera noise.

3) We design a feedback-based robot rotation control algorithm that increases coverage and maintains a desired level of monitoring resolution on the evolving aquatic process, e.g., the area expansion of an algal patch between two consecutive observations during the rotation. Based on the dynamics of the target process, the control algorithm adapts the rotation speed of Samba to meet user's requirement on monitoring resolution.

6.2 Overview of Samba

Samba integrates an off-the-shelf Android smartphone with a robotic fish. The phone loads an app that implements the image processing and mobility control algorithms, including *hybrid image segmentation*, *aquatic process detection*, and *adaptive rotation control*. The robotic fish propels Samba by beating its tail, and communicates with the phone via a USB cable. Samba is designed to operate on water surface and monitor harmful aquatic processes such as oil spill and HABs.

These processes typically disperse as patches in the aquatic environment and exhibit distinctive colors [11, 71]. To monitor a large affected region, multiple Samba robots can be deployed to form a surveillance network. Their local observations can be sent back to a central server via the long-range communication interface on smartphones and stitched into a global map. In this chapter, we focus on the design of a single Samba robot. Due to the limited angular view of the phone's cameras, it is challenging to monitor the scattered patches of the target process. Although extra optical components like fisheye lens can be used to broaden the camera view, their integration to Samba will complicate the system design by introducing additional computation overhead due to the distorted images [84]. We leverage mobility to increase the sensing coverage. Specifically, Samba can rotate by performing discrete tail beats. We focus on the rotation mobility, because it is much more energy-efficient than moving forward that requires continuous tail beats.

Before deployment, Samba is trained by some sample images of the target aquatic process (TAP). The sample images are close-up photos of the TAP and can be provided by domain scientists. Samba can monitor multiple TAPs simultaneously when provided with their sample images. Samba conducts aquatic monitoring at a set of orientations, which are selected to ensure a full coverage of surrounding region given the angular coverage of Samba's on-board camera. After the initial deployment, it begins a TAP surveillance process consisting of multiple rounds. In each round, Samba keeps monitoring toward an orientation for a certain time interval. At the beginning of a round, Samba rotates to a new orientation and starts to capture images at a certain rate. For each image, it identifies the water area and executes several CV algorithms to detect the existence of TAP in real time. At the end of this round, Samba estimates the dynamics of TAP (e.g., evolution rate) and computes the monitoring interval for the next round. For example, if the TAP evolves rapidly, Samba will shorten the monitoring interval such that it can rotate back to the current orientation sooner, keeping smooth track of the TAP evolution; and vice versa. When drastic

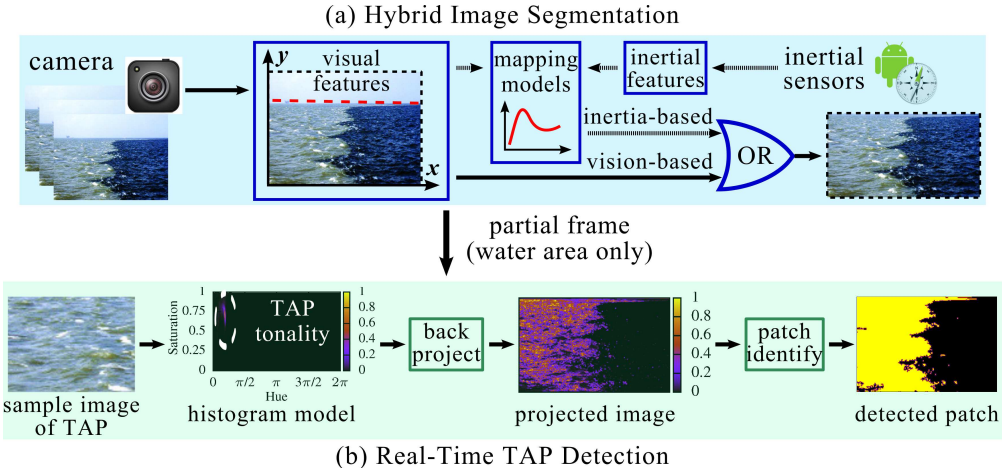


Figure 6.2: The aquatic environment monitoring pipeline when Samba keeps an orientation. Samba periodically adjusts its orientation to adapt to the dynamics of surrounding target aquatic process (TAP) patches.

evolution of the TAP is detected, Samba can alert the user by using the communication interface (cellular/WiFi) on smartphone. A primary design objective of Samba is to realize long-term (up to a few months) autonomous monitoring. Between two image captures, Samba sleeps to reduce energy consumption. One novel feature of Samba is the inertial-sensing-assisted image processing scheme that can lead to significant energy savings. In particular, Samba uses the learned models that partition an image based on inertial sensor readings, without actually extracting the visual features from the images using compute-intensive algorithms. Specifically, Samba comprises the following three major components.

First, it features hybrid image segmentation. By partitioning an image into water and non-water areas, Samba performs aquatic process detection in the water area only, thus reducing detection false alarms and computation energy consumption. Samba adopts a novel hybrid image segmentation framework as illustrated in Figure 6.2(a). Specifically, it uses both vision-based and inertia-based segmentation modes. This hybrid approach learns regression-based mapping models that project the inertial sensor readings to the visual features for image segmentation. Therefore, Samba can partition an image based on the visual features mapped from the inertial sensor read-

ings, avoiding executing the compute-intensive CV algorithms continuously. Samba switches to vision-based segmentation if the mapping models need to be updated, e.g., when the accuracy of inertia-based segmentation drops or Samba rotates to a new orientation.

Second, it features real-time TAP detection. This component detects TAP in the segmented images. As illustrated in Figure 6.2(b), it consists of two lightweight image processing modules, i.e., *back projection* and *patch identification*. First, Samba extracts the robust color features of TAP in HSV color space, and performs back projection to identify the candidate pixels of TAP in each segmented image. The projected image is then passed to patch identification for probability thresholding, noise removal, and patch recognition, which effectively deal with various environment and system dynamics such as the highly variable illuminance and camera's internal noise.

Third, it features adaptive rotation control. Samba monitors the surrounding TAP patches at fine spatiotemporal granularity while meeting energy consumption constraints. To adapt to the dynamics of TAP that is primarily affected by environmental conditions, we develop a feedback control algorithm to maintain the desired monitoring granularity. On the completion of a round, Samba estimates the dynamics of TAP (e.g., diffusion of oil slick and growth of HABs) based on detection results, and then calculates a new rotation speed such that the expected monitoring granularity in the next round can be maintained at the desired level.

6.3 Hybrid Image Segmentation

Image segmentation is the process of partitioning an image into multiple parts [84]. In aquatic monitoring, we adopt image segmentation to remove the non-water area from the captured images, and thus perform the detection of TAP in the water area only. This can avoid the false alarms that occur in the non-water area (e.g., those caused by trees in Figure 6.1) and reduce computation

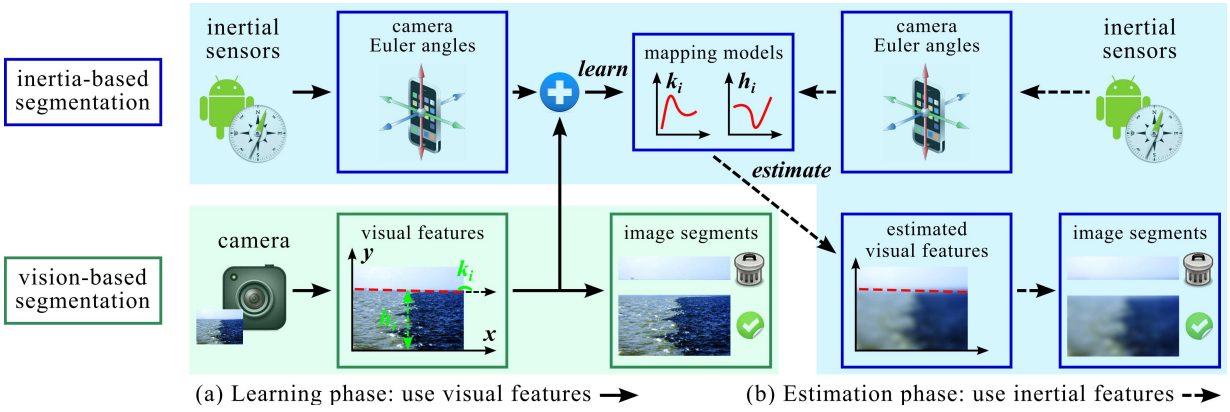


Figure 6.3: The overall structure of hybrid image segmentation where Samba switches between the vision-based and inertia-based segmentation modes. In the online learning phase, Samba jointly uses inertial and visual sensing to learn regression-based mapping models, which project the camera Euler angles (obtained from the inertial sensors) to the extracted visual features (obtained from the camera). In the estimation phase, Samba utilizes the learned mapping models to estimate the visual features and conducts the image segmentation accordingly.

in subsequent image processing tasks. Image segmentation is usually conducted based on visual features. For example, the shoreline can be used to identify the water area in inland lakes. However, most visual feature extraction CV algorithms are compute-intensive. According to our experiments (see Section 6.7.2.1), the Hough transform [84] for line extraction consumes more than 95% of the energy for processing an image. Moreover, the vision-based segmentation may fail due to the lack of differentiating color features and blurred images caused by waves. In this section, we propose a robust approach to overcoming the above limitations of vision-based segmentation.

6.3.1 Overview

In this chapter, we develop a novel hybrid image segmentation approach that utilizes both camera and inertial sensors on the phone, as illustrated in Figure 6.3. Inertial sensors provide camera's transient pose, which can be used to guide the segmentation of captured images. To characterize a camera's projection, the visual sensing approach is susceptible to the quality of captured image and blocked line of sight, while inertial sensing will not be affected by these factors. Moreover, the

energy consumption of inertial sensing is much lower than that of visual sensing. Our experiments show that the computation delay of inertia-based segmentation is only 2% of that of vision-based segmentation (see Section 6.7.2.1). The proposed hybrid approach aims to leverage these two heterogeneous sensing modalities. Specifically, it consists of an online learning phase and an estimation phase. The vision-based image segmentation is executed in the learning phase, and the inertia-based image segmentation is executed in the estimation phase. In our design, Samba switches between the two modes to save system energy while maintaining segmentation accuracy.

In the learning phase, images are segmented based on the extracted visual features, as shown in Figure 6.3(a). Meanwhile, the inertial sensor readings are converted to Euler angles, which describe the camera's orientation with respect to the world coordinate system. These angles, along with the corresponding visual features, are then used to learn the mapping models via regression. In the estimation phase, the learned mapping models estimate the visual features based on the inertial sensor readings and guide the image segmentation, as shown in Figure 6.3(b). Therefore, the compute-intensive visual feature extraction CV algorithms are avoided. The hybrid approach periodically calibrates the inertial sensors and updates the mapping models in the learning phase, thus adapting to the possible environmental condition changes.

6.3.2 Measuring Camera Orientation

A key step in the hybrid image segmentation is to relate the inertial sensing results with the corresponding visual sensing results, which relies on the camera projection model. The camera's orientation characterizes its projection. Therefore, it is critical to accurately measure the camera's orientation. There are typically no built-in physical orientation sensors on Android smartphone[1]. Hence, we need to implement our own virtual orientation sensor based on other available inertial

[1]The orientation sensor API has been deprecated in Android since version 2.2 [4].

