Applied Estimation of Mobile Environments

Kevin Weekly
Alexandre Bayen, Ed.
Kristofer Pister, Ed.
Costas J. Spanos, Ed.
Steven Glaser, Ed.

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2014-32
http://www.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-32.html

April 28, 2014

Applied Estimation of Mobile Environments

by

Kevin Pu Weekly

A dissertation submitted in partial satisfaction of the
requirements for the degree of
Doctor of Philosophy

in

Electrical Engineering and Computer Sciences

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Associate Professor Alexandre M. Bayen, Chair
Professor Kristofer S. J. Pister
Professor Costas J. Spanos
Professor Steven D. Glaser

Spring 2014

Applied Estimation of Mobile Environments

Abstract

Applied Estimation of Mobile Environments

by

Kevin Pu Weekly

Doctor of Philosophy in Electrical Engineering and Computer Sciences

University of California, Berkeley

Associate Professor Alexandre M. Bayen, Chair

For many research problems, controlling and estimating the position of the mobile elements within an environment is desired. Realistic mobile environments are unstructured, but share a set of common features, such as position, speed, and constraints on mobility. To estimate within these real-world environments requires careful selection of the best-suited estimation tools and software and hardware technologies. This dissertation discusses the design and implementation of applied estimation infrastructures which overcome the challenges of real-world deployments.

Estimating the mobility of water within rivers and estuaries is a significant area of study considering the need for fresh water all over the world. The Floating Sensor Network is designed to enable Lagrangian measurements, from devices called drifters, in these areas which was previously infeasible to collect. Two new types of drifters are developed: a low-cost Android smartphone based drifter and a motorized active drifter. The Android drifter is economical, allowing dense sensor deployments at low cost. Since drifter studies in rivers are often beset by drifters becoming pushed onto the banks, the active drifter is able to avoid these obstacles by using a Hamilton-Jacobi safety control algorithm. Multiple field operational tests validate that the active drifters successfully avoid becoming trapped in difficult terrain. Field tests also validate the operation of the estimation solution as a whole, measuring the water flow via drifters and producing flow fields of the river.

The mobile environment of occupants within an office building is also studied extensively. This dissertation introduces the environmental sensing platform for indoor occupant studies. The platform includes a design of a battery-powered environmental sensor device and the communication architecture needed to collect data into a central repository. The sensor devices themselves communicate via WiFi technology and have a rich suite of sensors, including passive infrared, temperature, humidity, light level and acceleration. Electrical current consumption measurements from the sensors show that they can operate for over 5 years on a single battery. Discussed is how these sensors can be used for occupant tracking and occupant estimation, either via the on-board instruments, or instruments which are added to the devices via an expansion port.

A unified particle filter is proposed which can both estimate occupancy and track occupants within a building. This dissertation presents several prerequisite studies to motivate this direction: Two studies are performed to understand how occupancy and occupant activity affects measurable variables: particulate matter and CO_2. These variables are chosen as they are otherwise important for monitoring indoor air quality. Experimental studies show that there are indeed correlations between occupant activity and these variables. Furthermore, an estimator can be built which estimates the occupancy of a conference room, given CO_2 measurements. Our third study accomplishes occupant tracking using a particle filtering framework and signal strength measurements from a radio-based indoor positioning system. The implementation forms a basis from which to build the unified particle filter.

To Donna Lai Weekly:

my peer, teacher, student, friend, and life partner

towards our eternal study of the meaning of life.

Contents

List of Figures

List of Tables

Acknowledgments

I find myself taking a step back to appreciate how fortunate I have been throughout my graduate career, to have met so many brilliant, friendly, and encouraging people who have guided my progress, tested my ideas, and developed my principles in conducting science and engineering.

Alex Bayen is the kind of advisor who brings his experience to the table, but feels more like a partner than a boss. I attribute so much of my success in navigating academia and research to his candid advice. He finds the unique strengths in every one of his students and treats his research group like a family. Although Alex's students worked on two different projects: the Floating Sensor Network and the Connected Corridors project, everybody was quick to pick up a box of drifters to help the family. Therefore, I want to thank Andrew, Carlos, Chiheng, Christian, Jad, Jack, Jean-Baptiste, Jean-Bonoît, Jon, Leah, Mohammad, Nikos, Nolan, Olli-Pekka, Pierre-Henri, Qingfang, Samitha, Saurabh, and Timothy, for their contributions to the success of the Floating Sensor Network, their guidance, and their friendship.

I met Kris Pister at the end of my first semester at Berkeley and immediately identified with his down-to-earth attitude about engineering, stripping away politics and prejudices. Kris is also extremely sharp, and will immediately correct and/or extend the ideas of his students. Kris holds weekly group meetings (home of the famous "burrito list"), where I brought my ideas to be truly tested by Kris and his students, and where my creative juices were invigorated. A special thanks to Ankur, Fabien, Travis, Mike, Thomas, and Xavi for keeping me sharp as I grew in the EECS program.

I started working with Costas Spanos while he was chair of the EECS department and I was president of EEGSA. I was fortunate to work with him again, in a research capacity, as part of the SinBerBEST project. In both instances, Costas and I have worked together towards significant accomplishments. He has a keen sense of the big picture, and at the same time, letting each student know how important their individual contributions are towards accomplishing it. I have enjoyed working with my teammates from both the Berkeley and Singapore side of the project, Chris, Han, Ioannis, Jimmy, Komang, Krish, Ming, Shayaan, Yuxun, and Zhaoyi. It has been very rewarding watching them grow and develop their own scientific minds and careers.

I cannot list everybody in the Berkeley EECS department, but nonetheless, every student has done their part contributing towards the EECS culture which is decidedly laid-back and supportive. Many students, faculty, and staff have helped me and others throughout the program as my friends and advisers. As EEGSA president, I want to thank Bobby, my co-president for taking on the challenge with me, and Matt and Gireeja for their guidance, as well as all of the officers.

I am eternally thankful to my loving family. My wife Donna, truly completes me and led me to discover myself, celebrating and sympathizing with me, supporting me every step along my journey (even when I decide to get married and graduate in the same month). Finally, I would not be the person I am without my mom, Sheila, dad, Roger, and sister, Sara's, upbringing, their encouragement and guiding principles.

Chapter 1

Introduction

1.1 Position of the research

From predicting the weather by tracking campfire smoke, to tracking enemy movements to guess their intention, humanity has always placed great importance in tracking the motion of objects and people in their environment. These problems deal with estimating mobile elements of an intrinsically mobile environment. Nowadays, we augment our senses, and our reasoning abilities, with electronic sensors and mathematical frameworks, allowing us to understand these environments at a denser and deeper level. These sensors and techniques can be seen as tools which are selected, combined, interpreted and synthesized; it is novel application of these tools towards real-world environments which advances the field of applied estimation. This dissertation is an investigation, discussion, and inspiration for future research into mobile environments, achieved by finding appropriate and powerful mathematical estimation tools, collecting rich sets of experimental data, and solving the implementation challenges that arise in applying the tools to the data. The product of the studies conducted are realizable and informative (i.e. the output is immediately meaningful) architectures for sensing mobile environments.

Our research takes place in two settings. The first setting is comprised of rivers and estuaries, where we would like to understand the movement of water through these conduits. The sensing instrument we use is a mobile floating sensor, or *drifter*, which is device purposefully placed into the river and intended to match the water's movement. Since our estimation algorithms benefit from increased visibility of the environment, we leverage the ubiquity of low-cost Android smartphones to build numerous passive drifter units. However, river and estuarial environments present mobility challenges that have traditionally beset drifter research. Therefore, we also design a motorized active drifter for the purpose of navigating these regions while still functioning as a drifter. This requires the use of an advanced control method based on the Hamilton-Jacobi framework, a mathematical tool for which certain implementation challenges were overcome to be applied successfully. The passive and active drifters are both components of a realized architecture which collects and processes the measurements from the sensors. This architecture produces informative outputs in the form of flow fields, that is, the river speed and direction at each point on a 2-dimensional grid within the experimental region.

Our second research setting is comprised of indoor office spaces where occupants work and spend much of their daily routine. Whereas the mobile elements were designed by us in the drifter work, in this setting, the mobile elements are the occupants themselves. The motion and position of these occupants is important knowledge to the greater smart building community, notably to regulate energy usage and to satisfy indoor air quality requirements. We apply the strengths of both technological and mathematical advancements towards this estimation challenge. Technologically, we introduce a sensing device and a supporting architecture which enables wirelessly recording environmental variables needed as inputs to the mathematical tools. This sensing system is low cost, easily deployed and maintained, and in having a long battery lifetime, is suitable to be deployed permanently in areas which are valuable to be measured. Mathematically, we apply advanced estimation techniques to achieve an informative result, such as the position of an occupant, or the amount of occupant activity in an area. We also provide a thorough discussion of the future direction of the work conducted in the office setting: to provide a unified algorithm which incorporates information from heterogeneous data sources spread throughout the building. The fact that the algorithm can digest and leverage different types of data leads to an important practical advantage: our method can be applied to sensors which are already deployed, such as CO_2 and passive infrared (PIR) sensors.

We position the research presented in this dissertation as a description of the challenges overcome in applying estimation techniques towards these mobile environments. Our goal is twofold: first to advance the understanding of mobility in the river and office environments through our experimental and modelling work. Our second is to archive and pass on our experiences to other researchers who, like us, are building sensing architectures and studying mobile environments.

1.2 Components of mobile environments

In some communities, the term *mobile environment* indicates a troublesome scenario where the mobility of elements is undesired, but must somehow be accounted for. For example, in the wireless networking community, the mobility of participants in the network can wreck havoc as nodes enter and leave each others' transmission range [1]. Even wireless positioning techniques, which are intrinsically interested in the mobile element, experience a degradation of performance when mobility is introduced [2]. Nevertheless, mobile environments are realities of our world and studying them has a large potential for societal improvement, with applications including reducing energy consumption and managing fresh water resources.

Characteristic of mobile environments is the concept of *position*, i.e. the location of objects or physical phenomena referenced to points within another environment. We define a *stationary* environment as representing an "assumed" reference frame. In some fields of science, such as astronomy, the definition of the stationary environment is more ambiguous since the earth can be seen as moving within the solar system or galaxy. For this dissertation however, the *stationary* environment is assumed to be comprised of all objects and phenomena whose positions are fixed to the earth, such as buildings and terrain. The *mobile elements* are those which change their position (either intentionally or passively) over time, relative to the earth. Together, the stationary environment and the mobile elements within that environment comprise the mobile environment.

Since mobile elements change their position over time, there is also the concept of *velocity*, which is the first derivative of position with respect to time. As well, there is a concept of *acceleration*, which is the second derivative of position with respect to time. Both velocity and acceleration are vector-valued quantities which can be decomposed into scalar parts. Velocity is commonly decomposed into *speed* and *direction*, and acceleration is commonly decomposed into its 3-dimensional components X,Y, and Z (since acceleration sensors can measure these quantities).

In real mobile environments, there are constraints on the mobility of the mobile elements. There are two constraints commonly encountered: *obstacles* are positions which the mobile element cannot occupy or cross, such as a solid wall. The other constraint is limited *control authority*: bounds on the amount a mobile element can change in its position, speed, or velocity. In some cases, these constraints are problems that must be overcome, such as a swimming robot that is too weak to swim upstream. In other cases, we leverage the imposed constraints to add information to the problem, such as a tracking problem where knowledge of obstacles can indicate which positions do not need to be considered.

1.3 Estimation problems and frameworks

One research problem in mobile environments is to estimate the position of the mobile elements. Directly *tracking* the object involves creating a model such that the state of the model includes the object's position. Then we use mathematical techniques to estimate the value of the model's state. However, sometimes the application calls for other metrics, such has how many of the mobile objects are in a certain area. In this case, the estimation technique may not directly track the mobile elements' positions, rather it estimates an aggregate variable which is a function of their positions. An example is the occupancy estimation problem, where the goal is to count the number of occupants in each room of a building.

There are many well-developed estimation frameworks in use, and their success mainly depends on their applicability to the problem and how well they are applied and implemented. In this dissertation, we demonstrate the use of several of these frameworks, including Ensemble Kalman Filtering [3], observers of partial differential equations [4], and particle filtering [5]. In general, the estimators we consider are those which rely on a model, constructed such that the state contains a variable that we want to estimate the value of. These estimators rely on an iterative or continuous process of making predictions, using the model, and then correcting those predictions via observations collected by real-world sensors.

Another school of study which we do not consider, but is a promising avenue for estimation problems, is machine learning and data mining [6]. Some of these techniques do not require a model or understanding of the underlying physics. In general, a machine learning algorithm attempts to learn the "correct" answer for a given set of inputs, using a training set of pairs of known correct answers and inputs. For example, a machine learning algorithm could learn how to do multiplication by being given a large table of products and their factors. These algorithms are particularly powerful when a massive amount of training data is available. Therefore, we believe machine learning could prove most effective after a sensing infrastructure has been installed for a long time, collecting a large data set from which to learn. In applying the estimation algorithms mentioned by this dissertation, we design and construct sensing architectures along the way, which we hope can also speed development and use of alternative estimation techniques, such as machine learning.

1.4 Challenges of applied estimation

The challenge of applying model-based estimators to real-world deployments is to design the model so that it is efficiently computable, produces accurate and informative outputs, and takes input measurements that can actually be obtained. Therefore, we aim to use algorithms which execute at real-time speed or better, even for large environments, since time is not a controlled variable in a real-world deployment. We must also either use commercial off-the-shelf sensors which collect the desired environmental variables, or design custom sensors to gather the required variables.

As well as overcoming the previous challenges, we also have the goal of demonstrating a practical implementation. Therefore, we design architectures which are low-cost, robust, and mostly automated. This design philosophy is embodied in the designs of the Floating Sensor Network and the indoor environmental sensing platform. These practical constraints indicate limitations on the choice of sensing infrastructure. Clearly the cost of the hardware and mechanical components will have a large role in how valuable the contribution of the design is to society, since the infrastructure must actually be bought and used to convey its benefits. Along the same lines, if the design is prone to failure, or requires human intervention, it will incur a large maintenance and operations cost, rendering it impractical. Therefore, we include a number of solutions to the practical side of the applied estimation problem, where careful and novel design is shown to significantly reduce construction and operating costs. One example is the design of the passive floating sensor which uses an Android smartphone to significantly reduce construction costs.

1.5 Organization

The dissertation is organized as follows: Chapters 2 and 3 operate within the setting of rivers and estuaries, describing our solution to estimating the motion of water through the environment. In Chapter 2, we introduce the floating sensor network, a fleet of autonomous Lagrangian drifters, both motorized (active) and non-motorized (passive). Since a primary contribution of the active drifter design is its ability to avoid obstacles, we extend the discussion of the Hamilton-Jacobi safety control technique in Chapter 3.