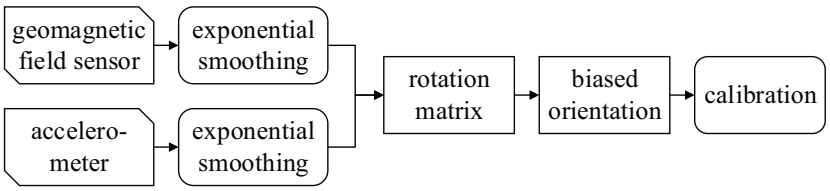


Figure 6.4: Workflow of virtual orientation sensor.

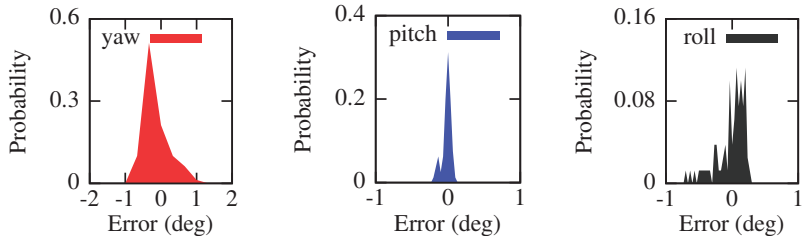


Figure 6.5: PDF of orientation measurement error.

sensors such as accelerometer and geomagnetic field sensor. However, inertial sensor readings are often inaccurate and volatile. Studies have shown that the mean error of orientation measurements computed based on inertial sensor readings by a simple algorithm can be up to 30° [8]. According to our experiments, the inaccuracy in orientation measurements mainly results from bias. To deal with the bias and random noise, we smooth and calibrate the sensor readings. Figure 6.4 shows the workflow of our virtual orientation sensor. It first uses inertial sensor readings to compute the rotation matrix that bridges the device coordinate system to the world coordinate system, and then obtains the phone's orientation. Figure 6.5 plots the error distribution of orientation measurements on a Google Nexus 4 phone, where the ground truth is measured using a protractor. We can see that our virtual sensor yields fairly accurate orientation measurements.

6.3.3 Feature Extraction

In order to establish the mapping models, Samba needs to extract features from both camera observations and inertial sensor readings for the same frame. In this chapter, we focus on the lake environment where the shoreline can help identify the water area. We assume that the shoreline is

the longest visible line to Samba. This is reasonable as the shoreline usually crosses the captured images horizontally, as shown in Figure 6.1. Note that the shoreline is not static to Samba because of waves and Samba's rotational movements. Our approach can be easily extended to address other scenarios with different visual references for image segmentation. For frame i, let k_i and h_i denote the shoreline slope and average vertical coordinate in the image plane, as illustrated in Figure 6.3(a). The phone can extract the shoreline using Hough transform [84] and thus obtain k_i and h_i. With these two parameters, frame i can be partitioned accordingly. Therefore, we define (k_i, h_i) as the *visual feature*s of frame i.

The camera's orientation, represented by Euler angles, is measured by the virtual orientation sensor. At runtime, we synchronize the orientation measurements with visual features obtained from the camera using the timestamps. We now derive their relationship based on the camera perspective projection model [84]. We set the world coordinate system originated at the camera. Let α_i, β_i, and γ_i denote the yaw, pitch, and roll angles of the camera when frame i is taken, and $\mathbf{P} = (x, y, z)$ denote the coordinates of an observable point on the shoreline in the world coordinate system. We first project $\mathbf{P}$ to the Cartesian coordinate system that is originated at the camera and rotated by $(\alpha_i, \beta_i, \gamma_i)$ with respect to the world coordinate system. The projected coordinates, denoted by $\mathbf{P}' = (x', y', z')$, are given by

$$\mathbf{P}' = \mathbf{P} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\gamma_i & \sin\gamma_i \\ 0 & -\sin\gamma_i & \cos\gamma_i \end{bmatrix} \cdot \begin{bmatrix} \cos\alpha_i & 0 & -\sin\alpha_i \\ 0 & 1 & 0 \\ \sin\alpha_i & 0 & \cos\alpha_i \end{bmatrix} \cdot \begin{bmatrix} \cos\beta_i & \sin\beta_i & 0 \\ -\sin\beta_i & \cos\beta_i & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot$$

Based on the similar triangles [84], we then project $\mathbf{P}'$ to the 2D image plane using the following

formula:

$$\begin{bmatrix} x'' & y'' \end{bmatrix} = \begin{bmatrix} x' & y' \end{bmatrix} \cdot \begin{bmatrix} f_x \cdot g(z') & 0 \\ 0 & f_y \cdot g(z') \end{bmatrix}.$$

where (x'', y'') denote the corresponding coordinates in the image plane, f_x and f_y are hardware-dependent scaling parameters, and $g(z')$ is a scaling factor determined by z'.

The above transforms allow us to relate the camera's orientation $(\alpha_i, \beta_i, \gamma_i)$ to the visual features (h_i, k_i). We first derive k_i that represents the slope of the shoreline. Suppose (x''_1, y''_1) and (x''_2, y''_2) are two points on the extracted shoreline in frame i, and (x_1, y_1, z_1) and (x_2, y_2, z_2) are the corresponding points in the world plane. The shoreline slope in the image plane, i.e., k_i, can be computed as

$$k_i = \frac{y''_1 - y''_2}{x''_1 - x''_2} = \frac{f_y}{f_x} \cdot \frac{(x_1 f_{i,3} - y_1 f_{i,4}) - (x_2 f_{i,3} - y_2 f_{i,4})}{(x_1 f_{i,1} - y_1 f_{i,2}) - (x_2 f_{i,1} - y_2 f_{i,2})} = \omega_1 \cdot \frac{f_{i,3} - \omega_2 f_{i,4}}{f_{i,1} - \omega_2 f_{i,2}}, \tag{6.1}$$

where $\omega_1 = f_y / f_x$ and $\omega_2 = (y_1 - y_2)/(x_1 - x_2)$ are two unknown but constant coefficients that are determined by the camera hardware and the shoreline slope in the world coordinate system. The 4-tuple $(f_{i,1}, f_{i,2}, f_{i,3}, f_{i,4})$ can be computed based on the camera's orientation by

$$\begin{aligned} f_{i,1} &= \cos(\alpha_i) \cdot \cos(\beta_i), \\ f_{i,2} &= \cos(\alpha_i) \cdot \sin(\beta_i), \\ f_{i,3} &= \sin(\alpha_i) \cdot \cos(\beta_i) \cdot \sin(\gamma_i) - \sin(\beta_i) \cdot \cos(\gamma_i), \\ f_{i,4} &= \sin(\alpha_i) \cdot \sin(\beta_i) \cdot \sin(\gamma_i) + \cos(\beta_i) \cdot \cos(\gamma_i). \end{aligned}$$

They interpret the camera projection model using orientation measurements. The above 4-tuple are defined as *inertial feature*s, which are related to the visual feature through the *mapping model*

given by Equation (6.1). Similarly, we can derive the mapping model between another set of inertial features (with 3 unknown coefficients) and the vertical position of the shoreline in the image plane (i.e., h_i). Due to space limitation, we omit the details here. With the mapping models, we can estimate the shoreline purely based on the inertial sensor readings and conduct image segmentation.

6.3.4 Learning Mapping Models

With the extracted visual and inertial features, Samba can learn the mapping models that project the latter to the former. Based on Equation (6.1), we adopt a regression-based approach to estimating the unknown coefficients ω_1 and ω_2. Specifically, the training data instance from frame i can be expressed as $(k_i, f_{i,1}, f_{i,2}, f_{i,3}, f_{i,4})$, in which k_i is obtained from the camera and $(f_{i,1}, f_{i,2}, f_{i,3}, f_{i,4})$ are computed based on inertial sensor readings. Suppose the training dataset consists of N frames. We define the *feature set*, denoted by $\mathbf{F}$, as an $N \times 4$ matrix that contains the inertial features extracted from the N frames

$$\mathbf{F} = \begin{bmatrix} f_{1,1} & f_{1,2} & f_{1,3} & f_{1,4} \\ \vdots & \vdots & \vdots & \vdots \\ f_{N,1} & f_{N,2} & f_{N,3} & f_{N,4} \end{bmatrix}_{N \times 4}.$$

Moreover, we define the *observation vector*, denoted by $\mathbf{K}$, as an $N \times 1$ vector that contains the visual features (i.e., the shoreline slope), i.e., $\mathbf{K} = [k_1, \ldots, k_N]^\intercal$. We then employ multivariate nonlinear regression to learn ω_1 and ω_2 with $\mathbf{F}$ and $\mathbf{K}$. Using the estimated ω_1 and ω_2, we can infer the shoreline slope from the inertial sensor readings based on the mapping model given by Equation (6.1).

We now discuss the impact of training dataset size (i.e., N) on system performance. The N training frames can be evenly selected from an updating period. Therefore, the value of N defines the frequency at which Samba switches between the vision-based and inertia-based segmentation modes. In order to perform regression, N should be no smaller than the number of coefficients to be learned (i.e., 2). In aquatic monitoring, N is expected to be larger than this lower bound to account for the dynamics resulted from camera shaking and sensor noises. Intuitively, a larger training dataset can enrich the diversity of training data and hence improve the regression accuracy. However, it also imposes additional computation overhead for the phone. Specifically, with a larger N, visual feature extraction has to be conducted more frequently to meet the need of observation vector used for learning mapping models. Moreover, the computation overhead of the regression also increases with N. In Section 6.7.2.4, we will evaluate the trade-off between regression accuracy and system overhead. At runtime, Samba can assess inertia-based segmentation accuracy in the learning phase by comparing it with vision-based segmentation, and adaptively configure N.

6.4 Aquatic Process Detection

The real-time TAP detection pipeline of Samba is illustrated in Figure 6.2(b). Although our approach is based on elementary CV algorithms, it is non-trivial to customize them for detecting TAP and efficiently implement on smartphone. Samba consists of two major image processing modules, i.e., *back projection* and *patch identification*. The back projection models the TAP by extracting its color histogram, which is built using selected channels in HSV. The HSV representation is more robust to the highly variable illuminance than the RGB color space [61]. It then detects candidate pixels of TAP in the image segment of water area. The patch identification removes the salt-and-pepper noises from the projected segmented image and then identifies the TAP patches.

6.4.1 Back Projection

Back projection is a CV technique that characterizes how well the pixels of a given image fit a histogram model [84]. In this chapter, we adopt back projection to identify the existence of TAP in the captured frames given its unique features. The patches of TAP are often scattered over large water area. For example, the HABs occurred at Lake Mendota in Wisconsin produce algal patches spread over 15 square kilometers water area [33]. Therefore, the robot may have a camera view dominated by the TAP patches when deployed in an affected region. The widely used target detection approach based on background subtraction is thus not applicable, as it needs to build a background model without the TAP. Moreover, this approach requires all the captured images to be aligned and thus has limited feasibility in uncalm waters. Our proposed approach is motivated by the fact that a TAP usually maintains featured and nearly constant colors. For example, oil slicks often appear to be brighter than the surrounding water surface [11], and the typical color of HABs is green or red [71]. Therefore, to perform detection, we can match the pixels in a new frame with the histogram model constructed with offline sample images of the TAP.

Back projection constructs the histogram model based on sample images as follows. Let I_0 denote a sample image of the TAP. We first convert the representation of I_0 to HSV model. In HSV, the hue channel represents the color, the saturation channel is the dominance of hue, and the value channel indicates the lightness of the color. The hue and saturation channels are robust to illumination changes [61] and hence effective in interpreting color features in the presence of reflection on water surface. Thus, we adopt the hue-saturation histogram and calculate it from I_0. We equally divide the range of hue $[0,360)$ and the range of saturation $[0,1]$ into $[0,h_2,h_3,\ldots,h_{p-1},360)$ and $[0,s_2,s_3,\ldots,s_{q-1},1]$, respectively, to compute the color histogram $\mathbf{M}_{q\times p}$. Specifically, the $(i,j)^{\text{th}}$ entry of $\mathbf{M}$ is the frequency of pixels in I_0 with saturation within $[s_i,s_{i+1})$ and hue within $[h_j,h_{j+1})$.

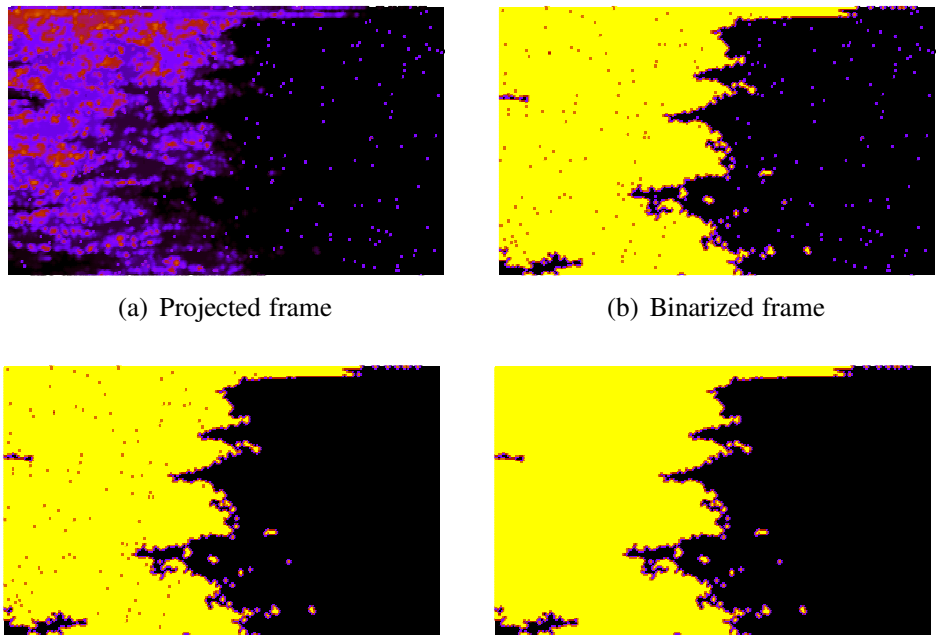
Note that the color histogram can be obtained based on multiple images by repeating the above process. Let $\widetilde{\mathbf{M}}$ denote the normalized histogram where each element quantifies the probability that the corresponding color represents the TAP. Therefore, $\widetilde{\mathbf{M}}$ depicts the tonality of the TAP, which is used as its histogram model. Figure 6.2(b) shows a sample image of HABs occurred in the Gulf of Mexico [91] and the extracted hue-saturation histogram model.

When a new segmented frame (denoted by I_t) is available, we leverage the histogram model $\widetilde{\mathbf{M}}$ to detect the existence of TAP. For each pixel $\mathbf{p}_{m,n}$ in I_t, we first extract its hue $h_{m,n}$ and saturation $s_{m,n}$, and locate the corresponding element in $\widetilde{\mathbf{M}}$, i.e., $\widetilde{\mathbf{M}}(s_{m,n}, h_{m,n})$. We then construct a projected frame I_1' that is in the same size of I_t but replaces each pixel $\mathbf{p}_{m,n}$ with $\widetilde{\mathbf{M}}(s_{m,n}, h_{m,n})$. Note that $\widetilde{\mathbf{M}}(s_{m,n}, h_{m,n})$ characterizes the probability that $\mathbf{p}_{m,n}$ in I_t represents a pixel of TAP. Visually, the brighter a pixel in I_1' is, the larger probability that the corresponding pixel in I_t is the TAP. An example of the projected frame I_1' is shown in Figure 6.6(a).

In back projection, each pixel in the new frame is classified based on how well it matches the color histogram obtained from TAP sample images. Therefore, the similarity in color between the TAP and water area affects the detection performance. In this chapter, we adopt a *color similarity* [87] metric to measure the color resemblance. It quantifies the similarity between any two colors based on their proximity in the cylindrical HSV model. For two colors indexed by (h_i, s_i, v_i) and (h_j, s_j, v_j), the color similarity, denoted by η, is given by

$$\eta = 1 - \sqrt{\frac{(v_i - v_j)^2 + (s_i \cos h_i - s_j \cos h_j)^2 + (s_i \sin h_i - s_j \sin h_j)^2}{5}}.$$

The η ranges from 0 to 1, where the lower bound 0 indicates the highest level of color contrast and the upper bound 1 represents the same color. Therefore, a larger value of η suggests a stronger color resemblance between the TAP candidate and water area, and hence the TAP candidate is less



(a) Projected frame (b) Binarized frame

(c) Opening morphology (d) Detected result

Figure 6.6: Sample patch identification process.

likely to be identified by the patch identification module in Section 6.4.2. In Section 6.7.2.5, we will evaluate the impact of η on detection performance.

6.4.2 Patch Identification

As illustrated in Figure 6.6, patch identification works on the projected image and identifies the TAP patches. In the projected frame shown in Figure 6.6(a), each pixel maintains a value within $[0, 1]$, which represents the probability that it belongs to the TAP. We adopt a threshold-based approach to removing the pixels with extremely low probabilities. Specifically, for any pixel with a value higher than a threshold, we consider this pixel as a candidate TAP pixel and round its value to 1; otherwise, it is classified as a pixel that represents the water area and set to 0. To determine this threshold, we try various settings and find that, with a setting of 0.1, the detected TAP boundary is most similar to visual observation. The binarized frame, as shown in Figure 6.6(b), often contains randomly distributed noise pixels. This is because the back projection is conducted in a pixel-wise

manner where the labeling of candidate TAP pixel can be affected by camera's internal noise. To remove these false alarms, we apply the opening morphology operation [84]. It consists of an erosion and a dilation, in which the former eliminates the noise pixels and the latter preserves the shape of true TAP area. Figure 6.6(c) depicts the resulted frame after noise removal. We then perform region growing [84] to identify the TAP patches. In particular, it uses the detected pixels as initial seed points and merges connected regions to obtain the candidate TAP patches. Finally, we apply another threshold to exclude the small patches. The patch with a small area in the frame usually indicates a false alarm. Figure 6.6(d) depicts the final detection result. We note that the detected TAP patches can be larger than the ground truth due to the reflection of trees on the water surface. One possible approach to addressing this is to improve the histogram model accuracy by using more selective sample images of TAP.

6.5 Adaptive Rotation Control

To monitor the surrounding TAP patches at fine spatiotemporal granularity, Samba needs to periodically adjust its orientation. However, this is challenging because the evolution of an aquatic process is heavily affected by changeable and even unpredictable environmental conditions. For example, the diffusion of oil slick can be affected by water salinity and temperature [50], and the growth of HABs is sensitive to local nutrient availability [36]. To adapt to such dynamics, we propose a rotation control algorithm, which maintains the monitoring granularity (e.g., the area expansion of an algal patch between two consecutive observations during robot rotation) at the desired level by adjusting the monitoring interval in each round.

6.5.1 Dynamics of Aquatic Process

The sensing coverage of a camera is described by its *field of view* (FOV) where any target with dimensions greater than some thresholds can be reliably identified [84]. FOV is typically modeled as a sector at the camera with an angular view θ and a radius R, in which θ depends on lens and R can be measured for a particular application. Since a TAP is likely to be scattered over a large region [11, 71], we define the *surveillance region* of Samba as the circular area originated at the robot with a radius R. Limited by θ, Samba needs to change its orientation to cover all the TAP patches within its surveillance region. As the TAP evolves (e.g., diffusion of oil slick), it has to continuously rotate to achieve the desired temporal coverage of each orientation.

The robot rotation initiates a monitoring round. Each round has a camera orientation and an associated monitoring interval. In our design, we equally divide the circular surveillance region into several sectors based on camera's angular view θ such that the surveillance region can be fully covered by these discrete orientations. During a round, Samba remains stationary toward an orientation and conducts hybrid image segmentation and TAP detection following the user-specified sleep/wake duty cycle. For each frame, the severeness of TAP can be measured by the total area of detected patches in the frame. The TAP and robot will drift at a similar speed and therefore remain in the same inertial system. Thus, the expansion of TAP patches characterizes the its dynamics. Using frames captured at different time instants, Samba can estimate the dynamics of TAP under current environmental conditions by computing the change in severeness.

To achieve fine spatiotemporal monitoring granularity with limited energy budget, the rotation of Samba needs to be carefully scheduled. Samba controls the monitoring interval of each round to adapt to the dynamics of TAP. We define *rotation speed*, denoted by ρ, as the rotated angle over a monitoring interval. Note that the rotated angle for each round is fixed. For simplicity

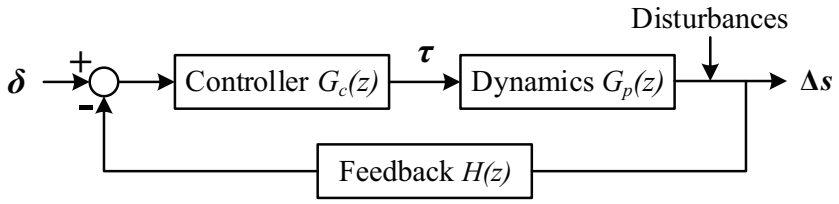


Figure 6.7: The closed loop for robot rotation control.

of exposition, the following discussion focuses on rotation speed. If the TAP spreads fast, ρ is expected to be large to capture the evolving dynamics. When the evolution of TAP slows down, Samba may decrease ρ accordingly to save energy. Let ε denote the dynamics of TAP measured by the camera. We define the monitoring resolution Δs as the change in severeness between two consecutive observations toward a certain orientation. Therefore, Δs depends on the dynamics of TAP and ρ. For example, the spread of diffusion process is approximately linear with time [16], i.e., $\Delta s = \varepsilon \cdot 2\pi/\rho$. In Section 6.5.2, a robot rotation control algorithm is designed based on this model, and it can be easily extended to address other models of TAP dynamics.

6.5.2 Robot Rotation Control

Our objective is to maintain a desired resolution on severeness change, denoted by δ, under various environment and system uncertainties. To save energy, Samba remains stationary as long as it can provide the required resolution. The setting of δ describes how smooth the user aims to keep track of the TAP. Figure 6.7 shows the block diagram of feedback control loop, where $G_c(z)$, $G_p(z)$, and $H(z)$ represent the transfer functions of the rotation scheduling algorithm, the dynamics model of TAP, and the feedback, respectively. In particular, the desired resolution δ is the *reference*, and the actually detected severeness change Δs is the *controlled variable*. Because Δs is a nonlinear function of ρ, we define $\tau = 2\pi/\rho$ as the *control input* to simplify the controller design. Thus, $\Delta s = \varepsilon \cdot \tau$, and its z-transform is $G_p(z) = \varepsilon$. In each round, Samba updates τ for the next round

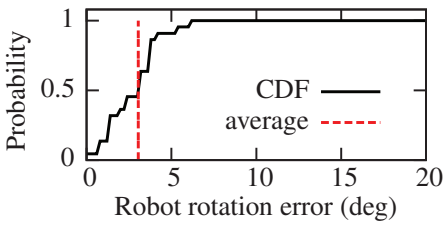


Figure 6.8: CDF and average of Samba orientation adjustment errors.