The Chapters 4–6 operate within the setting of an office building, where the mobile elements are occupants rather than drifters, and the goal is to estimate the mobility of these occupants. As in the previous river setting, a sensing architecture must be built for collecting the measurements to support practical and permanent estimation. We present the environmental sensing platform in Chapter 4 as a reference design for such an architecture. A valuable direction for study in the smart building setting is occupancy estimation. In Chapter 5, we describe studies in relating occupancy and occupant activity to environmental variables (CO_2 and particulate matter concentration) both of which are measured for indoor air quality. Our final investigation, described by Chapter 6, covers occupant tracking, i.e. estimating the position of occupant(s) over time. We describe a method which tracks an indoor occupant and provide results using a radio-based positioning system as the input to the algorithm. We illustrate how sensor data from the environmental sensor platform can be used, motivated by experiments with an environmental sensing smart watch. This chapter also discusses a unified occupancy estimation and occupant tracking framework to fuse heterogeneous sources of data from building sensors using particle filters.

Finally, we conclude the dissertation in Chapter 7, discussing the contributions of the work towards applied estimation and we motivate future directions into these mobile environments.

Chapter 2

Mobile Floating Sensors

2.1 Introduction

A complex mobile environment of high societal importance are the natural aqueducts of our freshwater supplies, that is rivers, lakes, streams, and estuaries. Throughout history, the movement, storage, consumption and contamination of water resources have formed and torn human communities together and apart. Freshwater is especially responsible for the flourishing of California's population and rich agriculture. Natural and man-made systems of channels, dams and aqueducts deliver and store the freshwater, supplying the agricultural, industrial, and consumer needs of the state. These are essential for the state's survival, and are sometimes tested to its limits, such as the drought of early 2014, when the California Department of Water Resources (DWR), for the first time ever, prepared to cut all water deliveries to the State Water Project's (SWP) agricultural customers. During this time, many urban areas had to rely on their local groundwater storage for drinking water [7].

The Floating Sensor Network (FSN) project at the University of California Berkeley [8] hopes to improve the understanding of estuarial environments in California. We attempt to estimate the mobility of the environment by inventing novel sensor systems and algorithms which can tell us where water moves within the environment. Most of our studies have occurred in the Sacramento-San Joaquin River Delta region in which water is stored and transported from melted snow in the Sierra mountains to the SWP and Central Valley Project (CVP) which supplies over 23 million Californians. The delta also connects to the ocean and freshwater outflow is required prevent saltwater from intruding into the delta and threatening the local ecosystem, as well as contaminating the freshwater supply. Therefore, if too much water is drawn from the delta, allowing salt water to flow in from the ocean, it could have disastrous consequences for all of California. In some cases, water must be released from reservoirs for expressly this purpose [9]. Thus, understanding the way water is moving and how it reacts to policy decisions will be extremely valuable in efficiently and safely using this water system. We believe that the invention and demonstration of our sensor network is large step towards this understanding.

Existing infrastructure is in place to monitor mass-flows of water entering and leaving the water system via fixed sensor stations distributed at various water-transfer junctions in the delta. These stations provide insight into how much water is entering and exiting the system. However, when faced with the question of where a particular “piece” of water moves within the system, hydrodynamic models must be used to estimate missing information that is not directly sensed. This process introduces significant modelling errors when determining where the pieces of water are moving. The Lagrangian sensing techniques developed by the FSN enable measuring the motion directly by embedding sensors in the environment which match the movement of the water mass. This has applications in tracking salt water intrusion into sensitive zones, or tracking the spread of contaminants after a chemical spill. For example, a fleet of drifters could be deployed at a recent spill site and their positions would be a proxy for the spread of the contaminant.

We have built an architecture to autonomously collect, store, and analyze measurements from our custom-built *drifter* instruments, which are designed to float along with the river water. We build a fleet of 100 drifters in two configurations: an active, motorized configuration which can avoid obstacles, and a passive, non-motorized and low-cost configuration which is based on an Android mobile phone. Measurements are collected wirelessly and in real-time over the mobile phone network and received by a server which stores the measurements and also sends them to be processed by a supercomputing cluster where the estimation is performed. The end results come in the form of flow field estimates of the region, where the water velocity of the river at each point (on a 2 dimensional grid) is estimated.

This chapter is primarily a description of the design and capabilities of the FSN fleet. The culmination of this work and proof of applicability of the system was demonstrated on May 9, 2012, when the FSN team deployed 28 motorized, active drifters and 68 passive drifters in the Sacramento River near its junction with the Georgiana Slough, near the town of Walnut Grove, California. The operation demonstrated the communication, obstacle avoidance, navigation, and data-gathering capabilities of the FSN fleet, and gathered flow data for use in demonstrations of an online Ensemble Kalman Filter based assimilation using a high-performance computing cluster.

Drifting Lagrangian sensors

In situ sensing refers to sensing techniques where a device is in direct contact with the environmental phenomena it measures. In contrast, *remote* sensing refers to techniques like satellite imagery, in which measurements are taken from afar. *In situ* sensing in fluid environments is classified into *Eulerian* and *Lagrangian* techniques, using the terminology for the different reference frames in hydrodynamics. Eulerian sensors are fixed to the external reference frame, e.g., the river bank, and take measurements from the water as it moves by. Lagrangian sensors float freely in the fluid itself, and gather measurements about the water as it moves through the environment.

Lagrangian sensors are a proven technology for oceanographic environments, where Eulerian sensor stations are impractical to deploy. For different reasons, Lagrangian sensing is

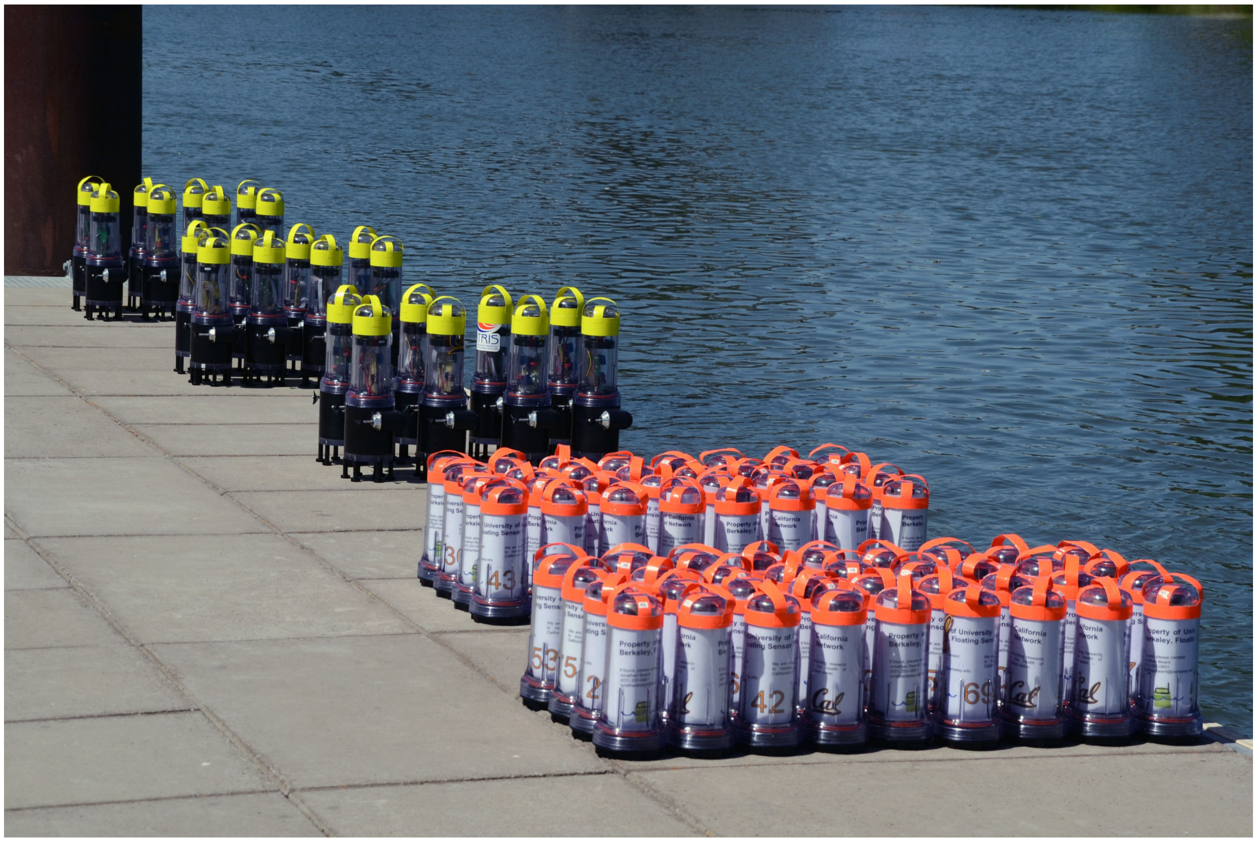


Figure 2.1: Drifter fleet on the Walnut Grove Public Dock on May 9, 2012 prior to deployment. Photo credit: Jérôme Thai

important to near-shore environments such as rivers and bay, as they better monitor the flow of freshwater and transport of constituents than Eulerian techniques. Example applications include assessing chemical spill or infrastructure failure vulnerabilities, planning reservoir release and gate control policies to mitigate saltwater intrusion, and monitor the effect of agriculture on freshwater supply. In contrast, Eulerian sensors are effective for directly measuring mass flow of water across certain points in the network, but must be integrated into a hydrodynamic model in order to track the transport of constituents.

Some Lagrangian sensors measure physical characteristics of the water in which they are immersed (e.g. dissolved constituents or temperature), while for others, the primary data gathered is its position over time. A well designed Lagrangian sensor should act like an “ideal particle” in the water flow, thus the time series of its position allows direct estimation of the velocity of the water which it traveled through. In the hydrodynamics literature, such sensor devices are called *drifters*. Drifter design has always been constrained by the positioning and communications technologies available. Modern oceanography began using drifters based on underwater acoustic communication in the 1950s [10]. Acoustic technology dominated until 1978, when the Argus satellite service gave oceanographic researchers a global location and data uplink system [11]. Power, cost, and size constraints meant that Argos-based drifters [12, 13, 14] were better suited to oceanography than inland environments like rivers and estuaries. In general, Lagrangian sensing has proven to be challenging in these environments due to shallow areas and relatively narrow water passageways which can trap drifters.

Global Positioning System (GPS) positioning and local radio frequency (RF) communication have enabled inland drifter studies, by allowing smaller units which can traverse shallow and obstacle-rich areas in a river system. GPS-carrying river drifters have been the focus of development by the FSN project [15] and other groups [16, 17]. Studies in regions with well-developed civilian infrastructure, like the continental United States, can take advantage of the mobile phone network for communications. The drifter design in this article is to our knowledge the first design to use commercial mobile phones not only as the communication system, but as the positioning and computation system as well.

2.2 Passive sensors using Android mobile phones

Motivation

Most of the drifters in our Floating Sensor fleet are our *Android drifters* [18], named as they are based on the Android platform [19], whereas all of our previous drifter designs relied heavily on custom, microcontroller-based designs. These custom designs required many months of development and debugging hardware components, and often required that features be removed from the final product due to cost or time constraints. Thus, the proliferation of Android smartphones offered us a relatively low-cost package offering the following features:

- All of the electronic functionalities required for a Lagrangian drifter– Positioning (via internal GPS), long-range Internet communication (via the mobile connection), and user interface (via the touch screen).
- A high-level programming interface (JAVA) and API for accessing the needed functionalities.
- Industry-tested hardware and software libraries, significantly improving reliability.
- High-volume pricing, due to the proliferation of smartphones in the consumer space.

In exchange for these features, the choice of the Android platform over a custom solution has the following drawbacks:

- Interfacing external sensors is difficult, requiring an unsupported and unreliable use of the Universal Serial Bus (USB) connection and a complex interfacing board. This has since been address by official support via the Android Accessory Development

Kit (ADK), however, the solution is still more complex than interfacing sensors to a microcontroller.

- The power usage is significantly increased due to a high-powered processor and increased processing needed to execute JAVA code as well as the rest of the miscellaneous operating system tasks and applications which are concurrently running. Thus, we need to add an external battery to achieve multi-day lifetimes (fortunately, a battery satisfies a dual-purpose as a ballast weight).
- We have significantly less control of the processor due to running in user-space in an operating system. Our application could be delayed or even stopped from other concurrent processes running on the phone. The primary effect is increased latency and lack of real-time assurances, which hinder using the phone for real-time control purposes (if the drifter were motorized), and possibly less reliability if our process was unexpectedly terminated, although we have never encountered this happening.

Ultimately, we recognized that many of our drifter studies could benefit from high numbers of floating sensors, which, due to cost, could not be met with custom hardware drifters. Thus, these Android drifters were designed to provide numerous sensors for studies in non-hostile environments, where obstacles posed less of a risk. Our smaller fleet of motorized drifters, using custom hardware, were used for those environments that proved too operationally difficult for these Android drifters to be used in.

Physical form of the Android drifter

A real-time Lagrangian drifter has two basic intelligence requirements: it can sense its own position and it report it to a central location. Since these are both satisfied by consumer Android mobile phones, we design an enclosure to hold one of these phones. Similar to our work with the active sensors, we identified three considerations when designing the form factor: First, as a Lagrangian flow sensor, it must suitably match the river's local velocity. Secondly, the mass must be distributed such that the phone is oriented vertically (to ensure the best GPS and communications reception). Finally, the phone should be nominally above the water line to ensure sufficient reception.

In the active drifter work, we found that a large, symmetric drag profile indicates good Lagrangian tracking performance, thus, the design is again a vertically oriented cylinder based on a water filter housing. In addition to the weight of the battery, an additional aluminum ballast weight was necessary to ensure a large separation between the center of mass (COM), at approximately 94 mm from the base, and center of buoyancy (COB), at approximately 126 mm from the base, while keeping the enclosure 90% submerged. Figure 2.2 shows how the COM and COB are distributed in the design.