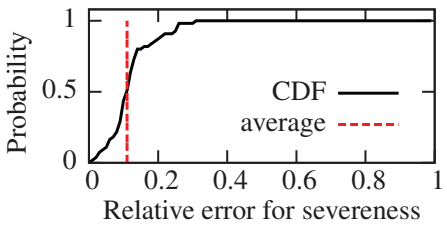


Figure 6.9: CDF and average of relative errors for detected severeness.

and sets ρ accordingly. As the feedback control will take effect in the next round, $H(z) = z^{-1}$, representing the delay of one orientation adjustment. Given that the system is of zero order, it is sufficient to adopt a first-order controller to maintain the stability and convergence of the control loop [68]. Therefore, we set $G_c(z) = \frac{a}{1-b\cdot z^{-1}}$ where $a > 0$ and $b > 0$. Following the standard method for analyzing stability and convergence [68], the stability and convergence condition can be obtained as $b = 1$ and $0 < a < 2/\varepsilon$.

The uncertainties are modeled as disturbances in the control loop shown in Figure 6.7. First, Samba has rotation errors. It may not head exactly toward the desired orientation due to complex fluid dynamics. Figure 6.8 evaluates this error by the discrepancies between desired and actual orientations, where we collect the data using our prototype (see Section 6.6). Second, the detected severeness of TAP exhibits variance. We define the relative error for Δs as the absolute error to the ground truth of TAP area in the image. As the detection of severeness is based on CV algorithms that work in a pixel-wise manner, it can be affected by dynamics like camera noise. In light of these disturbances, we design the controller $G_c(z)$ to mitigate their impact. From control theory [68], the effects of injected disturbances on the controlled variable Δs can be minimized if the gain of $G_c(z)G_p(z)H(z)$ is set as large as possible. By jointly considering the stability and convergence, we set $a = 2c/\varepsilon$ where c is a relatively large value within $[0, 1]$. In the experiments, we set c to be 0.85.

Implementing $G_c(z)$ in the time domain gives the robot rotation scheduling algorithm. Accord-

ing to Figure 6.7, we have $G_c(z) = \tau(z)/(\delta - H(z)\Delta s)$. From $H(z)$ and $G_c(z)$, the control input can be expressed as $\tau(z) = z^{-1}\tau(z) + 2b\varepsilon^{-1}(\delta - z^{-1}\Delta s)$, and its time-domain implementation is given by $\tau_k = \tau_{k-1} + 2b\varepsilon^{-1}(\delta - \Delta s_{k-1})$ where k is the count of orientation adjustments. The average rotation speed to be set for the k^{th} round is thus given by $\rho_k = 2\pi/\tau_k$.

6.6 Samba Implementation

We have built a proof-of-concept prototype of Samba for evaluation. The hybrid image segmentation, TAP detection, and adaptive rotation control algorithms presented in Sections 6.3 – 6.5 are implemented on Android. The app takes about 6.24 MB storage on the phone after installation, and requires about 10.8 MB RAM allocation while running on a frame resolution of 720×480. To exploit the multi-core computation capability, the visual feature extraction, mapping models learning, and TAP detection are implemented in separate threads.

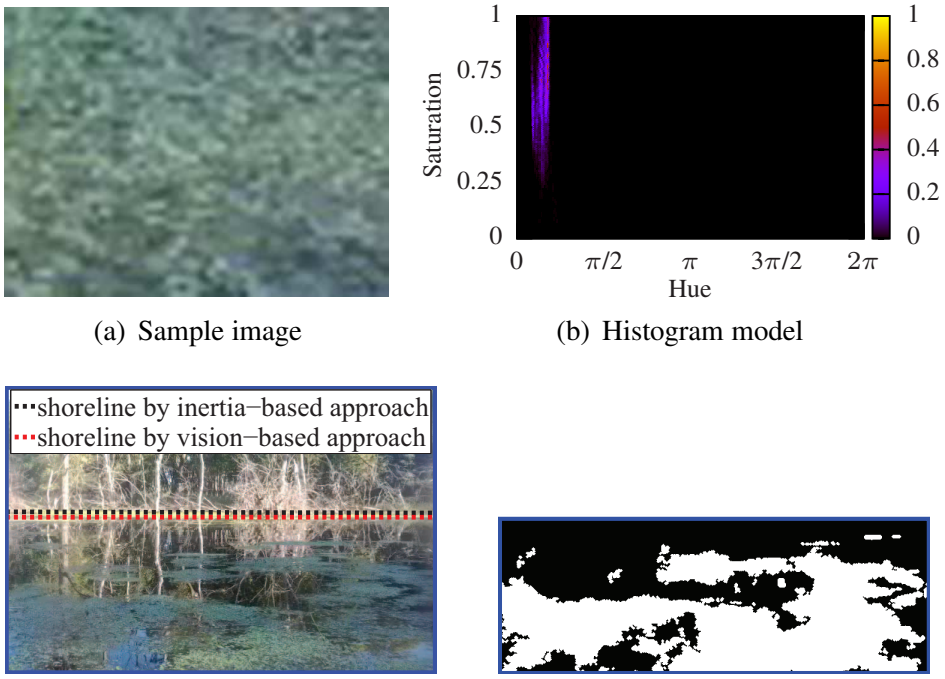
On initialization, Samba extracts the hue-saturation histogram of TAP based on sample images. When a new frame is available, it first conducts image segmentation to identify the water area. After the mapping models are learned, Samba switches between the vision-based and inertia-based segmentation modes according to the frequency specified by N (see Section 6.3.4). Then the robot executes the TAP detection algorithms on the segmented frame. During the implementation, we find that the Hough transform, which is used to extract the shoreline, incurs excessive delay. To address this issue, we use OpenCV's implementation through Java Native Interface. We also examined the frame processing performance when the app uses the new runtime option ART [5], which is introduced in Android 4.4 and pre-complies the Java code into system-dependent binaries. According to our measurements, ART can reduce the system delay by about 20% over Dalvik.

6.7 Performance Evaluation

We evaluate Samba through field experiments, lab experiments, and trace-driven simulations. The field experiments thoroughly test Samba's performance in detecting real HABs in an inland lake. The lab experiments evaluate system overhead, monitoring effectiveness under a wide range of environmental conditions, as well as the overall performance of a fully integrated robot. The trace-driven simulations examine the performance of adaptive rotation control with varied settings. Our results show that Samba can achieve reliable detection performance in the presence of various dynamics, while maintaining a lifetime up to nearly two months.

6.7.1 Field Experiments

To test the detection performance of Samba in real world, we deploy our prototype in an inland lake with an area of 200,000 square feet on September 22, 2014. Part of the lake is covered by patches of filamentous algae [10] that exhibit pale green color, as shown in Figure 6.1. During the experiments, the average illuminance is around 1,082 lux. The impact of significant illuminance change is evaluated in Section 6.7.2.6. We set the frame resolution to be 720×480 and the frame rate to be 0.5 fps. The hybrid image segmentation updates the mapping models every 20 frames, i.e., $N = 20$. Because the HABs at the test site were in a stable stage, we focused on evaluating the detection performance while disabling the robot rotation. Samba runs the hybrid image segmentation and TAP detection algorithms consecutively on each frame. A total of 5,211 frames were captured and processed. For each frame, we manually pinpoint the boundary of algal patches to provide the ground truth. For comparison, we adopt a baseline approach that uses the same sample images but constructs the color histogram model in the RGB color space. The RGB-based baseline is executed offline using the captured frames. The purpose of our field experiments is



(a) Sample image (b) Histogram model

(c) A typical frame (d) Detection result

Figure 6.10: Sample HABs detection in the field experiments.

three-fold. First, we test the detection performance of Samba in a real aquatic environment with complex background and colors. Second, we evaluate the real-time execution of hybrid image segmentation and TAP detection algorithms. Lastly, we validate that Samba can effectively reduce the false alarms caused by the non-water area through image segmentation.

6.7.1.1 Sample HABs Detection

Figure 6.10 depicts a sample HABs detection in the field experiments. An image of the algal patch, as shown in Figure 6.10(a), is used as the sample image for the detection pipeline. Figure 6.10(b) shows the normalized hue-saturation histogram, in which each element characterizes the probability that the corresponding color represents the HABs. Therefore, a majority of colors in the histogram are of near-zero probability. For each frame, Samba first extracts the water area by locating the shoreline through hybrid image segmentation. Figure 6.10(c) shows a typical frame

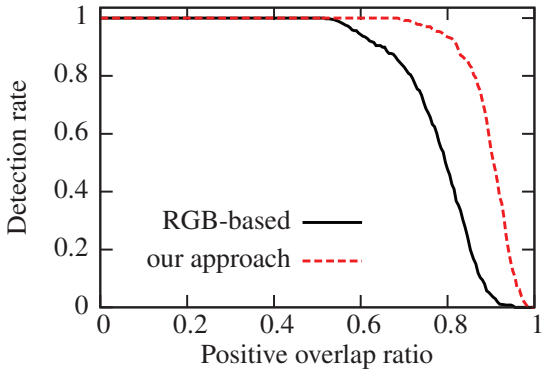


Figure 6.11: Detection performance under various positive overlap ratios.

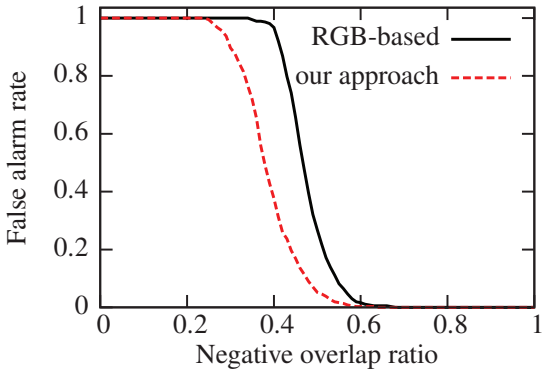


Figure 6.12: False alarm rate under various negative overlap ratios.

captured by Samba, where the red and black dashed lines represent the shorelines obtained by the vision-based and inertia-based image segmentation, respectively. As we can see, the inertia-based approach yields a fairly accurate estimation of shoreline, which is partially due to the calm water during field experiments. In Section 6.7.2.3, we will evaluate the performance of hybrid image segmentation under more dynamic environment. On the segmented water area, Samba executes the TAP detection algorithms. Figure 6.10(d) presents the detection result after back projection and patch identification. We can observe that our TAP detection algorithms effectively identify the algal patches in the segmented frame.

6.7.1.2 Detection Performance

We now evaluate the detection performance of Samba quantitatively. For each frame, we define the *positive overlap ratio* as the ratio of the overlap area between detection and ground truth to the actual TAP area. Hence, the positive overlap ratio characterizes the effectiveness of detection algorithms in a frame. Given a threshold on positive overlap ratio, we calculate the *detection rate* as the ratio of the frames with positive overlap ratio larger than this threshold to the total frames with TAP patches. Figure 6.11 plots the detection rate versus positive overlap ratio for our approach (i.e., using hue-saturation histogram) and the RGB-based baseline, respectively. When the positive

overlap ratio is lower-bounded at 0.8, Samba can achieve detection rate up to 94%. Moreover, our approach yields consistently higher detection rate than the baseline. In the field experiments, the HABs and water area have a color similarity η of 0.88, which represents a rather strong color resemblance (see Section 6.4.1). In Section 6.7.2.6, we will evaluate the impact of η on detection performance.