The hull is primarily an ultraviolet (UV) stabilized polyvinyl chloride (PVC) water filter housing manufactured by Pentec. The bottom receiving cap of the water filter housing is machined from Delrin, which is known for its durability and ease of machining by computer

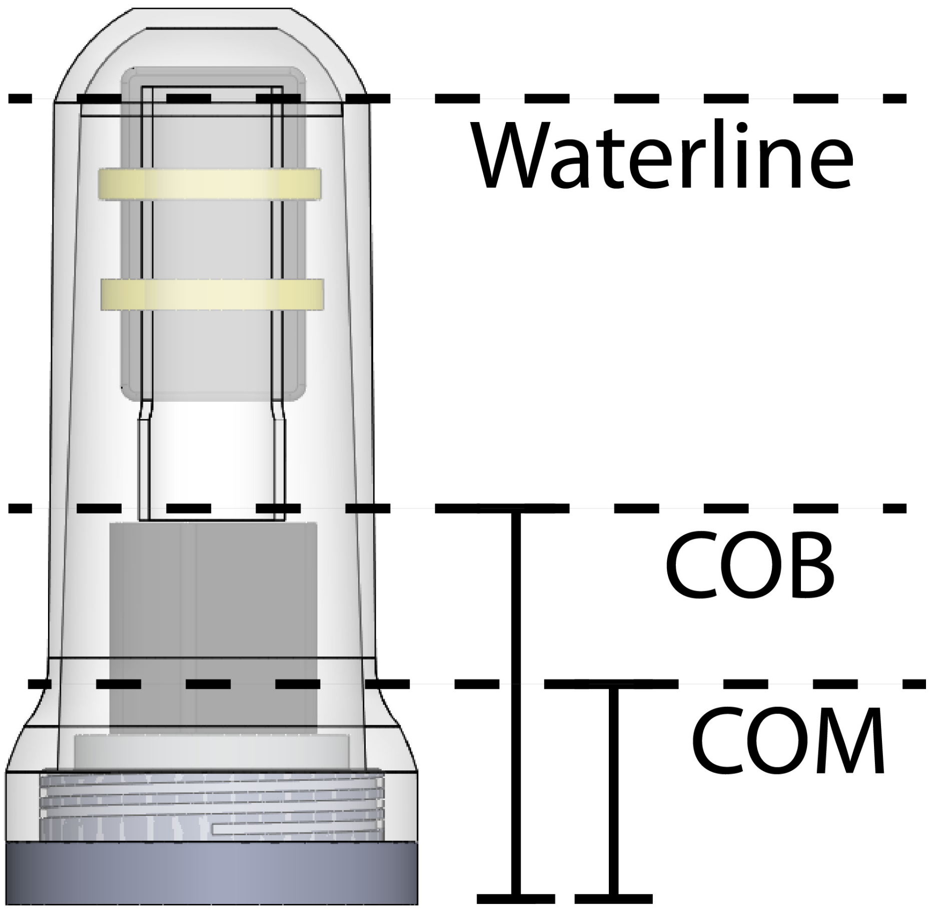


Figure 2.2: Sensor dimensions and COM and COB locations.

numeric controlled (CNC) equipment. Also, since it absorbs just 0.2% of its volume in water (24 hour submersion), it undergoes minimal changes in dimensional tolerances, which could be a source of leaks in other materials. The Delrin cap is given male threads which mate to the water filter housing. A silicone O-ring forms a watertight seal between the water filter housing and the Delrin cap and the two are tightened with two wrenches also manufactured by Pentek. The Delrin cap has notches which receive the special wrench. Internally, the Delrin cap has an impression for the ballast weight to reside in. Additionally there is a machined PVC tube which holds the phone in the desired position. Slots are machined into the PVC tube to hold the phone as well as to not impede airflow to help ventilate the phone. The overall assembly cost and diagram of components is displayed in Table 2.1 and Figure 2.3.

Our choice of battery was primarily driven by the typical residence time of water in the Sacramento-San Joaquin River Delta, approximately 48 hours. Thus we chose a battery which allowed 48 hours of continuous usage of the GPS, accelerometer, and Global System

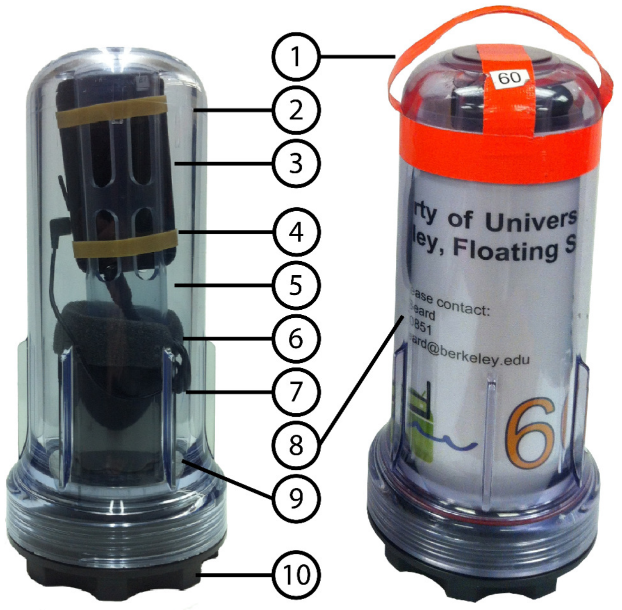


Figure 2.3: Android drifter assembly with part numbers.

for Mobile (GSM) systems on the phone. We found that this provides sufficient time for a lost drifter to be located and retrieved before the battery expires. We chose to use a high energy-density Li-Ion, 115 Watt-hour (at 1 Amp current draw) battery pack, providing, via an embedded circuit board, regulated 5 Volts to a micro Universal Serial Bus (USB) cable which plugs into the charging port of the phone.

Android software

Overview

The software for the Android Drifter was written in Java, following the model of the Android platform's Application Programming Interface (API). The Android platform distinguishes two types of programs: *activities* and *services*. Activities are intended to be user-interactive, thus having a high execution priority, but can be terminated by the operating system if they lose focus in order to regain their resources. Examples of activities are web browsers or games. Services, on the other hand, are typically given less priority, but are the last to be terminated when the operating system is low on memory. Services run in the background and interaction with them must be through an activity. Examples of services are music players, battery monitors, or email notification applications.

For our application we implemented the following three programs:

- *AndroidDrifterService*: Encompasses the main functionality of the Android Drifter, periodically transmitting and logging the last GPS position along with the valid flag

#	Item	Price	Quantity
1	Duct tape handle	$1	1
2	Upper PVC hull	$25	1
3	Motorola DEFY	$175	1
4	Rubber bands	$1	2
5	PVC phone holster	$15	1
6	Foam disk	$1	1
7	LiPo battery	$250	1
8	If-found placard	$1	1
9	Aluminum disk	$10	1
10	Delrin base	$50	1
	Total	$530	11

Table 2.1: Android drifter bill of materials

indicating if the unit is upright.

- *AndroidDrifterActivity*: Provides the user buttons to start and stop the service and status indicators to determine if data is being delivered properly.
- *ConfigureActivity*: Allows the user to configure the parameters (server address, port, update rate, etc.) of the service.

A simplified Unified Modeling Language (UML) diagram is provided in Figure 2.4. In the rest of this section, we describe the functionalities of the AndroidDrifterService in more detail:

GPS location collection

To collect the GPS location of the device, we used the LocationManager API class and related libraries. We call the API by requesting location change updates to a custom callback method and ask for minimum interval to receive updates. Thus, our callback method is called about every second with a new location object. The latitude and longitude coordinates are converted to UTM coordinates and store temporarily into the *last_loc* variable.

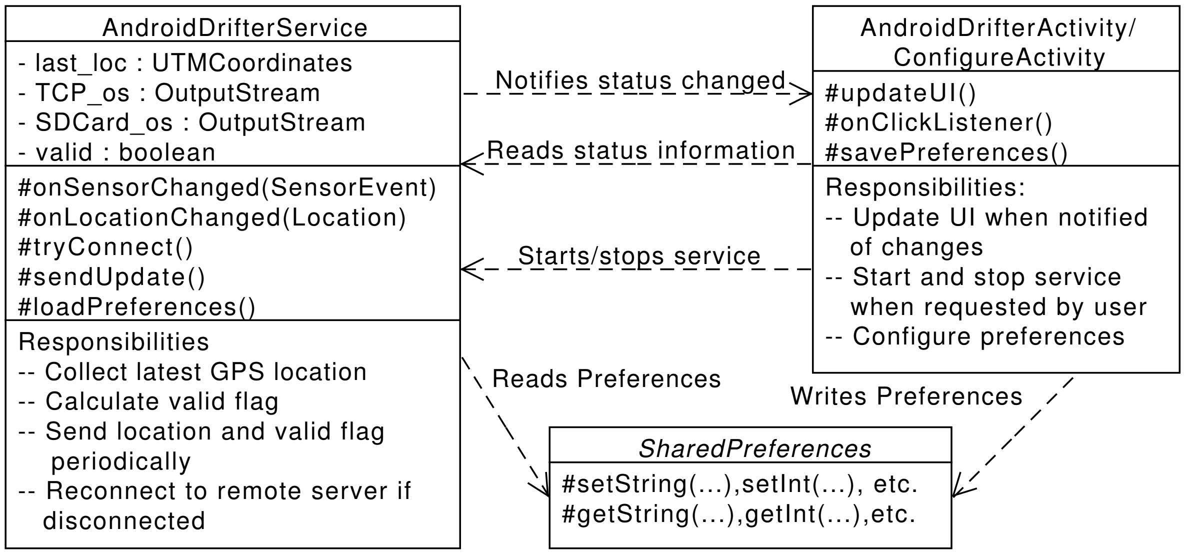

AndroidDrifterService
- last_loc : UTMCoordinates
- TCP_os : OutputStream
- SDCard_os : OutputStream
- valid : boolean
#onSensorChanged(SensorEvent)
#onLocationChanged(Location)
#tryConnect()
#sendUpdate()
#loadPreferences()
Responsibilities
-- Collect latest GPS location
-- Calculate valid flag
-- Send location and valid flag periodically
-- Reconnect to remote server if disconnected
Notifies status changed
Reads status information
Starts/stops service
Reads Preferences
Writes Preferences
AndroidDrifterActivity/ ConfigureActivity
#updateUI()
#onClickListener()
#savePreferences()
Responsibilities:
-- Update UI when notified of changes
-- Start and stop service when requested by user
-- Configure preferences
SharedPreferences
#setString(...),setInt(...), etc.
#getString(...),getInt(...),etc.

Figure 2.4: Simplified UML diagram showing interactions between software components of the Android drifter.

Valid flag calculation

The Android Drifter monitors its orientation to determine if its GPS position is valid, or drifting free in the water, or invalid, such as when it is being stored or transported. The Android phone inside the Android Drifter has a fixed orientation, therefore, the phone's orientation moves along with the orientation of the drifter as a whole. Given that the units are typically stored on their sides in containers and are upright when floating, we can use the phone's built in accelerometer to determine when the drifter is in either of these two states. The x-axis of the accelerometer points to the right side of the phone's screen, the y-axis points to the top of the phone, and the z-axis points out of the screen of the phone.

To collect the accelerometer readings from the phone we used the SensorManager API class and related libraries. Similar to the LocationManager class, we configured a callback to be provided updates at the fastest possible rate. On the incoming readings (raw_x, raw_y, raw_z) we implement a variant of the exponential moving average:

$$\begin{aligned} a_x &\leftarrow (1-\alpha dt)\cdot raw_x + \alpha dt \cdot a_x \\ a_y &\leftarrow (1-\alpha dt)\cdot raw_y + \alpha dt \cdot a_y \\ a_z &\leftarrow (1-\alpha dt)\cdot raw_z + \alpha dt \cdot a_z \end{aligned}$$

where dt is the time since the last sample and α is the degree of weighting decrease. Unlike the standard exponential moving average, samples are also scaled by the sampling interval because the Android operating system does not provide samples at a consistent rate. The weighting factor $\alpha = 0.001$ was chosen as a tradeoff between responsiveness in detecting placement in water and storage and effectiveness in filtering out bumps and shakes. We then compute the valid flag representing these two states as:

$$valid \leftarrow \begin{cases} 1 & \text{if} \quad a_y > |a_x| \text{ and } a_y > |a_z| \\ 0 & \text{otherwise.} \end{cases} \tag{2.1}$$

Therefore, if the phone's longest dimension is aligned with the gravity vector and upright, *valid* becomes 1, whereas if the phone is on one of its other sides, *valid* is 0.

Without the filter, we find that there are many false-negatives, i.e. the drifter reports $valid = 0$ when it should report $valid = 1$. The errors are caused by noisy accelerometer signals which can be attributed to wave action buffeting the drifter as well as normal noise from the sensor. We ran a study to determine the effect of the filter described above on the false-negative rate. For this test, the drifter was moored floating upright for 14 hours while logging accelerometer samples. Sampling intervals during this study had a mean of 203 mS and a standard deviation of 17 mS. We then calculate the raw and filtered pitch angles ϕ_{raw} and ϕ using the following formulae:

$$\phi_{raw} = \tan^{-1}\left(\frac{raw_y}{\sqrt{raw_x^2+raw_z^2}}\right)$$
$$\phi = \tan^{-1}\left(\frac{a_y}{\sqrt{a_x^2+a_z^2}}\right)$$

We evaluate the arctangent function $\tan^{-1}\left(\frac{y}{x}\right)$ using the two-argument function $\text{atan2}(y, x)$ common to many computer languages in order to place the angle in the correct quadrant.

The cumulative distribution functions of these two signals are plotted in Figure 2.5, along with the 45° threshold implied by Equation (2.1). From this, we calculate that the probability of false-negatives for the unfiltered data and filtered data is 10.6% and 0.583%, respectively. Thus, we conclude that the filtering is necessary to practically use the accelerometer as a tilt detector for determining whether the drifter is floating in the water or stored in its box.

Data transmission and Logging

The Timer API class is used to schedule a callback function periodically at a user-defined interval (typically 5s). In this callback, sendUpdate(), a type-value (TV) string is constructed, delimited by the forward slash character. An example of such a string is:

```
id/A78/ts/1334251708/x/630062.63/y/4233832.13/zn/10/valid/0/<LF>
```

where <LF> is the line feed character 0x0A. Here, the unit with ID A78 is reporting that on April 12, 2012 at 17:28:28 Greenwich Mean Time (1334251708 seconds since January 1, 1970), it was at UTM Coordinates 10N 630062.63 4233832.13, and that the data is invalid. Since this is about 10:30AM Pacific Standard Time, this corresponds to a time when the unit was still in storage in a container, waiting to be deployed.

The phone attempts to both write this string to a file on the phone's external memory card and send the string to a remote server over a transmission control protocol (TCP) connection. If either method is disrupted, the phone attempts to reestablish the file-handle or socket connection, respectively. When the phone does not have a GPS lock, it will still write data to the file and TCP connection at the defined interval, reusing the last GPS sample sent (or populate the fields with 0 if a lock was never achieved since the service was started). This assists later analysis as we can determine when GPS functionality was compromised and distinguish problems with the GPS from other parts of the program.

Applications

The Android drifters continue to be used for missions in the Sacramento-San Joaquin River Delta. For example, the FSN and Stanford University's Environmental Fluid Mechanics and Hydrology Department have each managed two missions independently. We have also provided the drifters to the University of California Davis and the United States Geographical Survey for extended periods to support their missions in the delta. This is a testament to the ease of use of the drifter platform: Only two Android drifters have been lost, thanks to

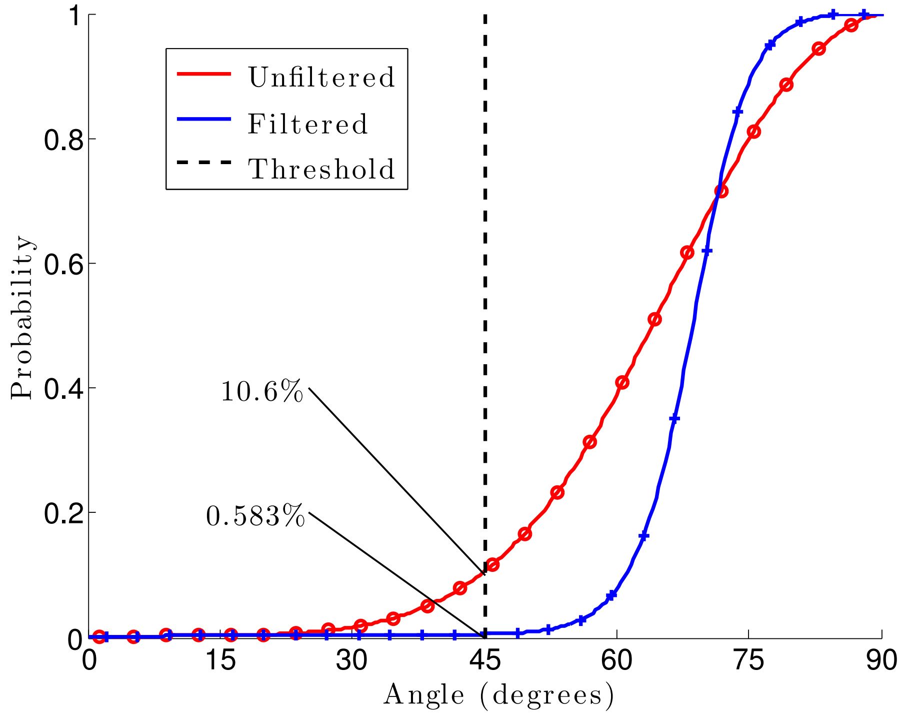


Figure 2.5: CDF of the angle calculated from the unfiltered (red circles) and filtered (blue pluses) accelerometer signal recorded by the Android phone. Unit was floating upright for the entirety of the 14 hour test. Black line shows the threshold value on which the software decides if the GPS position is valid.

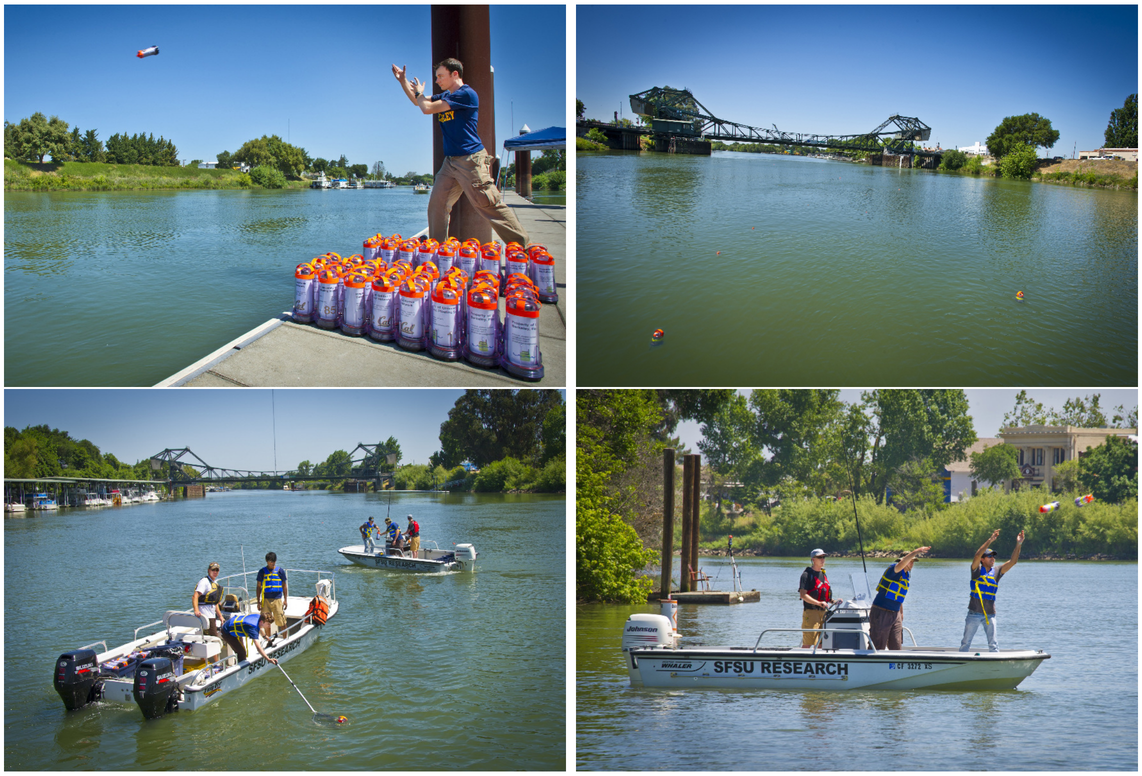


Figure 2.6: Android drifter usage.

the real-time tracking feature. This is also evidence for the durability of the drifters: No drifter has leaked in over 1300 hours of operation, and. In one instance, a drifter was lost for 320 days and later found and returned by a local boat operator. The drifter itself was very scratched but showed no signs of water leakage and the electronics were still operational after charging the battery.

Figure 2.6 shows pictures from a deployment of 70 Android drifters on May 9, 2012. The deployment lasted for 8 hours and spanned over 3.5 of the Sacramento River and the Georgiana Slough near Walnut Grove, California. The drifters travelled from the Walnut Grove dock, where they were deployed by hand, then down either the Sacramento River or Georgiana Slough, after which they were retrieved by boat teams armed with pole nets.

2.3 Actuated Generation 3 drifter design

The need for actuated drifters

Android drifters have proven to be useful tools for measuring the river environment, however, without supervision, the are likely to be rendered useless when they are stuck on obstacles and thus no longer track the river velocity. Thus, fixed, Eulerian infrastructure is traditionally used in the river environment. However, the vision of the FSN is long term operation of the drifters, thus, some mechanism is needed to avoid obstacles such as debris or the river shore. Thus, a motorized floating sensor is developed which has the ability to avoid these hazards, while being low-cost and manufacturable. In fact, the FSN is the first to design and produce a fleet of 40 such motorized drifters and demonstrate their effectiveness in a field experiment.

Our design included the electrical and mechanical requirements for an advanced safety control technique presented in Chapter 3. This safety control technique allows us to not only have the motorized drifters avoid obstacles that threaten their operation, but also specify a particular part of the river to proceed along. We demonstrated this ability near the river junction of the Sacramento river and the Georgiana Slough near Walnut Grove, CA [20].

Hardware

We designed the Generation 3 drifter to be modular with respect to functionality, with the main division being between hard real-time tasks, such as the drive signals sent to the motor pods, and soft real-time tasks, such as servicing messages from the GPS and communication modules. A picture of the drifter as well as a module-level view is shown in Figure 2.7. We have provided a summary of the important hardware capabilities of the drifter in Table 2.2. The central processing unit (CPU) is a Gumstix Overo microprocessor and communicates, via a FT4232H USB-to-serial chip, to the other modules on the board. The CPU also uses some general-purpose input/output ports to control the power to each module as well as various control lines.

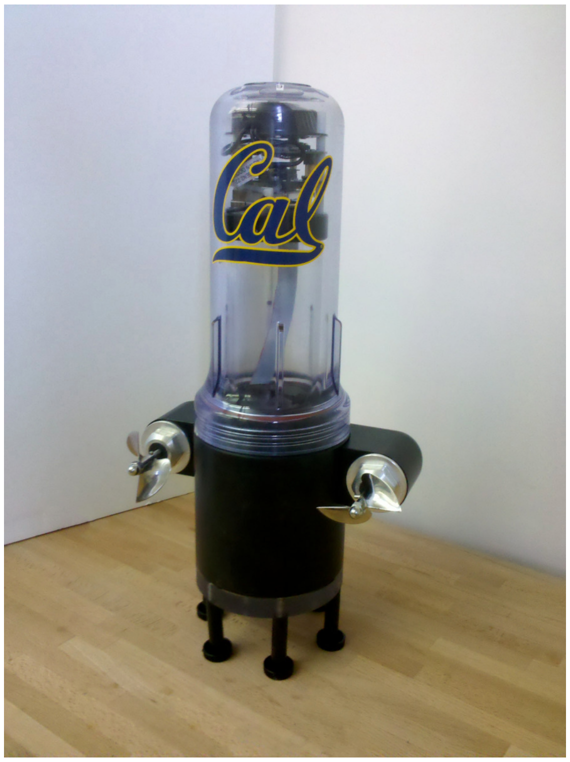
The drifter electronics hardware is comprised of six circular circuit boards separated by function. A major constraint driving the multitude of separate boards was shape of the vessel [21], designed to be cylindrical. Shown in Figure 2.8 is a breakdown of how the circuit boards are arranged in the drifter unit. From top to bottom, we describe the responsibilities and components of each board:

A: The *Top Board* is a 3.1 inch 2-layer circular PCB. It contains the GPS module, compass module, XBee-compatible pin headers, and an ATxmega128A3 microcontroller which reads the compass and can intercept communications to or from the GPS or XBee modules. The microcontroller is given control of the power domains of the drifter and can thus extend battery life by turning off the high-consumption subsystems such as the main CPU.

B: The *Overo Board* is a 3.1 inch 4-layer circular PCB. The board connects to the top board via two 14 pin board-to-board headers and communicates to the bottom board stack through an additional 14 pin ribbon cable. It also receives power by a 4-conductor large-gauge cable. This board houses the GSM cell-phone modem, the Overo CPU, and indicator LEDs. A FT4232H USB-to-serial chip provides the Overo CPU with four serial ports with which to communicate with the drifter's various modules.

Table 2.2: Generation 3 Drifter Platform Capabilities

Master CPU	Gumstix Overo Water single-board computer 720MHz OMAP3530 ARM Cortex-A8 256MB RAM, 256MB Flash OpenEmbedded Linux Distribution
Experimental Data Memory	1GB or 2GB micro-SD card
Real-time processors	2 ATxmega128A3 processors 32MHz 8-bit AVR 8KB RAM, 128KB Flash
Short-Range Communication	XBee-PRO S2 radio module IEEE802.15.4 / ZigBee compliant +17dBm output
Long-Range Communication	Motorola G24 modem GSM compliant using GPRS data service +33dBm output
Position Sensing	MT3329 GPS Module 66 Channels, 10Hz update rate -165dBm sensitivity, <3m accuracy
Orientation Sensing	HMC6352 2-axis digital compass 20Hz update rate Automatic calibration
Propulsion	11.1V Brushed DC Motors in differential drive Hobby-grade electronics speed controllers Plastic Propellers
11.1V Lithium-Ion Batteries	Motor: 10.4Ah(6.5h lifetime) Electronics: 4.2Ah(48h lifetime)



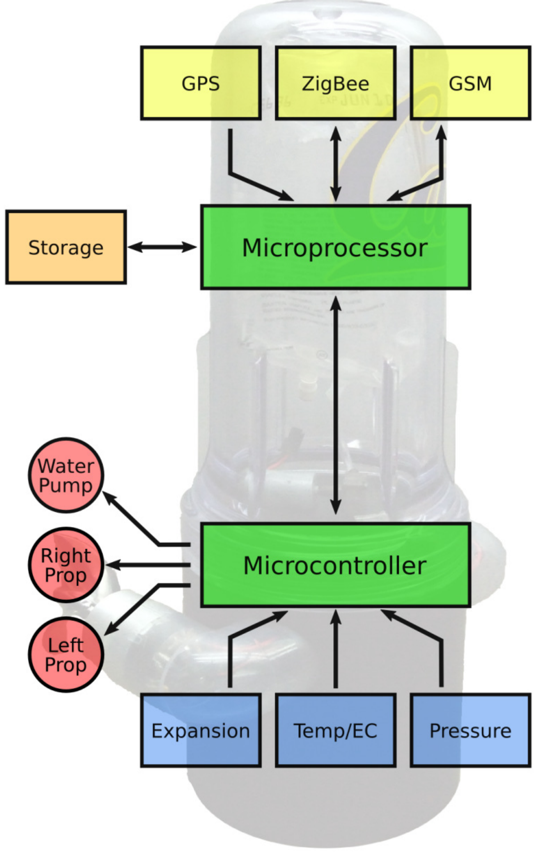


Figure 2.7: Left: motorized sensor unit. Right: modular breakdown of drifter.

C: The *ESC Board* is a rectangular PCB placed vertically in the tube supporting the upper electronics stack. It contains two commercial electronic speed controllers (ESCs) and the level-shifting circuitry to operate them. A 14 pin ribbon cable connects this to the power board (which also provides a connection to the control board) and a two pin battery connector provides power.

D: The *Power Board* is a triangular PCB responsible for converting the 11.1V battery voltage to 3.7V to run the electronics and measuring the power consumption via a MAX4173F chip. The voltage conversion is facilitated by a buck converter circuit controlled by a TPS5430 chip.

E: The *Control Board* board contains another ATxmega128A3 microcontroller responsible for controlling the direction of the drifter and reading measurements from the water quality sensors below. Originally the compass was to be placed on this board, but we found that between the battery and motors there was too much magnetic distortion for the digital compass to function.