6.7.1.3 Effectiveness of Image Segmentation

In TAP detection, we define the *negative overlap ratio* as the area with false detections to the actual TAP area in each frame. A false alarm occurs when the negative overlap ratio exceeds a given threshold. We calculate the *false alarm rate* as the ratio of the frames with false alarms to the frames without TAP. In the field experiments, the false detections mainly result from the captured shore area. In particular, the trees on the shore, sharing a color similarity η with the target filamentous algae of up to 0.97, are the major contributor. Figure 6.12 plots the false alarm rate versus negative overlap ratio for our approach and the RGB-based baseline, respectively. We find that our approach achieves consistently lower false alarm rate than the baseline, as it can characterize the TAP color features more effectively. We then validate the effectiveness of image segmentation by evaluating the reduction in false alarms. Image segmentation allows Samba to perform HABs detection in the water area only. Figure 6.13 compares the false alarm rates for our approach and the RGB-based baseline, respectively, with and without image segmentation. The reported values are calculated by upper-bounding the negative overlap ratio at 0.5. We can see that by applying image segmentation, the false alarm rate of our approach is reduced by over 90%. Moreover, the baseline also benefits from image segmentation, but still yields a higher false alarm rate than our approach.

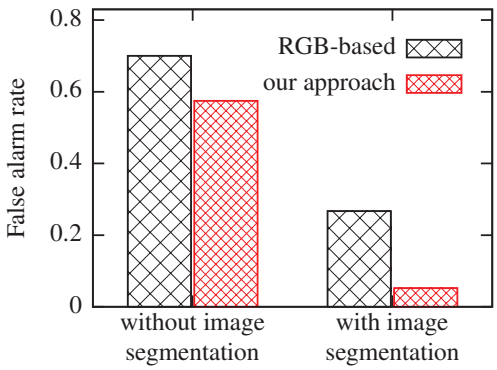


Figure 6.13: Reduction in false alarm rate after image segmentation.

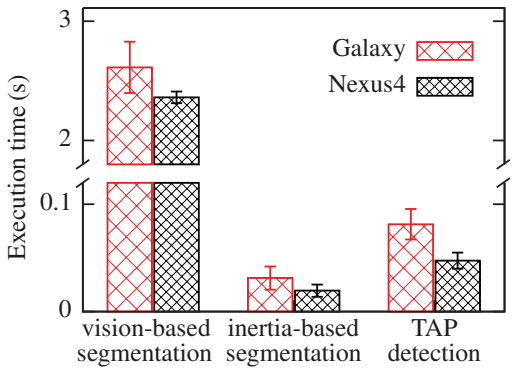


Figure 6.14: Execution time on representative smartphone platforms.

6.7.2 Lab Experiments

The objective of lab experiments is to evaluate Samba under more dynamic environment. The experiments were conducted in a 15 feet $\times$ 10 feet water tank in our lab. We vertically place a white foam board along one side of the tank to produce an artificial shoreline. The settings of Samba are consistent with those in the field experiments, unless otherwise stated. We test the performance of hybrid image segmentation and TAP detection algorithms under a wide range of environmental conditions and system settings such as training dataset size, wavy water, illuminance change, and color similarity between the TAP and water area.

6.7.2.1 Computation Overhead

We first evaluate the overhead of image segmentation and TAP detection algorithms on phone. Specifically, we measure the computation delay of each module, i.e., vision-based segmentation, inertia-based segmentation, and TAP detection, on Galaxy and Nexus4, respectively. The computation delay is measured as the elapsed time using Android API `System.nanoTime()`. The results are plotted in Figure 6.14. We can see that the vision-based segmentation incurs the longest delay, which is mainly due to the compute-intensive Hough transform. In contrast, the inertia-

Table 6.1: Samba Power Consumption Profile.

	Voltage (V)	Current (A)	Power (W)
Galaxy wake	3.7	0.439	1.624
Galaxy sleep	3.7	0.014	0.518
Servo motor	6	0.5	3

based segmentation is drastically efficient, achieving over 98% reduction in computation delay. Note that in the hybrid segmentation, Samba learns the mapping models every $N = 20$ frames. Thus, the measured overhead of inertia-based segmentation provides an overhead upper bound of the proposed hybrid approach. The TAP detection algorithms take about 80 and 50 milliseconds on Galaxy and Nexus4, respectively. Therefore, our aquatic monitoring solution is efficient on mainstream smartphones and can well meet the real-time requirement. Compare with SOAR [111] which typically takes more than 3 seconds to process a frame, Samba reduces the computation delay by about 97%.

6.7.2.2 Projected Lifetime

We now evaluate the lifetime of Samba based on its power consumption profile as shown in Table 6.1. Smartphone computation, standby, and fish rotation consume the highest powers. The energy drain on phone can be calculated using offline measurements and duty cycle. Specifically, we measure the power consumption of Galaxy using an Agilent 34411A multimeter when the phone is executing the TAP detection algorithms with vision-based and hybrid segmentations. According to the integrated evaluation, a monitoring round lasts 5 minutes on average, and Samba can rotate to the scheduled orientation within 15 seconds. We can thus upper-bound the daily consumed energy for fish rotation by $(12 \times 60/5) \times (15/3600) \times p_r \mathrm{W} \cdot \mathrm{h}$, where p_r is the motor power consumption for beating the tail. Figure 6.15 shows the projected lifetime of Samba when running the vision-based and hybrid segmentations, respectively. We can see that the system life-

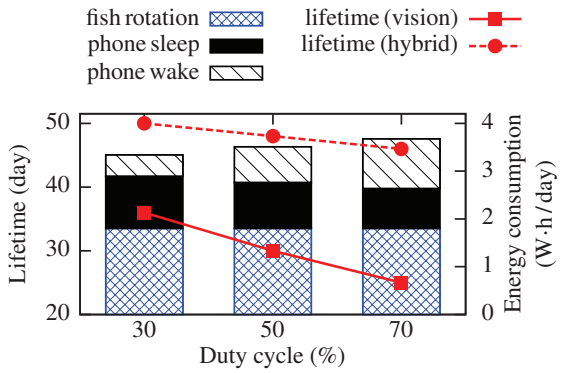


Figure 6.15: Projected lifetime and daily energy consumption breakdown of Samba.

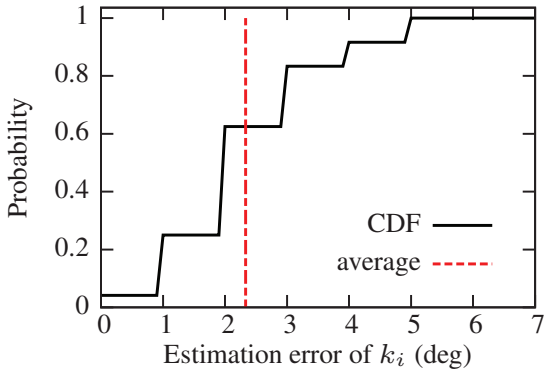


Figure 6.16: CDF and average of estimation error for shoreline slope k_i.

time is almost doubled by switching vision-based segmentation to hybrid approach. Figure 6.15 also evaluates the impact of duty cycle, which is defined as the ratio of wake time to the total time. As expected, the system lifetime decreases with duty cycle. In our current prototype, Samba has a total of 170 W · h battery capacity, including a backup 13.5 W · h and two main 75 W · h batteries on the fish, and a 6.48 W · h battery on the phone. Even with half of the battery capacity, Samba can achieve a lifetime of nearly a month with hybrid segmentation and 30% duty cycle. Moreover, the breakdown of daily energy consumption is plotted in Figure 6.15. We find that a majority of energy is actually consumed by the sleep periods and fish rotation. Owing to the high efficiency of hybrid image segmentation and TAP detection algorithms, the wake periods consume the least amount of energy. Therefore, the lifetime of Samba can be further extended if the phone is powered off during the sleep periods.

6.7.2.3 Image Segmentation Accuracy

We then evaluate the accuracy of hybrid image segmentation. To create dynamics, we generate waves by connecting a feed pump to the tank. As a result, the shoreline slope (i.e., k_i) varies up to 20°, and the average vertical coordinate (i.e., h_i) varies up to 170 pixels. We define the estimation errors of visual features as $|k_i - \widetilde{k_i}|$ and $|h_i - \widetilde{h_i}|$, where $\widetilde{k_i}$ and $\widetilde{h_i}$ are the estimated visual

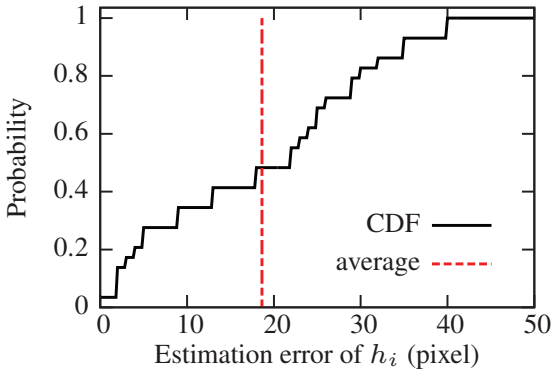


Figure 6.17: CDF and average of estimation error for shoreline position h_i.

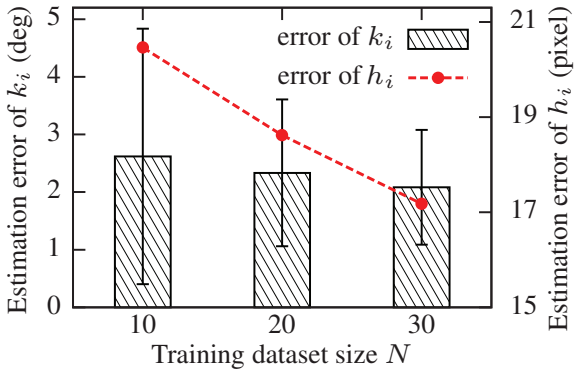


Figure 6.18: Impact of training dataset size N on estimation accuracy.

features by the inertia-based approach, and k_i and h_i are the ground truth measured by the camera. Figure 6.16 plots the CDF and average of the estimation errors for k_i. We can see that the inertia-based segmentation can accurately estimate k_i, with an average estimation error of about 2.3°. Moreover, Figure 6.17 plots the CDF of the estimation errors for h_i. As shown in this figure, the average estimation error is around 18 pixels. This set of experiments validate that the proposed hybrid image segmentation can achieve satisfactory performance under wavy water environment.

6.7.2.4 Impact of Training Dataset Size

In this set of experiments, we study the impact of training dataset size N. Section 6.3.4 discusses the trade-off between estimation accuracy and system overhead caused by N. In Samba, the setting of N determines the size of the buffer that stores the training data for mapping models learning. Figure 6.18 plots the estimation errors of k_i and h_i under various settings of N. It can be observed that the average estimation accuracy of both k_i and h_i increases with N. This is because a larger N improves the diversity of training data, resulting in mapping models that account for a wider range of dynamics. Moreover, we find that the variance of estimation errors decreases with N. Meanwhile, the setting of N contributes to the system overhead. As shown in Figure 6.19, the computation delay of hybrid segmentation increases with N. This is mainly because the visual

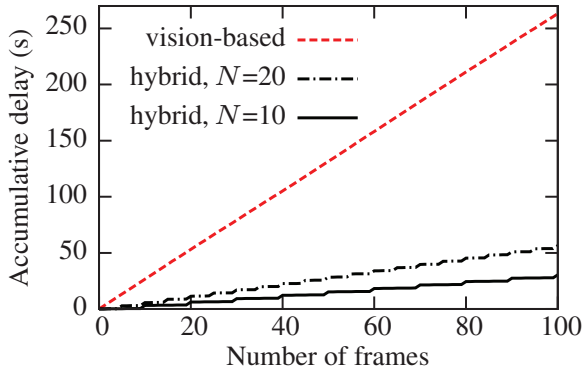


Figure 6.19: Impact of training dataset size N on computation delay.