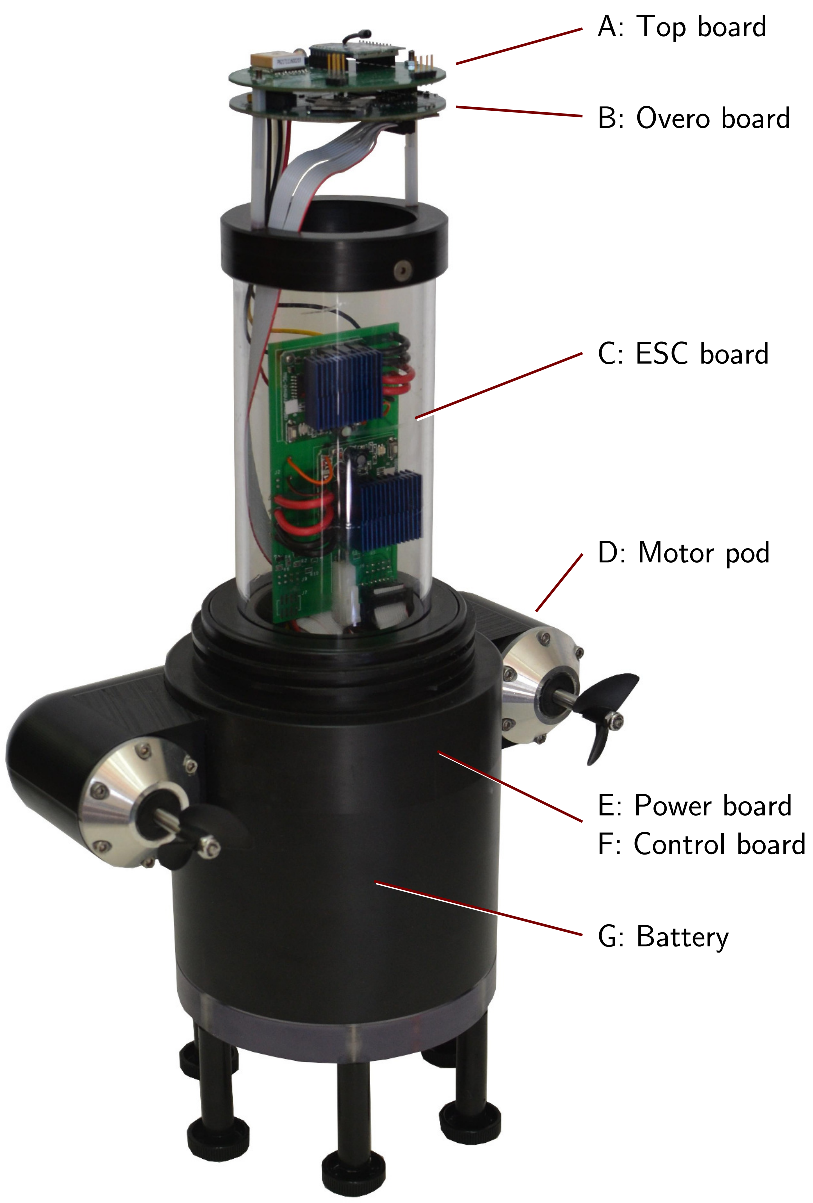


Figure 2.8: Generation 3 Drifter Hardware Components.

Software

With the power and flexibility of the Gumstix Overo running Linux, we designed our software to be primarily written in the Python language. This choice of language offered some important advantages over previous generations which were written in C. Most notably, since Python is a scripting language, it does not require cross-compilation targeting the Overo. Thus, we were able to have rapid design cycles, as each code change required only uploading the new code to the drifter. Additionally, the scripts could be edited on the drifters themselves.

An additional decision to aid in development and robustness was to separate the code into modules which would run in independent processes. The most immediate advantage of this strategy is that, if one module experiences errors and crashes, it can be restarted without affecting the rest of the system. We chose to use UNIX IPC sockets for inter-process communications (IPC) which enabled a straightforward implementation of a SIL simulator. To serialize and decode messages, we use Google Protocol Buffers[22].

The modules are separated according to function, as shown in Figure 2.9 which lists the set of modules used in a typical experiment. Each module hosts a UNIX IPC server and a corresponding library of functions for other processes to call, making the appropriate UNIX IPC client connections and messages.

A feature of this modular design choice is that we can write *virtual* modules which emulate the functionality of physical systems. For example, when testing the control algorithm presented in this article, we wrote a virtual GPS module providing coordinates from the dynamics simulator described in Section 3.3. The dynamics simulator includes a simulated heading-hold controller which exports a virtual motor interface and takes motor commands via this service.

The intent is that the same production code which runs on the actual drifter hardware can be connected to a simulated drifter. This enables running certain experiments without the overhead of a field operation. Part of the motivation for future field tests is to tune simulator parameters to produce the same trajectories as what we observe in the physical world.

An Internet server hosted at UC Berkeley runs a set of modules based on the same codebase as the embedded Python modules running on the drifters. The Internet server runs Ubuntu Linux, providing a similar environment for the code to be developed and run in. In the remainder of this section, we describe the various Python modules (processes) running on the drifter and Internet server.

Device Drivers

- **G24** : The G24 device is configured and operated with a set of AT serial commands. The corresponding device driver is a state machine responsible for creating a TCP/IP connection to our remote Internet server and forwarding messages from other processes.

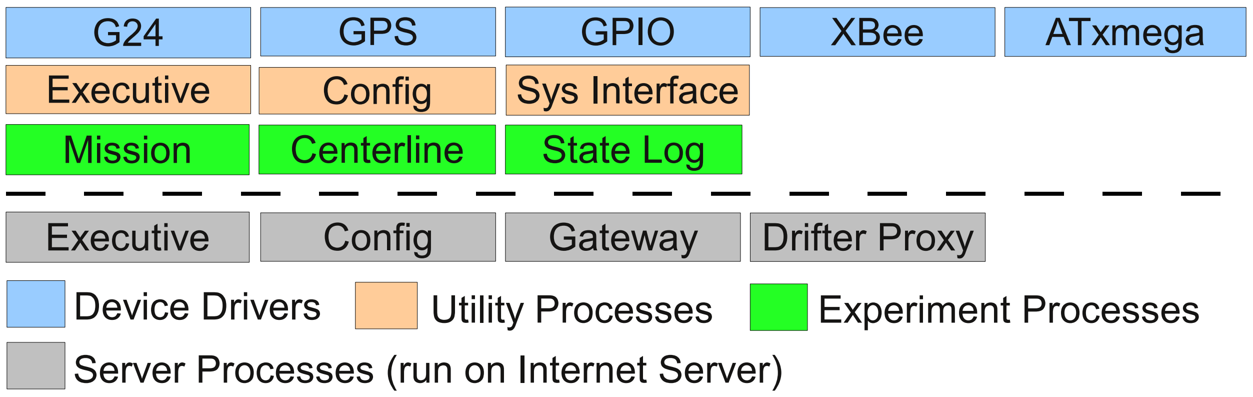


Figure 2.9: Set of Python modules (processes) running during a typical experiment.

- **GPS** : This device driver is capable of reading position readings from the MT3329 module in both NMEA and binary formats. The driver then sends the GPS updates to one or more client processes at a configurable interval.
- **GPIO** : This driver services the 27 GPIO control lines, through the sysfs interface in linux, or via the ATxmega driver, depending on which chip the GPIO line is connected to. It is responsible for translating logical pin names to physical port and pin indexes and initializing all pins to the correct direction and values on startup.
- **XBee** : This driver provides functionality similar to the G24, but for operating the XBee short-range radio modem instead, using the "API mode" of the device. Other responsibilities are fragmenting and reassembling messages to/from 84 byte packets, retrying transmissions, and translating the logical IDs of devices to and from their 64-bit network address.
- **ATXmega** : This driver communicates to the firmware on the two ATXmegas using a Command-Length-Value protocol which reads and writes to virtual registers (e.g. *motor_mode* or *heading_reading*). The driver services requests from other processes to read/write registers and sends the request to the top or bottom microcontroller as required.

Utility Processes

- **Executive** : This process is the entry point for the rest of the scripts as it is responsible for starting all of the other processes and re-starting a process if it crashes. The module also captures the output of each module, time-stamping it, and writing it to a unified log file. There is also an IPC interface and command-line tool provided to start, stop, and restart any of the gen3 processes.

- **Config** : This process parses key-value pairs from a master configuration file which, in-turn, imports other subfiles. Key-value pairs are categorized by the modules they belong to, although any module can access any other module's configuration. An IPC server allows other modules to retrieve their configuration values.

- **Sys Interface** : This process uses the XBee and G24 modules to implement a remote control for field debugging. Commands can be sent over the Internet or over the ZigBee network to the drifter to start and stop processes, read status, or drive the drifter's motor.

Experiment Processes

- **Mission** : This process is responsible for the main experimental purpose of the drifters– to measure and report their position. The mission module supports reading the GPS position and attempting to communicate over both G24 and XBee modules, depending on the communications infrastructure in-place. It also supports configurable report intervals.

- **Centerline** : This process implements the obstacle avoidance and path selection algorithm detailed in this article. Configuration options include choosing the thresholds for the target and unsafe regions, choosing which policy files are used as well as automate switching of policy files over the course of a 24-hour day to account for tidal cycles.

- **State Log** : This process periodically logs debugging and post-analysis data such as position, number of satellites used, motor speed and desired bearing. The collected values are stored in a sqlite3 database to be downloaded and analyzed at the end of a mission.

Server Processes

- **Executive and Config** : These are the same modules as described above.

- **Gateway** : During normal operation, all drifters and other devices such as field laptops connect to the remote server via a TCP/IP server socket which this process opens. Additionally, other server processes such as the drifter proxy can connect. Each device or processes which connects to the gateway process provides an ID and a list of message types to subscribe to. A device or process can then *send* (to one destination), *publish* (to all subscribers), or *broadcast* (to all clients) a message. All destination devices or processes are addressed by the IDs they provided.

- **Drifter Proxy** : This process is responsible for communicating drifter locations to a data assimilation server residing on the National Energy Research Scientific Computing Center (NERSC), which in turn integrates this into a large-scale model of the river system. Here, the line protocol is a stream of measurements separated by carriage returns and each measurement is a list of key-value pairs, with fields such as *id, timestamp, utm_x, utm_y.*

Heading-hold Control

Control of the drifter's propulsion facilities is divided into the high-level centerline module implementing the algorithm described in Chapter 3, and low-level heading-hold control implemented on the lower ATXmega micro-controller. The goal of the heading-hold controller is to drive the drifter forward along a bearing (angle relative to magnetic north). The controller generates pulse width modulated (PWM) signals for the ESCs which is ratiometric to the power delivered to the motors. The controller's feedback comes from the HMC6352 compass as tenths of degrees from magnetic north.

We chose to use a proportional-integral-derivative (PID) controller[23] to accomplish the control task. Given a desired bearing, θ_{desired}, from the centerline module, and θ_{actual} from the compass, the control law of the heading-hold controller can be expressed as:

$$\begin{aligned}
\theta_{\text{error}}(t) &= \theta_{\text{actual}}(t) - \theta_{\text{desired}}(t) \\
u_{\text{diff}}(t) &= k_d\dot{\theta}_{\text{error}}(t) + k_p\theta_{\text{error}}(t) + k_i\int_0^t \theta_{\text{error}}(\tau)d\tau \\
u_{\text{left}}(t) &= u_{\text{mid}} + u_{\text{diff}}(t) \\
u_{\text{right}}(t) &= u_{\text{mid}} - u_{\text{diff}}(t)
\end{aligned}$$

where u_{left} and u_{right} are the inputs to the PWM generator for the left and right motors and k_d, k_p, k_i and u_{mid} are tuneable constants. The values for the PID constants were determined experimentally by setting a constant desired bearing and using classic PID tuning techniques so that the drifter would travel in a straight line with minimal oscillations. We also ensured that, when giving the drifter a new bearing command, that the step response of the system was stable, a challenge particularly because of the nonlinearity at $0° = 360°$. In practice, we found that using the integral error term was only necessary and useful when θ_{error} was small, so we set a configurable threshold on the error, outside of which, the integrator's state is set to zero. Without this modification, the step response of the system was usually unstable.

Simulator

A highly-parameterized and extensible SIL simulator was developed to assist in debugging code before it was deployed onto the drifters. SIL means that unmodified drifter code can be run in a simulation environment, versus a separate behavioral model being constructed. The primary advantages are reducing code duplication and that the simulator can be used to check for bugs in production code.

A simulation definition file is given to the simulator, specifying one or more *simulator modules* to be loaded and how to connect their interfaces. The difference between the simulator modules and the drifters' modules is that a single simulator module process can be responsible for each unit in the entire fleet versus a drifter module process must be started for each unit in the fleet. The simulator acts as a sort of switchboard by connecting the interfaces of both the simulator and drifter modules, according to the simulation file.

The simulator modules are Python objects having a number of *exported* and *imported* interfaces, where exported interfaces are methods which can be called by other simulator modules, and imported interfaces are those which must be set to another simulator module's exported method. For example, the *drifter_heading_hold_PID* exports a *motor_control* method returning the left and right motor inputs. This is connected to the *drifter_dynamics* module's imported interface also called *motor_control.* In addition to these code interfaces, the simulator modules additionally open UNIX sockets to accept connections from drifter modules being tested.

To verify an entire fleet, the simulator can simulate multiple units by dynamically generating configuration files and running multiple copies of the modules under test. It ensures that there are no conflicts in UNIX socket names. We can configure groups of drifters and set specific parameters for those groups, for example, to configure a part of the fleet to go down a different fork of river.

This allows us to run a wide variety of studies including

- Obstacle avoidance and path selection studies with a fleet of motorized drifters, verifying they stay in the river and go down the correct path, respectively.
- Data delivery studies, exercising the mission code for errors and ensuring data is delivered to the Internet server.
- Wireless networking experiments where we can simulation communication between the short-range radios of the devices.

For example, figure 3.8 shows an example of a simulated fleet of drifters performing path selection.

2.4 Architecture for mobile floating sensor data collection and assimilation

Communications architecture

Figure 2.10 shows the communication links between various elements of the system. Data collected by the active and passive drifters is communicated back to the database server using the General Packet Radio Service (GPRS) of GSM. The Android smartphone on board each passive sensor provides the necessary GPRS functionality. Our original design for the

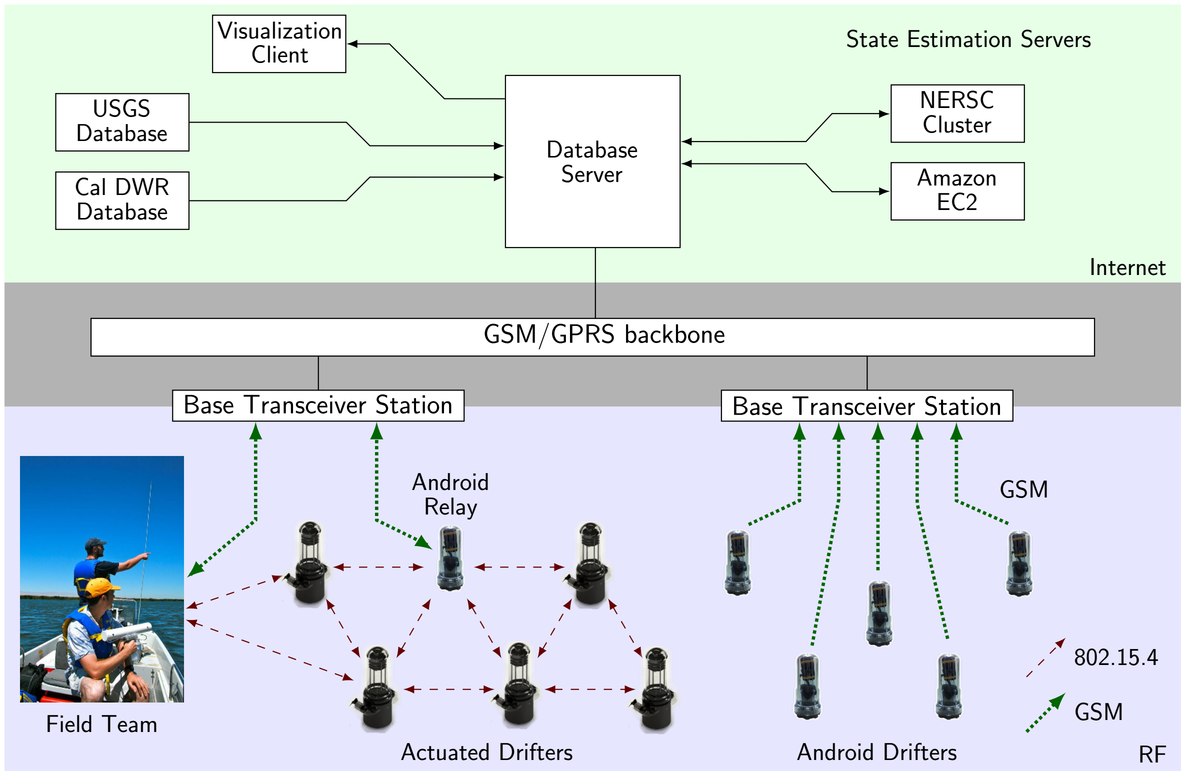


Figure 2.10: Communication architecture, showing the flow of data from drifters in the field to the database server and computation servers via the GSM service.

active drifters included two communication modules: a Motorola G24 OEM GSM module for direct communication with the server, and a Digi XBee-PRO 802.15.4 module for short-range communication with other drifters and the field team. Reliability issues prevented us from using the G24 GSM module, however, and so the active drifters communicate solely through the XBee module.

The XBee-PRO module conforms to the IEEE 802.15.4–2006 draft standard for low-power mesh networking. Our experience in outdoor environments shows that point-to-point links of 100 m are reliable, and we have seen connectivity at distances of 1 km. In order to bridge between the 802.15.4 short-range networking and the database servers, we built 10 specialized Android drifters carrying a XBee-PRO module as well as an Android smartphone. These devices, called “Relays”, were put in static locations around the experimental environment. They did not gather data themselves, but simply collected the data from the active drifters and transmitted it to the database server via GSM.

Field teams carried laptop computers with GSM modules and XBee-PRO modules. The active drifters can be sent commands via their XBee modules for diagnosis and troubleshooting. Capabilities include enabling and disabling the motors, running or terminating processes on the Gumstix, and querying various values like mission state or sensor readings. These commands can be sent directly over the XBee link, or can be sent indirectly over GSM through the database server and the Android relays. This command capability is for development and debugging purposes. During the April 12 and May 9 experiments, no commands were sent to the active drifters; they operated autonomously.

The database server acts as a central repository for gathered data and assimilation results. In addition to the Lagrangian data collected by the drifters, relevant data from USGS and California Department of Water Resources (DWR) sensor stations is collected and stored. The sensor data is sent to the computational cluster, which can be either a collection of Amazon Elastic Computing Cloud (EC2) processors or the NSERC computational cluster. Results from the assimilation process are stored on the same database server, and queried by the visualization application, which can be accessed on the Web by any browser.

Assembly and testing

The construction and debugging of this kind of autonomous system is made especially challenging due to the quantity of units we built. In this section we describe our experience in building the fleet, possibly offering ideas to others who wish to undertake a similar effort. We prototyped the drifters in several stages. The first drifter was built by hand assembling the electronics boards and drifter body. Subsequent drifters had the electronics and hull produced by outside companies, although all of the pieces still needed to be attached together. We created detailed documentation with pictures so that we could employ outside help such as interns to assemble them. Our first batch of drifters was a quantity of 10, which we dubbed *Generation 3.0*. These electronics of the units had several problems, in particular, they were expensive, used unreliable connectors, and were difficult to service. The second batch of 10 was dubbed *Generation 3.1* and was redesigned to address these problems. Finally, we produced 20 more Generation 3.1 drifters, bring our total fleet size to 40.

From our experience with the Generation 2 drifters, we made it a specific goal in designing Generation 3 for them to be quick to set-up for experiments as well as update their software. Features of the drifters include a screw-top lid and battery connections that can be pulled out to be charged. A serial port is easily accessible which provides console access to the Linux computer. Since it proved time-consuming to update software or download logs from each drifter individually, a custom piece of software was written to automate the process by autonomously entering in console commands and interpreting the results. Combined with a 20-port serial cable and a bootloader program residing on the drifter, we can update the software on all of the devices in less than an hour.

For keeping track of the fleet status, we found it invaluable to maintain a spreadsheet, shared amongst the group, listing drifter IDs and the status of their systems. Drifter IDs are assigned chronologically based on when they were built, from 0 to 39. We divide the fleet into four groups: Drifters which are beyond repair, drifters which are not field ready but could be repaired, drifters which perform sub-optimally, but can be deployed in the field, and drifters which perform well. Tests to classify the drifters were carried out at a neighborhood swimming pool to see if the drifters could drive in a straight line. By far, the most common failure was the compass module not being properly calibrated.

Several days prior to an experiment, we charge the batteries of the drifters, in batches of 10 drifters, two batteries each. If a software update is needed, we also use the 20-port cable to do this as well. The day before the experiment, drifters and equipment are packed into plastic tubs.

On the day of the experiment, we take the tubs to the experiment site, unpack the drifters, turn them on and verify communication to the server. Over most of the day, we perform several cycles of dropping off the drifters upstream and picking them up downstream. At the end of the day, drifters are retrieved, turned off, and repacked.

After the experiment, we connect the 20-port cable to download the drifter logs which can be analyzed later if a failure has occurred. The fleet status spreadsheet is also updated to reflect if any failures were encountered in the field.

Data assimilation and the Ensemble Kalman Filter

The database server performs the data assimilation algorithms to obtain flow field estimates from the Lagrangian measurements. We use an adapted data assimilation method based on the Ensemble Kalman Filter (EnKF) which is computationally efficient enough to be run in real-time [3]. At a high-level, like other Kalman Filter derived techniques, this technique relies on a model of the state of the system (such as water velocity and stage), and data with which to check the model output (Lagrangian measurements). A Kalman Filter essentially alternates between a Prediction step and Update step, where the Prediction step refers to propagating the model state forward in time, and the Update step refers to evaluating the state estimate using the physical measurements. Since our system physically has an infinite number of states (velocity and stage at every point), the EnKF technique does not track the entire state of the system, but evaluates a representative set of states, called the ensemble.

There are two main difficulties in using Lagrangian data that we have experienced: Firstly, Lagrangian motion is heavily affected by local flow perturbations and thus may not reflect the larger picture of water velocity. Secondly, Eulerian and Lagrangian behaviors are not simply related to each other. Thus, as long-term collection of continuous Lagrangian measurements is not yet developed, this motivates designing the ability to rapidly deploy a dense fleet of Lagrangian drifters. Furthermore, we design the sensing-modelling system to predict regional flows and transport, in real-time, without dependence on historical data.

The EnKF [3] and variants are the main data assimilation technique in atmospheric and oceanic sciences. The main differences between variants of the technique is how the analysis ensemble is generated and also how model/measurement noise is handled. One family of schemes relies on perturbed observations [24, 25, 26, 27, 28], whereas the other schemes, the Kalman square-root filters, attempt to reduce the size of the ensemble in a different way. The Kalman square-root techniques only analyse the ensemble once, obtaining both the mean analysis and the analysis error covariance matrix. Our application relies on a system Bayesian approach [29, 30, 31] to recover from modelling errors. This approximating approach [32] approximates the modeling error with additive Gaussian noise processes.

We chose to use the REALM flow model developed by Lawrence Berkeley National Lab (LBNL). Every time step, new data is received from the drifters, and a shallow water model generates a representative set of states by evolving the processed inputs. The EnKF routine then compares the real data to gathered output estimates in order to give the best estimation of the flow.

For the May 9, 2012 experiment, 2 hours of field data was collected and run in the assimilation experiment (shown in Figure 2.11). A variety of ensemble members and inflation factors were tested, as well as different data configurations and model setups. Since there were no USGS Eulerian measurements available in the experimental domain, the data reconciliation method developed by [33] was used to get estimates of the discharge, and pose it to be the first guess of the data assimilation process. This data has also been used to verify an experimental model and estimation technique using hyberbolic partial differential equations subject to periodic forcing [34].

2.5 Conclusions

The Floating Sensor Network (FSN) has enabled new types of experimental work in rivers and estuaries via two novel drifter designs: First, the low-cost, manufacturable, and easy-to-use Android drifter enables economical and high-density collection of Lagrangian data in real-time. We have shown that the capabilities provided by Android smartphones satisfy most of the features required by a surface drifter. The Android phone also enables real-time reporting and orientation detection by the GSM communications link and accelerometer sensor, respectively.

The second technological contribution is the active motorized drifter which demonstrated the feasibility of motorized Lagrangian sensors and proven that they can indeed avoid obstacles, thus significantly extending the particle lifetime of a Lagrangian sensor. We have developed a high-level Hamilton-Jacobi Safety controller, described in the next chapter, as well as the low-level heading-hold controller which successfully regulates the direction of the motorized drifter under actuation.

These two developments have implications for future drifter studies where previously impractical areas of water systems can be monitored. Furthermore, the Lagrangian data which is collected by the FSN has unique advantages over the traditional method of collecting fixed Eulerian data, particularly when the mobility of the water is primarily important. In our many experiments carried out with the fleet of drifters, we have observed interesting phenomena such as jets (when fast moving water such as from a river, enters a stationary body, such as a lake) and tidal inversions (when water enters a river system from the ocean). In these cases, Lagrangian data is crucial to determining how the water is mixed and distributed, as the drifters themselves will be proxies for the water mass.

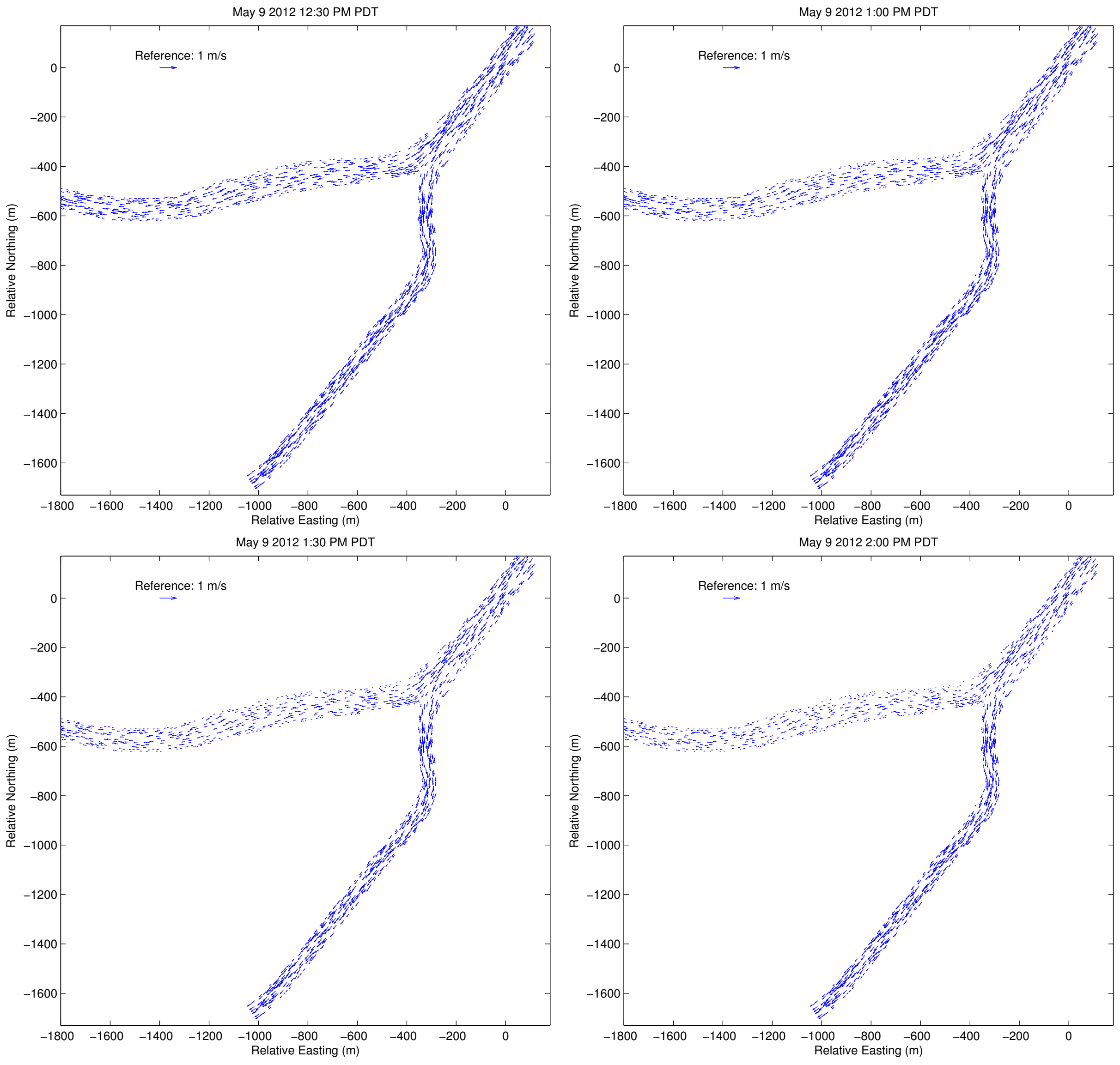


Figure 2.11: Quiver plots of assimilation results for flow fields during May 9 experiment.

As well, we hope that our successes within the FSN are not constrained to drifter studies, but find applications in mobile environments, such as monitoring airborne pollutants, or fluid flow in internal medicine. For instance, in the next chapter, we elaborate on the Hamilton-Jacobi Safety control method developed for the drifters which can be easily adapted to the aforementioned applications by changing the model.

Chapter 3

Hamilton-Jacobi safety control for underactuated sensors

3.1 Introduction

As we have demonstrated in the previous chapter, the estimation of water mobility within rivers and estuaries requires Lagrangian sensing. In this scenario, the drifter's own mobility is essentially the sensing technique. Therefore, when this mobility is impeded, such as by obstacles, it *directly* precludes the drifter from accomplishing its sensing task.

In Section 2.3, we described the *Generation 3* drifter platform of the Floating Sensor Network, which is designed to expand the operational capability of our river studies by venturing into obstacle-laden environments. We find that, in many of the river environments in the Sacramento–San Joaquin delta, drifter studies using non-motorized sensors are infeasible. Even when continuously monitored using boat teams, the drifters become stuck on obstacles such as the river banks or vegetation. It is simply impossible to retrieve and re-deploy the drifters fast enough to keep them simultaneously unimpeded. Thus, we developed the Generation 3 motorized drifters and a navigation technique to allow unattended studies in these challenging environments.