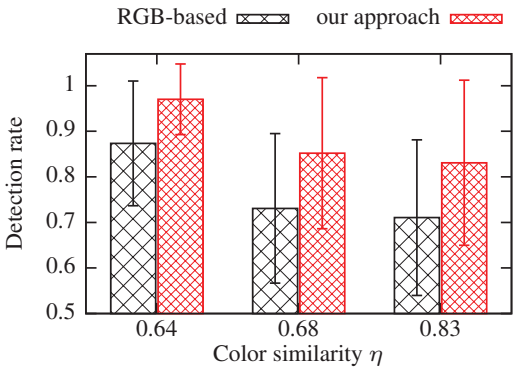


Figure 6.20: Impact of color similarity η on detection performance.

features are extracted more frequently under a larger N. The computation overhead of mapping models learning also increases with N. According to our measurements, such delay is in the order of millisecond and thus has limited impact on computation delay. Figure 6.19 also compares the accumulative computation delay of vision-based and hybrid approaches. We can see that the delay is notably shortened with the hybrid image segmentation.

6.7.2.5 Impact of Color Similarity

This set of experiments evaluate the impact of color similarity η between the TAP and water area on detection performance. We adopt three boards in different colors as TAB patches. For each colored patch, we conduct 10 runs of experiments. For each run, we calculate the detection rate by lower bounding the positive overlap ratio at 0.8. Figure 6.20 shows the detection rate under various η, where the error bar represents standard deviation. We can see that the detection rate decreases with η. As η quantifies the proximity of two colors in the cylindrical HSV model, the increase in η reduces the likelihood of identifying the TAP patches from water. We can also find that our TAP detection algorithms achieve satisfactory detection performance under high color similarity. For example, when $\eta = 0.83$, the detection rate is about 84%. Moreover, our approach

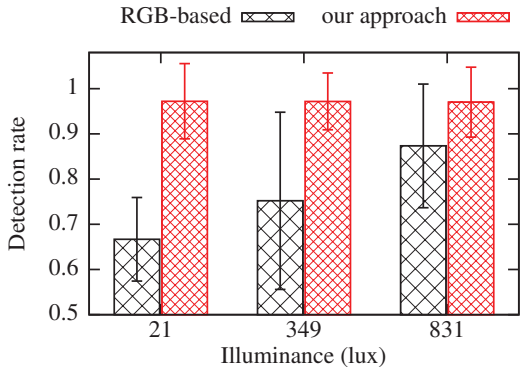


Figure 6.21: Impact of illuminance on detection performance.

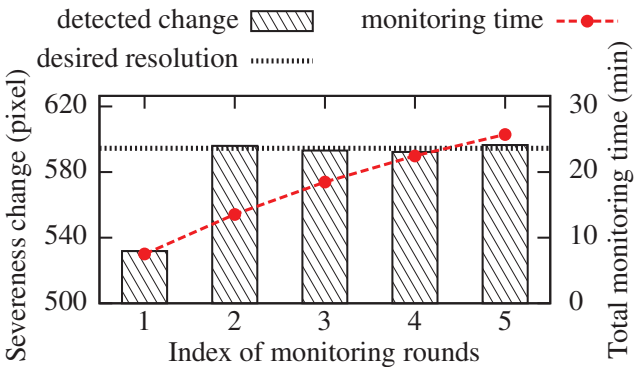


Figure 6.22: Detected severeness change and total monitoring time versus index of rotation.

yields consistently better detection performance than the RGB-based baseline since the HSV model differentiates similar colors more effectively.

6.7.2.6 Impact of Illuminance

As discussed in Section 6.4.1, we adopt two designs to enhance Samba's robustness to illuminance change. First, the color histogram is built in the HSV model and excludes the value channel that is sensitive to lighting condition. Second, we normalize the color histogram before applying it to back projection. In the experiments, we create various lighting conditions by using different combinations of three compact fluorescent lamps and a Power Tech LED light [72]. We use the patch with $\eta = 0.64$ and conduct 10 runs of experiments for each lighting condition. Figure 6.21 plots the detection rate under various lighting conditions, where the reported illuminance is measured by the phone's built-in light sensor. We can see that our TAP detection algorithms maintain consistently satisfactory performance under different illuminance levels, while the RGB-based baseline is sensitive to illuminance change.

6.7.2.7 Integrated Evaluation

In this set of experiments, all modules of Samba, i.e., hybrid image segmentation, TAP detection, and adaptive rotation scheduling, are integrated and evaluated. Moreover, on the control board of robotic fish, we implement a closed-loop PID controller that adaptively controls the tail beats based on the discrepancy between the scheduled and actual orientations. We imitate the evolution of TAP by gradually uncovering the board's surface (with $\eta = 0.64$). Based on the angular view of Galaxy, we select the camera orientations as $\{0, \pi/4, \pi/2, 3\pi/4, \pi\}$ with respect to the tank's side to ensure that the semi-circle can be fully covered when Samba slides over these orientations. At $t = 0$, Samba is deployed in parallel to the tank's side and starts the aquatic monitoring. We set the initial average rotation speed as 6 deg/min. Therefore, the first monitoring round has an interval of 7.5 minutes. After the first round, Samba adaptively schedules its rotation speed to maintain the detected severeness change at the desired level, which is set to be 595 pixels, until it is parallel to the tank's side again. Throughout the experiments, Samba achieves the detection rate of around 97% and false alarm rate of about 5%. Figure 6.22 plots the detected severeness changes in the first 5 rounds, which are well maintained at the desired level. Moreover, Figure 6.22 shows the total monitoring time versus index of robot rotation. We can see that the monitoring time varies across rounds, and it has a 5-minute length on average. During the experiment, we also find that our PID controller can generally direct Samba to the desired orientation with an error lower than 7°.

6.7.3 Trace-Driven Simulations

We evaluate the adaptive rotation control of Samba through trace-driven simulations, given the difficulty in accessing an actively evolving TAP. To simulate realistic settings, we collect Samba

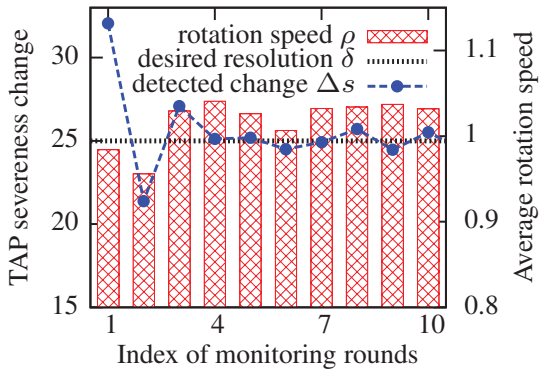


Figure 6.23: Detected severeness change and Samba rotation speed versus index of rotation.

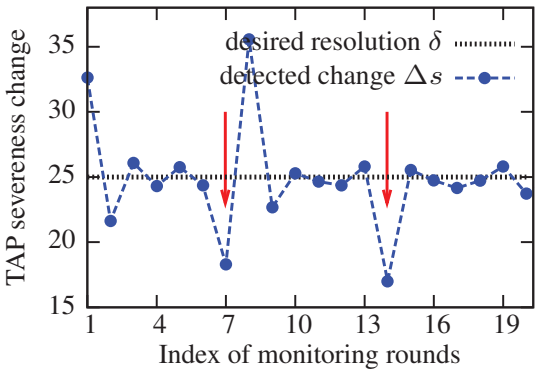


Figure 6.24: Impulsive and step responses (at the 7^{th} and 14^{th} rounds) of rotation control algorithm.

rotation errors and real chemical diffusion process. First, the error traces of rotation are collected using our prototype in the water tank. We measure the rotation error by the discrepancy between the desired orientation and actual heading direction of Samba. Second, we record the diffusion traces of Rhodamine-B, which is a solution featuring red color, in saline water using a camera. Hence, the evolution of diffusion process is characterized by the expansion of the red area over time. The diffusion traces contain the detected Rhodamine-B area and corresponding timestamp.

In the simulations, Samba monitors TAP within its circular surveillance region. According to our measurements, the camera on Galaxy has an angular view of $5\pi/18$. Hence, we partition the surveillance region into 8 sectors such that a full coverage can be achieved by a complete scan. For each rotation, the actual direction is set based on the collected error traces and thus is subjected to errors. For each orientation, the monitoring interval is determined by the scheduled rotation speed. We use the collected diffusion traces as severeness measurements, based on which the robot estimates the dynamics of TAP and schedules the rotation speed. Other settings include the desired resolution $\delta = 25$ and controller coefficient $c = 0.85$.

Figure 6.23 plots the detected severeness change Δs in the first 10 rounds. We can see that Δs quickly converges to the desired severeness resolution δ. Figure 6.23 also shows the scheduled

rotation speed, which is scheduled based on the current dynamics of TAP and δ. We further evaluate the response of our algorithm to the sudden changes in TAP evolution. Specifically, we artificially reduce the severeness measurements by 30% at the 7^{th} rotation (i.e., the left arrow in Figure 6.24) and continuously since the 14^{th} rotation (i.e., the right arrow in Figure 6.24). For both types of changes, our algorithm converges within a few rounds of rotation. Therefore, the proposed algorithm can effectively adapt the rotation of Samba to the evolving dynamics of TAP.

6.8 Conclusion

This chapter presents Samba – a novel aquatic robot designed for monitoring harmful aquatic processes such as oil spill and HABs. Samba integrates an off-the-shelf Android smartphone and a robotic fish. Samba features hybrid image segmentation, TAP detection, and adaptive rotation control algorithms. The hybrid image segmentation effectively reduces the false detections by removing the non-water area and significantly decreases the overhead of continuous visual sensing by integrating inertial sensing via a learning-based approach. The TAP detection algorithms are lightweight and robust to various environment and system dynamics such as illuminance change and camera noise. The adaptive rotation control enables Samba to maintain the desired monitoring granularity on the spatiotemporally evolving TAP by adjusting the rotation speed.

Chapter 7

Conclusion

This dissertation explores four representative problems in aquatic environment monitoring. First, we propose an accuracy-aware approach to profiling an aquatic diffusion process such as oil spill. In our approach, the robotic sensors collaboratively profile the characteristics of a diffusion process including its source location, discharged substance amount, and evolution over time. In particular, the robotic sensors reposition themselves to progressively improve the profiling accuracy. We formulate a novel movement scheduling problem that aims to maximize the profiling accuracy subject to limited sensor mobility and energy budget. To solve this problem, we develop an efficient gradient-ascent-based algorithm and a near-optimal dynamic-programming-based algorithm.

Second, we present a novel approach to reconstructing a spatiotemporal aquatic field such as HABs. Our approach features a rendezvous-based mobility control scheme where robotic sensors collaborate in the form of a swarm to sense the aquatic environment in a series of carefully chosen rendezvous regions. We design a novel feedback control algorithm that maintains the desirable level of wireless connectivity for a sensor swarm in the presence of significant environment and system dynamics. Moreover, information-theoretic analysis is used to guide the selection of rendezvous regions so that the reconstruction accuracy is maximized subject to the limited sensor mobility.