In this chapter, we address the problem of obstacle avoidance and path selection in a mobile environment by applying the solutions of Hamilton-Jacobi-Bellman-Isaacs (HJBI) equations. The *obstacle avoidance* task is the ability of the drifters to avoid becoming stuck on hazards. To comprehensively understand water conditions, it is also necessary for a drifter fleet to distribute itself among multiple paths throughout the water system, which we refer to as the *path selection* task. Although we have developed this technique specifically for the drifter application, these techniques could be applied to any other mobile environment where we have some control over the mobile elements' states, and these states must be held within some constraints. Examples include Unmanned Aerial Vehicles (UAV), Autonomous Underwater Vehicles (AUV), and nano-surgical implanted robots.

Controlling in the presence of obstacles is linked to *path planning* problems [35, 36], which sometimes rely on the same theory as this article [37]. Two features of our problem distinguish it from the traditional path-planning problem:

- The drifter is an underactuated system. That is, the unit is in the presence of a river current which is more powerful than the propulsion of the unit. Thus, a successful algorithm must account for the river current and act preemptively to avoid being pushed into an obstacle.
- Our goal is not to reach a single target way-point, rather, the goal of the drifter is to not run into obstacles.

Several approaches can be used to solve these types of problems. Viability-based approaches compute regions of the state-space such as "all points guaranteed to be safe" [38, 39]. Another approach is to use the level and sublevel sets of solutions to HJBI equations [40, 41, 42, 43, 44]. We chose to use the HJBI framework, which can be used to compute the same sets, in order to use an existing mathematical toolbox [45] to solve these equations numerically.

Generally, the HJBI method provides a way to evaluate the danger of areas in the river by the time to reach obstacles, as opposed to the distance to reach obstacles. It incorporates the dynamics and movement of the drifter due to its own actuation, in addition to the movement that the surrounding water current imparts on it. The HJBI method is a formulaic method of calculating these areas from intuitive inputs. It applies to many parts of a river system: either large bodies of water, or small tributaries.

We show that the solution to a HJBI equation can be used to construct a minimum-time-to-reach (MTTR) function to a given target region. Two such MTTR functions, V_{center} and V_{shore}, are used to determine the transitions of the on-off controller. With the proper MTTR function, it is also possible to find the optimal control policy, i.e. the direction in which to travel in order to reach a target the quickest. It is well known that because the HJBI equation is derived by application of Dynamic Programming (DP) techniques [42], synthesizing these MTTR functions suffers the same curse of dimensionality as other DP methods [46]. Fortunately, for low-dimensional systems, such as described in this article, the problem is tractable.

The drifter unit's measured velocity matches the local water velocity only while the drifter is *not* under actuation. Thus, we seek to maximize the amount of time the motors are turned completely off. On-off control is therefore a natural choice, since it specifies using maximum actuation or none at all. It then remains to determine the on-state policy and thresholds of the on-off controller.

In this chapter, we extend upon the results of [20] and go more into depth describing the work, particularly with respect to simulation.

3.2 Hamilton-Jacobi control methodology

Model and problem statement

We model the system dynamics of a single drifter as a 2-dimensional single-integrator:

$$\begin{aligned} \dot{x} = f(x, a, b) &= w(x) + a(t) + b(t), \\ \|a(t)\|_2 &\leq \bar{a}, \\ \|b(t)\|_2 &\leq \bar{b}, \end{aligned} \tag{3.1}$$

where $x \in \mathbb{R}^2$ is a two-dimensional state vector representing the position of the drifter in meters, $a(t)$ is the *control input*, and $b(t)$ is the *disturbance input*. Note the absence of a yaw state variable, which is intentionally omitted to reduce the computational burden. We believe the term to be largely irrelevant for longer time scales as the vehicle is highly maneuverable around its vertical axis, owing to the differential drive configuration.

An estimate of the river current, $w(x)$, is given by a forward simulation of the region *without* integration of drifter-collected data. Note that, although the purpose of the FSN project, as a whole, is to provide more accurate estimates of the river currents using the drifters, nevertheless, a less-accurate simulation can be used for the purposes of control. As the river current estimates improve by integrating the drifter-collected data, then the control will also benefit for future experiments by having a more accurately specified $w(x)$.

We also define the following sets of functions and parameters, $\bar{a}$ and $\bar{b}$:

$$\mathbf{A} \triangleq \{a(\cdot) \colon \|a(t)\|_2 \leq \bar{a} \ \forall t\},$$
$$\mathbf{B} \triangleq \left\{b(\cdot) \colon \|b(t)\|_2 \leq \bar{b} \ \forall t\right\},$$

and parametrize the trajectory of the system in terms of time, initial condition, and the $a(\cdot)$ and $b(\cdot)$ inputs,

$$x = x(t; x_0, a(\cdot), b(\cdot)).$$

With these functions we will set up a *differential game* [47, 41] in which the inputs $a(\cdot)$ and $b(\cdot)$ work against each other to either satisfy or attempt to violate a safety condition, respectively. As we will see later, $b(\cdot)$ will always act in the opposite direction of $a(\cdot)$ with magnitude $\bar{b}$. Therefore, in this case running this differential game with input constraints $(\bar{a}, \bar{b})$ is equivalent to running a single-player game with input constraints $(\bar{a} - \bar{b}, 0)$, although this is not true for general differential games [47].

To set-up this game, we are given a set of undesirable positions, $\mathcal{T}_{\text{shore}} \subset \mathbb{R}^2$ corresponding to collisions with obstacles. Conversely, the complement of this set, $\mathcal{S} \triangleq \mathcal{T}_{\text{shore}}^C$, gives the positions for which the system is *safe*.

Our goal is to find a control input $a(\cdot)$ such that

$$\forall t > 0, \ \forall x_0 \in \mathcal{S}, \ \forall b(\cdot) \in \mathbf{B}, \ x(t; x_0, a(\cdot), b(\cdot)) \in \mathcal{S}. \tag{3.2}$$

Ideally, $a(\cdot)$ also minimizes the time of actuation,

$$t_{\text{act}} = \int_0^\infty a_{\text{on}}(t)dt,$$
$$a_{\text{on}}(t) = \begin{cases} 1 & a(t) \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

however, the proposed algorithm does not necessarily produce an optimal $a(\cdot)$ in this respect.

Mathematical solutions

In this section we describe the meaning of a MTTR in the context of HJBI equations. In the next section, we describe several MTTR functions used in an algorithm to satisfy (3.2). We begin with a *target set*, $\mathcal{T} \subset \mathbb{R}^n$, giving a set of states we are trying to reach. Consider the construction of a *static cost function*, $V(x_0)$:

$$V(x_0) = \inf_{a(\cdot)\in\mathbf{A}} \sup_{b(\cdot)\in\mathbf{B}} \left\{ \int_0^{t^\star} l\left(x(t; x_0, a(\cdot), b(\cdot)), a(\cdot), b(\cdot)\right) dt \right\}, \tag{3.3}$$
$$t^\star(x_0, a(\cdot), b(\cdot)) = \inf\left\{t\colon x(t; x_0, a(\cdot), b(\cdot)) \in \mathcal{T}\right\}, \tag{3.4}$$

where $l(\cdot,\cdot,\cdot)\,(\mathbb{R}^n, \mathbb{R}^{n_i}) \mapsto \mathbb{R}$ is a Lagrangian cost functional associating a cost for the system to be in a certain state and taking a certain action.

Thus, the interpretation of (3.4) is that it designates the first time the trajectory,

$$x(\cdot; a(\cdot), b(\cdot)),$$

enters $\mathcal{T}$.

Suppose we take $l(\cdot,\cdot,\cdot) \equiv 1$, representing a constant accrual of cost until the target set is reached. Substituting into (3.3),

$$\begin{aligned} V(x_0) &= \inf_{a(\cdot)\in\mathbf{A}} \sup_{b(\cdot)\in\mathbf{B}} \left\{ \int_0^{t^\star} 1 \cdot dt \right\} \\ &= \inf_{a(\cdot)\in\mathbf{A}} \sup_{b(\cdot)\in\mathbf{B}} t^\star(x_0, a(\cdot), b(\cdot)), \end{aligned} \tag{3.5}$$

or, more concisely

$$V(x_0) = t^\star\left(x_0, a^\star(\cdot), b^\star(\cdot)\right), \tag{3.6}$$
$$a^\star(\cdot) = \arg \inf_{a(\cdot)\in\mathbf{A}} t^\star, \tag{3.7}$$
$$b^\star(\cdot) = \arg \sup_{b(\cdot)\in\mathbf{B}} t^\star,$$

where $a^\star(\cdot)$ is called the *optimal control* for this particular cost function as it minimizes the accrued cost, and $b^\star(\cdot)$ is the *worst case* disturbance.

In this case, (3.6) simply gives the minimum time to reach the target set $\mathcal{T}$ from x_0, so we call V a MTTR function for this system. Note that $V(x_0) = +\infty$ in the case in which the target set is not reachable from the initial condition x_0.

We are interested in finding the optimal control, or disturbance, which satisfies (3.7) and achieves a minimum-time trajectory to, or from, $\mathcal{T}$. Both can be computed explicitly as a function of the gradient of the MTTR function by the following relations:

$$a^\star(x) = -\bar{a}\frac{\nabla V(x)}{\|\nabla V(x)\|_2}, \qquad b^\star(x) = \bar{b}\frac{\nabla V(x)}{\|\nabla V(x)\|_2}. \tag{3.8}$$

In general, V is difficult to compute especially for systems such as (3.1) which have an arbitrary forcing term, w, and an arbitrary target set, $\mathcal{T}$. We elect to extend the technique found in [45] for finding the MTTR function of a holonomic system: a *time-dependent HJBI* equation, for which there are known methods to solve [48], is constructed as follows:

$$0 = \phi_t + \min\left[0, \bar{G}(x, \nabla\phi)\right], \qquad 0 < t < h, \tag{3.9}$$

$$\bar{G}(x,p) \triangleq \max_{\|a\|_2 \le \bar{a}} \min_{\|b\|_2 \le \bar{b}} \left\{p^T \cdot f(x,a,b)\right\}.$$

$$\phi(x,0) = \begin{cases} -1 & x \in \mathcal{T} \\ 1 & \text{otherwise} \end{cases}, \tag{3.10}$$

noting that, although ϕ is, by definition, discontinuous at $t = 0$, that by using the Lax-Friedrichs numerical method, this discontinuity is be dissipated and the solution is stable [49, 45].

As shown in [41], $\mathcal{T}$ can be reached in h time or less from the set of points

$$\mathcal{G}(h) = \{x \colon \phi(x,h) \le 0\}.$$

The frontier of this set of points as it evolves through time is also the set of points from which $\mathcal{T}$ can be reached in exactly h time. The set is related to V by

$$\partial\mathcal{G}(h) = \{x \colon \phi(x,h) = 0\}.$$

Consider a contour, $\{x \colon V(x) = h\}$ of the MTTR function, describing a set of points reachable in exactly h time units. Comparing this contour with $\partial\mathcal{G}(h)$, we find that, for a given x, $V(x)$ is given by the first temporal zero crossing of $\phi(x,t)$. If such a zero crossing does not exist, this means the system cannot navigate from x to $\mathcal{T}$, therefore, $V(x) = +\infty$. This zero crossing can be calculated numerically as in [45].

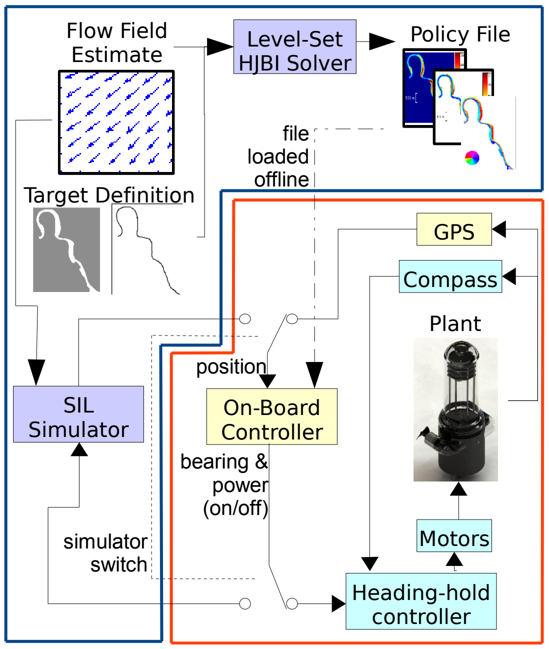


Figure 3.1: Controller Implementation Diagram – Orange box indicates online operations; Blue box indicates offline processes.

3.3 Implementation and simulation

The block-level diagram of the obstacle avoidance and path selection system is shown by Figure 3.1. In this section, we describe the flow field estimate, HJBI solver, and on-board controller components. Additionally, we show results from some of the many SIL simulations we have run to verify the drifter's behavior.

Flow field modeling

We use flow field estimates from REALM, a forward simulation model of the Sacramento–San Joaquin Delta [50], for the values of $w(x)$ required for computation. Due to the tidal nature of the flows in the area, multiple flow field estimates are taken, corresponding to different times of day. The on-board controller will automatically cycle through the policies throughout the course of the day to account for varying water currents.

For simulation purposes, we implement a simple kinematic model of the drifter under the effects of viscous friction and random forces. A REALM flow field is integrated into the environment, and all other physical parameters of the model are designed to most closely resemble the behavior of the physical prototypes we built.

Computation of control feedback

The on-board controller requires three two-dimensional arrays to be computed offline and loaded before the experiment:

1. The MTTR function, shown on the left of Figure 3.3, towards the shore of the river is designated V_{shore}. The V_{shore} MTTR satisfies (3.5), where $\bar{a} > 0$ is the maximum current speed which could push the drifter to the shore, no other force disturbs the drift ($\bar{b} = 0$), and $\mathcal{T} = \mathcal{T}_{\text{shore}}$, the left binary image from Figure 3.2. The function therefore describes how much time the vehicle would move, pushed at speed $\bar{a}$, before crashing on the shore.

2. The MTTR function, shown on right of Figure 3.3, towards the center of the river is designated as V_{center}. This function also satisfies (3.5), with $\bar{a} > 0$ is the maximal propulsion of the drifter, $\bar{b}$ is the maximal disturbance, and $\mathcal{T} = \mathcal{T}_{\text{center}}$, the right binary image from Figure 3.2.

3. The optimal bearing towards the center of the river, denoted by $\angle^{\star}(x)$, is the angle component of the optimal control given (3.8), where V in this relation is V_{center}. This function is shown in Figure 3.4.

These three arrays are combined into a *policy file* which is loaded onto the drifter prior to the experiment. If we repeat this process for different values of w or $\mathcal{T}$, we could generate several such policy files. The drifter is able to select with policy file is used in the on-board controller. For example, the on-board controller could automatically change the policy file over the course of the day to account for periodic tidal flows.