Third, we develop a vision-based, cloud-enabled, low-cost, yet intelligent solution to aquatic debris surveillance. This approach features real-time debris detection and coverage-based rotation scheduling algorithms. Specifically, the image processing algorithms for debris detection are

specifically designed to address the unique challenges in aquatic environment, e.g., constant camera shaking due to waves. The rotation scheduling algorithm provides effective coverage of sporadic debris arrivals despite camera's limited angular view. Moreover, we design a dynamic task offloading scheme to offload the computation-intensive processing tasks to the cloud for battery power conservation.

Finally, we design Samba – an aquatic surveillance robot that integrates an off-the-shelf Android smartphone and a robotic fish for general water surface monitoring. Using the built-in camera of on-board smartphone, Samba can detect spatially dispersed aquatic processes. To reduce the excessive false alarms caused by the non-water area, Samba segments the captured images and performs target detection in the identified water area only. We propose a novel approach that leverages the power-efficient inertial sensors on smartphone to assist the image processing. Samba also features a set of lightweight and robust computer vision algorithms, which detect harmful aquatic processes based on their distinctive color features.

Using our prototype systems, we conduct extensive field experiments, lab experiments, and trace-driven simulations. The evaluation results demonstrate that the proposed algorithms can effectively deal with various dynamics in aquatic environment and achieve satisfactory sensing performance. Moreover, they incur low computation overhead on resource-constrained sensing platforms, meeting the real-time requirements for aquatic environment monitoring.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] T. Abdelzaher, Y. Diao, J. Hellerstein, C. Lu, and X. Zhu, "Introduction to control theory and its application to computing systems," in *Performance Modeling and Engineering*. Springer, 2008, pp. 185–215.

[2] S. Alag, K. Goebel, and A. Agogino, "A methodology for intelligent sensor validation and fusion used in tracking and avoidance of objects for automated vehicles," in *Proceedings of the American Control Conference*, 1995, pp. 3647–3653.

[3] J. Ammerman and W. Glover, "Continuous underway measurement of microbial ectoenzyme activities in aquatic ecosystems," *Marine Ecology Progress Series*, vol. 201, pp. 1–12, 2000.

[4] Android 2.2 highlights. http://developer.android.com/guide/topics/sensors_position.

[5] ART and Dalvik. http://source.android.com/devices/tech/dalvik/art.html.

[6] J. Baras, X. Tan, and P. Hovareshti, "Decentralized control of autonomous vehicles," in *Proceedings of the 42nd IEEE Conference on Decision and Control*, 2003, pp. 1532–1537.

[7] D. Barrett, M. Triantafyllou, D. Yue, M. Grosenbaugh, and M. Wolfgang, "Drag reduction in fish-like locomotion," *Journal of Fluid Mechanics*, vol. 392, no. 1, pp. 183–212, 1999.

[8] J. Blum, D. Greencorn, and J. Cooperstock, "Smartphone sensor reliability for augmented reality applications," in *Proceedings of the 10th International Conference on Mobile and Ubiquitous Systems: Computing, Networking, and Services*, 2013, pp. 127–138.

[9] R. Boland and M. Donohue, "Marine debris accumulation in the nearshore marine habitat of the endangered Hawaiian monk seal," *Marine Pollution Bulletin*, vol. 46, no. 11, pp. 1385–1394, 2003.

[10] B. Boyd, *Introduction to Algae*. Prentice Hall, 2003.

[11] C. Brekke and A. Solberg, "Oil spill detection by satellite remote sensing," *Remote Sensing of Environment*, vol. 95, no. 1, pp. 1–13, 2005.

[12] R. Burkard, M. Dell'Amico, and S. Martello, *Assignment Problems, Revised Reprint*. Society for Industrial and Applied Mathematics, 2009.

[13] California Environmental Preotection Agency. http://waterboards.ca.gov/water_issues.

[14] Y. Chen and A. Terzis, "On the implications of the log-normal path loss model: an efficient method to deploy and move sensor motes," in *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2011, pp. 26–39.

[15] J.-C. Chin, D. Yau, N. Rao, Y. Yang, C. Ma, and M. Shankar, "Accurate localization of low-level radioactive source under noise and measurement errors," in *Proceedings of the 6th ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2008, pp. 183–196.

[16] J. Crank, *The Mathematics of Diffusion*. Oxford University Press, 1975, vol. 2, no. 3.

[17] Cyclops-7 User's Manual. http://turnerdesigns.com.

[18] K. Dantu, B. Kate, J. Waterman, P. Bailis, and M. Welsh, "Programming micro-aerial vehicle swarms with karma," in *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2011, pp. 121–134.

[19] J. Davies, J. Baxter, M. Bradley, D. Connor, J. Khan, E. Murray, W. Sanderson, C. Turnbull, and M. Vincent, *Marine Monitoring Handbook*. Joint Nature Conservation Committee, 2001.

[20] C. Detweiler, M. Doniec, M. Jiang, M. Schwager, R. Chen, and D. Rus, "Adaptive decentralized control of underwater sensor networks for modeling underwater phenomena," in *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2010, pp. 253–266.

[21] J. Dolan, G. Podnar, S. Stancliff, E. Lin, J. Hosler, T. Ames, J. Moisan, T. Moisan, J. Higinbotham, and A. Elfes, "Harmful algal bloom characterization via the telesupervised adaptive ocean sensor fleet," Robotics Institute, Carnegie Mellon University, Tech. Rep., 2007.

[22] R. Duda, P. Hart, and D. Stork, *Pattern Classification*. Wiley, 2012.

[23] M. Dunbabin and A. Grinham, "Experimental evaluation of an autonomous surface vehicle for water quality and greenhouse gas emission monitoring," in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, 2010, pp. 5268–5274.

[24] A. Elliott, "Shear diffusion and the spread of oil in the surface layers of the North Sea," *Ocean Dynamics*, vol. 39, no. 3, pp. 113–137, 1986.

[25] C. Eriksen, J. Osse, R. Light, T. Wen, T. Lehman, P. Sabin, J. Ballard, and A. Chiodi, "Seaglider: a long-range autonomous underwater vehicle for oceanographic research," *IEEE Journal of Oceanic Engineering*, vol. 26, no. 4, pp. 424–436, 2001.

[26] M. Faulkner, M. Olson, R. Chandy, J. Krause, M. Chandy, and A. Krause, "The next big one: detecting earthquakes and other rare events from community-based sensors," in *Proceedings of the 10th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, 2011.

[27] Fitbit. http://fitbit.com.

[28] T. Gandhi and M. Trivedi, "Pedestrian protection systems: issues, survey, and challenges," *IEEE Transactions on Intelligent Transportation Systems*, vol. 8, no. 3, pp. 413–430, 2007.

[29] Gaussian surface fit. http://bit.ly/1jph1yB.

[30] M. Gennaro and A. Jadbabaie, "Decentralized control of connectivity for multi-agent systems," in *Proceedings of the 45th IEEE Conference on Decision and Control*, 2006, pp. 3628–3633.

[31] Gliding robotic fish. http://nbcnews.to/1fGAomj.

[32] GNU Scientific Library (GSL). http://gnu.org/software/gsl.

[33] HABs and Lake Mendota. http://blooms.uwcfl.org/mendota.

[34] T. Hao, R. Zhou, and G. Xing, "COBRA: color barcode streaming for smartphone systems," in *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2012, pp. 85–98.

[35] T. He, J. Stankovic, C. Lu, and T. Abdelzaher, "SPEED: a stateless protocol for real-time communication in sensor networks," in *Proceedings of the 23rd International Conference on Distributed Computing Systems (ICDCS)*, 2003, pp. 46–55.

[36] R. Hecky and P. Kilham, "Nutrient limitation of phytoplankton in freshwater and marine environments: a review of recent evidence on the effects of enrichment," *Limnology and Oceanography*, vol. 33, no. 4, pp. 796–822, 1988.

[37] S. Hemminki, P. Nurmi, and S. Tarkoma, "Accelerometer-based transportation mode detection on smartphones," in *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems (SenSys)*, no. 13, 2013.

[38] B. Hoadley, "Asymptotic properties of maximum likelihood estimators for the independent not identically distributed case," *The Annals of Mathematical Statistics*, vol. 42, no. 6, pp. 1977–1991, 1971.

[39] C. Hu, F. Müller-Karger, C. Taylor, D. Myhre, B. Murch, A. Odriozola, and G. Godoy, "MODIS detects oil spills in Lake Maracaibo, Venezuela," *Eos, Transactions, American Geophysical Union*, vol. 84, no. 33, pp. 313–319, 2003.

[40] Hurricane Sandy. http://usat.ly/LWoxTI.

[41] J. Illingworth and J. Kittler, "A survey of the Hough transform," *Computer Vision, Graphics, and Image Processing*, vol. 44, no. 1, pp. 87–116, 1988.

[42] A. Jain, *Fundamentals of Digital Image Processing*. Prentice Hall, 1989, vol. 3.

[43] P. Jain, J. Manweiler, A. Achary, and K. Beaty, "FOCUS: clustering crowdsourced videos by line-of-sight," in *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems (SenSys)*, no. 8, 2013.

[44] 2011 Japan tsunami. http://marinedebris.noaa.gov/tsunamidebris.

[45] G. Judd, X. Wang, and P. Steenkiste, "Efficient channel-aware rate adaptation in dynamic environments," in *Proceedings of the 6th International Conference on Mobile Systems, Applications and Services (MobiSys)*, 2008, pp. 118–131.

[46] S. Kako, A. Isobe, and S. Magome, "Low altitude remote-sensing method to monitor marine and beach litter of various colors using a balloon equipped with a digital camera," *Marine Pollution Bulletin*, vol. 64, no. 6, pp. 1156–1162, 2012.

[47] A. Krause, C. Guestrin, A. Gupta, and J. Kleinberg, "Near-optimal sensor placements: maximizing information while minimizing communication cost," in *Proceedings of the 5th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, 2006, pp. 2–10.

[48] A. Krause, A. Singh, and C. Guestrin, "Near-optimal sensor placements in gaussian processes: theory, efficient algorithms and empirical studies," *Journal of Machine Learning Research*, vol. 9, pp. 235–284, 2008.

[49] P. Kulkarni, D. Ganesan, P. Shenoy, and Q. Lu, "SensEye: a multi-tier camera sensor network," in *Proceedings of the 13th ACM International Conference on Multimedia (MM)*, 2005.

[50] G. LaRoche, R. Eisler, and C. Tarzwell, "Bioassay procedures for oil and oil dispersant toxicity evaluation," *Journal of Water Pollution Control Federation*, pp. 1982–1989, 1970.

[51] N. Leonard and J. Graver, "Model-based feedback control of autonomous underwater gliders," *IEEE Journal of Oceanic Engineering*, vol. 26, no. 4, pp. 633–645, 2001.

[52] LG Optimus Net. http://amzn.to/Y4BvO9.

[53] C.-J. Liang, J. Liu, L. Luo, A. Terzis, and F. Zhao, "RACNet: a high-fidelity data center sensing network," in *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2009, pp. 15–28.

[54] S. Lin, J. Zhang, G. Zhou, L. Gu, J. Stankovic, and T. He, "ATPC: adaptive transmission power control for wireless sensor networks," in *Proceedings of the 4th ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2006, pp. 223–236.

[55] Linx GPS Receiver User's Manual. http://linxtechnologies.com.

[56] H. Liu, J. Li, Z. Xie, S. Lin, K. Whitehouse, J. Stankovic, and D. Siu, "Automatic and robust breadcrumb system deployment for indoor firefighter applications," in *Proceedings of the 8th International Conference on Mobile Systems, Applications and Services (MobiSys)*, 2010, pp. 21–34.

[57] D. Lymberopoulos and A. Savvides, "XYZ: a motion-enabled, power aware sensor node platform for distributed sensor network applications," in *Proceedings of the 4th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, 2005, pp. 449–454.

[58] T. Mace, "At-sea detection of marine debris: overview of technologies, processes, issues, and options," *Marine Pollution Bulletin*, vol. 65, no. 1, pp. 23–27, 2012.

[59] R. Maheshwari, S. Jain, and S. Das, "A measurement study of interference modeling and scheduling in low-power wireless networks," in *Proceedings of the 6th ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2008, pp. 141–154.

[60] N. March, M. Tosi, and N. March, *Introduction to Liquid State Physics*. World Scientific, 2002.

[61] R. Maree, P. Geurts, J. Piater, and L. Wehenkel, "Random subwindows for robust image classification," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2005, pp. 34–40.

[62] J. Matthes, L. Gröll, and H. Keller, "Source localization by spatially distributed electronic noses for advection and diffusion," *IEEE Transactions on Signal Processing*, vol. 53, no. 5, pp. 1711–1719, 2005.

[63] S. Murray, "Turbulent diffusion of oil in the ocean," *Limnology and Oceanography*, vol. 17, no. 5, pp. 651–660, 1972.

[64] A. Nehorai, B. Porat, and E. Paldi, "Detection and localization of vapor-emitting sources," *IEEE Transactions on Signal Processing*, vol. 43, no. 1, pp. 243–253, 1995.

[65] J. Nelder and R. Mead, "A simplex method for function minimization," *Computer Journal*, vol. 7, no. 4, pp. 308–313, 1965.

[66] Networked Aquatic Microbial Observing System). http://robotics.usc.edu/~namos.

[67] NOAA Marine Debris Clearinghouse. http://clearinghouse.marinedebris.noaa.gov.

[68] K. Ogata, *Discrete-Time Control Systems*. Prentice Hall, 1995.

[69] Ohio Environmental Protection Angency. http://epa.ohio.gov.

[70] Oil spills and disasters. http://infoplease.com/ipa/A0001451.html.

[71] T. Platt, C. Fuentes-Yaco, and K. Frank, "Marine ecology: spring algal bloom and larval fish survival," *Nature*, vol. 423, no. 6938, pp. 398–399, 2003.

[72] Power Tech LED. http://swflashlights.com.

[73] Project Kaisei. http://projectkaisei.org.

[74] A. Purohit, Z. Sun, F. Mokaya, and P. Zhang, "SensorFly: controlled-mobile sensing platform for indoor emergency response applications," in *Proceedings of the 10th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, 2011, pp. 223–234.

[75] L. Qiu, Y. Zhang, F. Wang, M. Han, and R. Mahajan, "A general model of wireless interference," in *Proceedings of the 13th Annual International Conference on Mobile Computing and Networking (MobiCom)*, 2007, pp. 171–182.

[76] M. Rahimi, R. Baer, O. Iroezi, J. Garcia, J. Warrior, D. Estrin, and M. Srivastava, "Cyclops: in situ image sensing and interpretation in wireless sensor networks," in *Proceedings of the 3rd ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2005, pp. 192–204.

[77] K. Ramachandran and C. Tsokos, *Mathematical Statistics with Applications*. Academic Press, 2009.

[78] T. Rappaport, *Wireless Communications: Principles and Practice*. Prentice Hall, 1996.

[79] C. Rasmussen, *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[80] S. Richardson and P. Green, "On Bayesian analysis of mixtures with an unknown number of components," *Journal of the Royal Statistical Society*, vol. 59, no. 4, pp. 731–792, 1997.

[81] L. Rossi, B. Krishnamachari, and J. Kuo, "Distributed parameter estimation for monitoring diffusion phenomena using physical models," in *Proceedings of the 1st IEEE International Conference on Sensing, Communication, and Networking (SECON)*, 2004, pp. 460–469.

[82] S. Ruberg, S. Brandt, R. Muzzi, N. Hawley, G. Leshkevich, J. Lane, T. Miller, and T. Bridgeman, "A wireless real-time coastal observation network," *Eos, Transactions, American Geophysical Union*, vol. 88, no. 28, pp. 285–286, 2007.

[83] D. Rudnick, R. Davis, C. Eriksen, D. Fratantoni, and M. Perry, "Underwater gliders for ocean research," *Marine Technology Society Journal*, vol. 38, no. 2, pp. 73–84, 2004.

[84] L. Shapiro and G. Stockman, *Computer Vision*. Prentice Hall, 2001.

[85] A. Singh, R. Nowak, and P. Ramanathan, "Active learning for adaptive mobile sensing networks," in *Proceedings of the 5th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, 2006, pp. 60–68.

[86] A. Singh, A. Krause, C. Guestrin, and W. Kaiser, "Efficient informative sensing using multiple robots," *Journal of Artificial Intelligence Research*, vol. 34, no. 1, pp. 707–755, 2009.

[87] J. Smith and S.-F. Chang, "VisualSEEk: a fully automated content-based image query system," in *Proceedings of the 5th ACM International Conference on Multimedia (MM)*, 1997, pp. 87–98.

[88] H. Solomon, *Geometric Probability*. Society for Industrial and Applied Mathematics, 1978, vol. 28.

[89] C. Song, Z. Qu, N. Blumm, and A.-L. Barabási, "Limits of predictability in human mobility," *Science*, vol. 327, no. 5968, pp. 1018–1021, 2010.

[90] Z. Song, Y. Chen, J. Liang, and D. Ucinski, "Optimal mobile sensor motion planning under nonholonomic constraints for parameter estimation of distributed systems," *International Journal of Intelligent Systems Technologies and Applications*, vol. 3, no. 3, pp. 277–295, 2007.

[91] Southern Regional Water Program. http://srwqis.tamu.edu.

[92] S. Srinivasan, K. Ramamritham, and P. Kulkarni, "Ace in the hole: adaptive contour estimation using collaborating mobile sensors," in *Proceedings of the 7th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, 2008, pp. 147–158.

[93] Stargate User's Manual. http://xbow.com.

[94] C. Stauffer and E. Grimson, "Adaptive background mixture models for real-time tracking," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 1999, pp. 246–252.

[95] R. Tan, H. Huo, J. Qian, and T. Fang, "Traffic video segmentation using adaptive-k Gaussian mixture model," in *Advances in Machine Vision, Image Processing, and Pattern Analysis*. Springer, 2006, pp. 125–134.

[96] X. Tan, "Autonomous robotic fish as mobile sensor platforms: challenges and potential solutions," *Marine Technology Society Journal*, vol. 45, no. 4, pp. 31–40, 2011.

[97] T. Teixeira, E. Culurciello, J. Park, D. Lymberopoulos, A. Barton-Sweeney, and A. Savvides, "Address-event imagers for sensor networks: evaluation and modeling," in *Proceedings of the 5th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, 2006, pp. 458–466.

[98] TelosB, IRIS, Imote2, and MICA2 User's Manuals. http://memsic.com.

[99] E. Tsotsas and A. Mujumdar, *Modern Drying Technology*. Wiley, 2009, vol. 2.

[100] J. Twigg, J. Fink, P. Yu, and B. Sadler, "RSS gradient-assisted frontier exploration and radio source localization," in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, 2012, pp. 889–895.

[101] U.S. Environmental Preotection Agency. http://water.epa.gov/type/oceb/marinedebris.

[102] I. Vasilescu, K. Kotay, D. Rus, M. Dunbabin, and P. Corke, "Data collection, storage, and retrieval with an underwater sensor network," in *Proceedings of the 3rd ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2005, pp. 154–165.

[103] S. Vijayakumaran, Y. Levinbook, and T. Wong, "Maximum likelihood localization of a diffusive point source using binary observations," *IEEE Transactions on Signal Processing*, vol. 55, no. 2, pp. 665–676, 2007.

[104] H. Wang, K. Yao, G. Pottie, and D. Estrin, "Entropy-based sensor selection heuristic for target localization," in *Proceedings of the 3rd ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, 2004, pp. 36–45.

[105] Y. Wang, J. Yang, H. Liu, Y. Chen, M. Gruteser, and R. Martin, "Sensing vehicle dynamics for determining driver phone use," in *Proceedings of the 11th International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2013, pp. 41–54.

[106] Y. Wang, R. Tan, G. Xing, X. Tan, J. Wang, and R. Zhou, "Spatiotemporal aquatic field reconstruction using robotic sensor swarm," in *Proceedings of the 33rd IEEE Real-Time Systems Symposium (RTSS)*, 2012, pp. 205–214.

[107] ——, "Spatiotemporal aquatic field reconstruction using cyber-physical robotic sensor systems," *ACM Transactions on Sensor Networks*, vol. 10, no. 4, p. 57, 2014.

[108] Y. Wang, R. Tan, G. Xing, J. Wang, and X. Tan, "Accuracy-aware aquatic diffusion process profiling using robotic sensor networks," in *Proceedings of the 11th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, 2012, pp. 281–292.

[109] ——, "Profiling aquatic diffusion process using robotic sensor networks," *IEEE Transactions on Mobile Computing*, vol. 13, no. 4, pp. 880–893, 2014.

[110] Y. Wang, R. Tan, G. Xing, J. Wang, X. Tan, and X. Liu, "Samba: a smartphone-based robot system for energy-efficient aquatic environment monitoring," in *Proceedings of the*

14th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN), 2015, pp. 262–273.

[111] Y. Wang, R. Tan, G. Xing, J. Wang, X. Tan, X. Liu, and X. Chang, "Aquatic debris monitoring using smartphone-based robotic sensors," in *Proceedings of the 13th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, 2014, pp. 13–24.

[112] J. Weimer, B. Sinopoli, and B. Krogh, "Multiple source detection and localization in advection-diffusion processes using wireless sensor networks," in *Proceedings of the 30th IEEE Real-Time Systems Symposium (RTSS)*, 2009, pp. 333–342.

[113] G. Werner-Allen, S. Dawson-Haggerty, and M. Welsh, "Lance: optimizing high-resolution signal collection in wireless sensor networks," in *Proceedings of the 6th ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2008, pp. 169–182.

[114] Y. Xu, J. Choi, and S. Oh, "Mobile sensor network navigation using gaussian processes with truncated observations," *IEEE Transactions on Robotics*, vol. 27, no. 6, pp. 1118–1131, 2011.

[115] C. You, N. Lane, F. Chen, R. Wang, Z. Chen, , T. Bao, Y. Cheng, M. Lin, L. Torresani, and A. Campbell, "CarSafe app: alerting drowsy and distracted drivers using dual cameras on smartphones," in *Proceedings of the 11th International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2013, pp. 13–26.

[116] B. Zhang and G. Sukhatme, "Adaptive sampling for estimating a scalar field using a robotic boat and a sensor network," in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, 2007, pp. 3673–3680.

[117] T. Zhao and A. Nehorai, "Distributed sequential bayesian estimation of a diffusive source in wireless sensor networks," *IEEE Transactions on Signal Processing*, vol. 55, no. 4, pp. 1511–1524, 2007.

[118] M. Zuniga and B. Krishnamachari, "Analyzing the transitional region in low power wireless links," in *Proceedings of the 1st IEEE International Conference on Sensing, Communication, and Networking (SECON)*, 2004, pp. 517–526.