Path selection

To accomplish path selection, we calculated two policy files for the region shown in Figure 3.5, which also illustrates the sets $\{R_1, \ldots, R_6\}$. The first policy file is loaded on drifters which should go down the west path and is generated with the inputs

$$\begin{aligned} \mathcal{T}_{\text{shore}} &\leftarrow R_1 \cup R_4 \cup R_5 \\ \mathcal{T}_{\text{center}} &\leftarrow R_6 \cup R_3 \end{aligned}$$

and the second policy file is loaded on drifters which should go down the east path and generated with the inputs

$$\begin{aligned} \mathcal{T}_{\text{shore}} &\leftarrow R_1 \cup R_2 \cup R_3 \\ \mathcal{T}_{\text{center}} &\leftarrow R_6 \cup R_5 \end{aligned}$$

For each case, we are treating the unselected path as an obstacle and the selected path as the only viable option for the drifter to be directed.

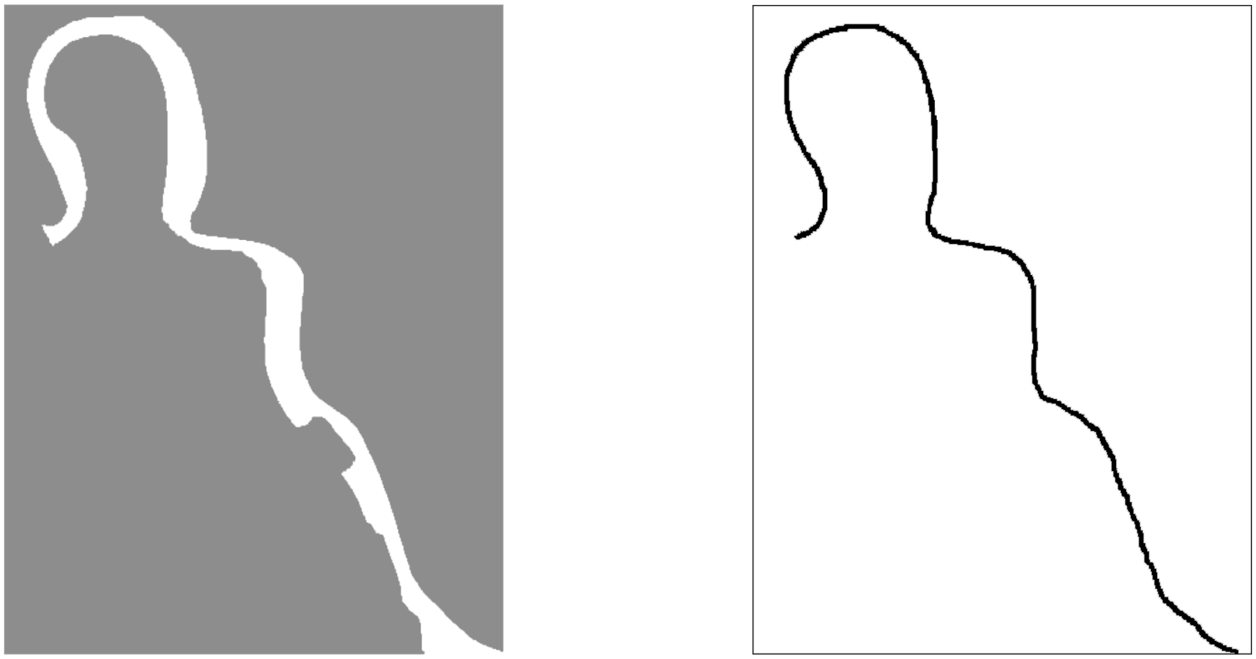


Figure 3.2: Left: $\mathcal{T}_{\text{shore}}$, the constraint set which we want the drifter to avoid. Right: $\mathcal{T}_{\text{center}}$, target set which the drifter needs to reach after touching the unsafe set.

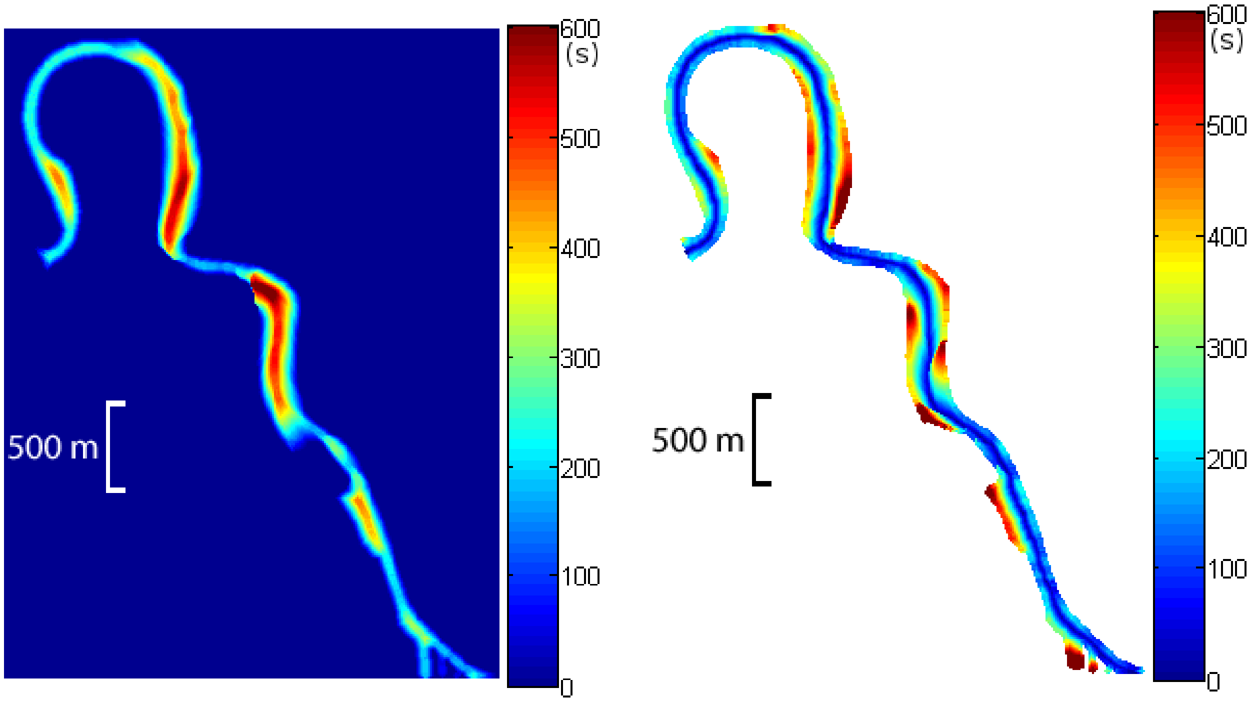


Figure 3.3: Time-to-reach the targets defined by Figure 3.2. Left: V_{shore} function. Right: V_{center} function.

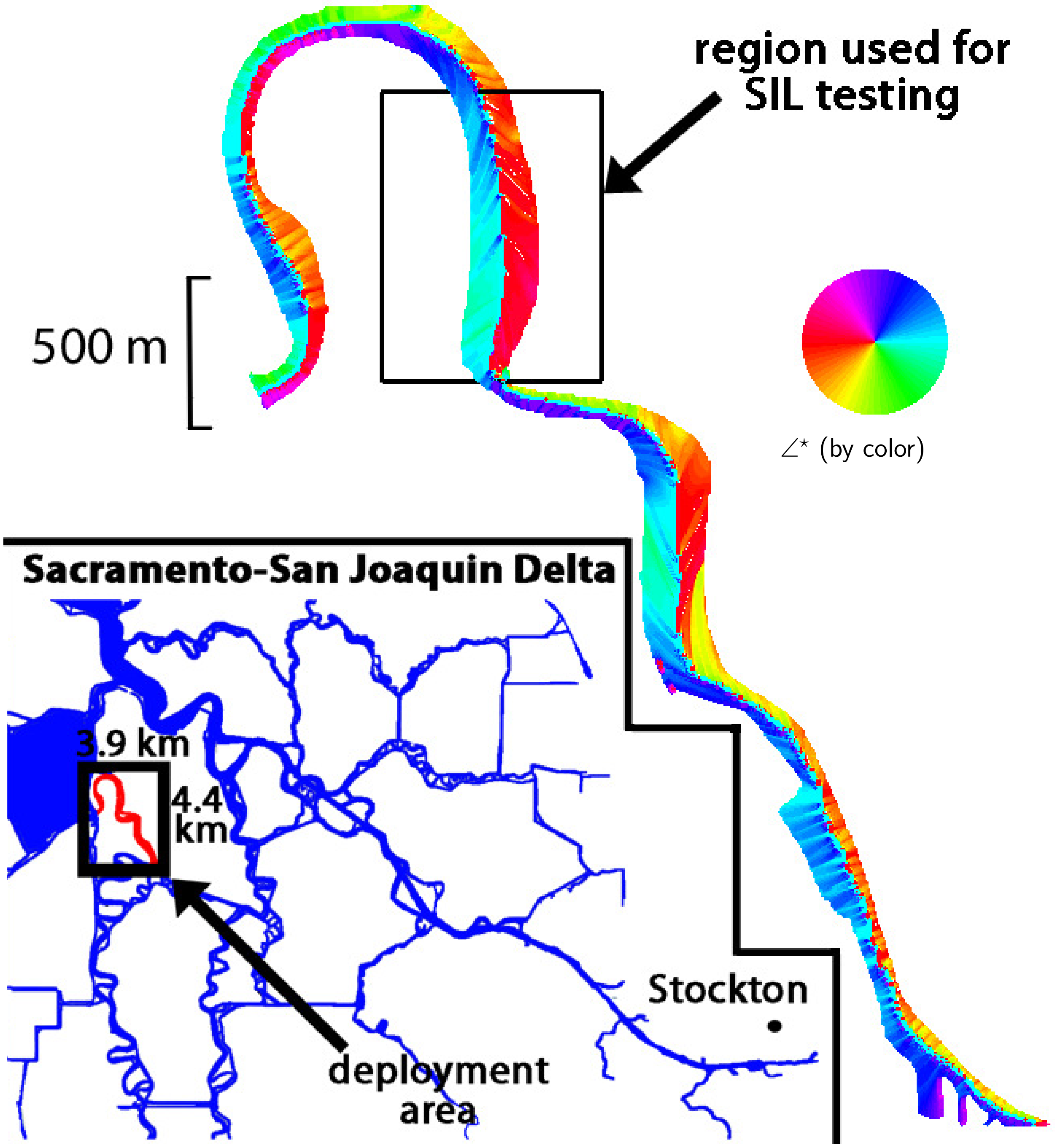


Figure 3.4: The optimal bearing to center of the river, $\angle^\star(x)$.

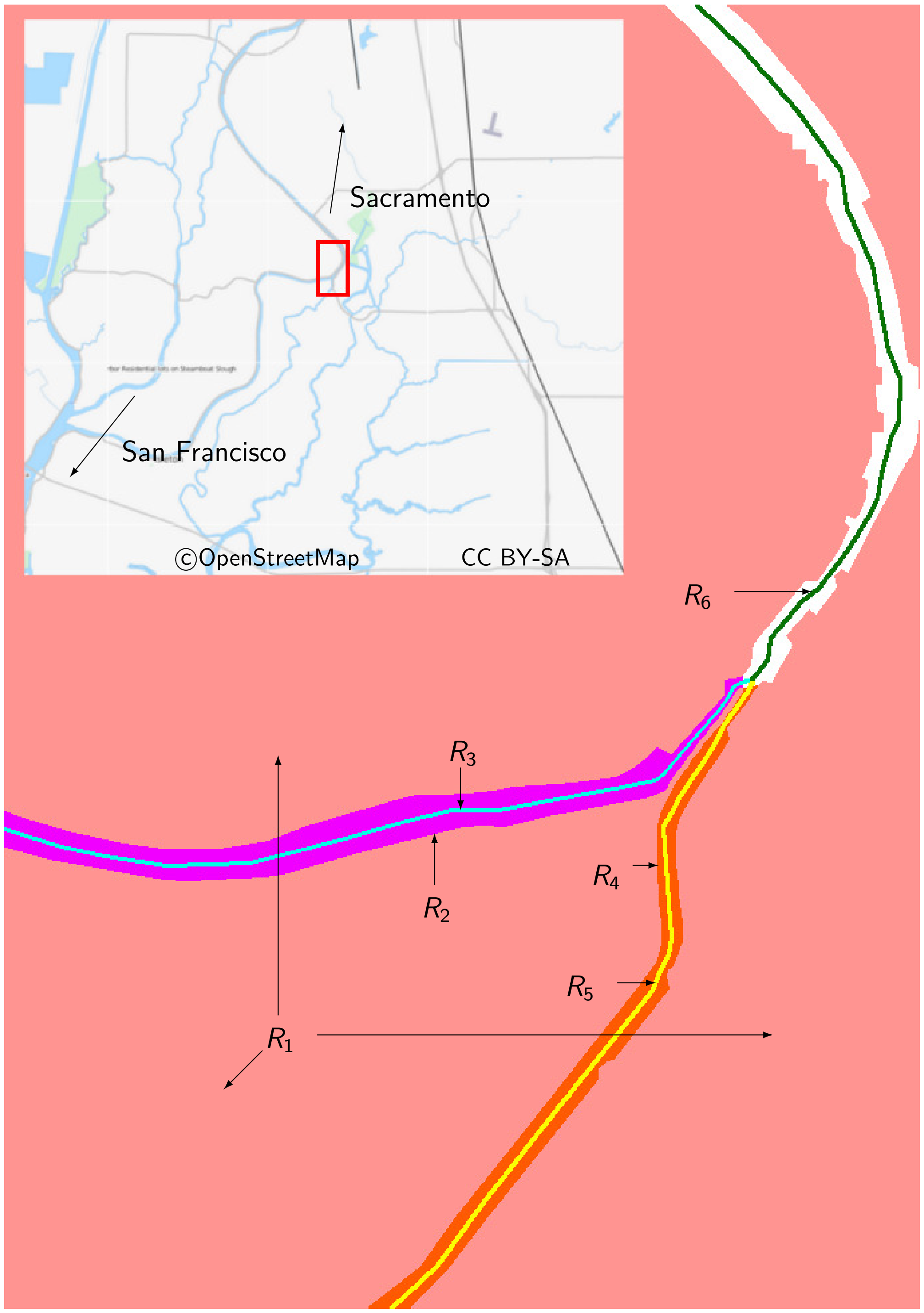


Figure 3.5: Source files for lane-splitting algorithm superimposed and color-coded and map showing geographic location. Each color represents a set of points and is labeled for reference.

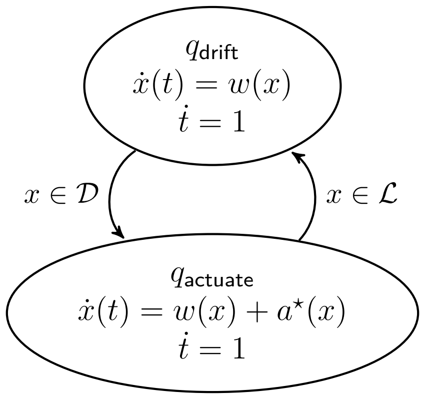

$$\begin{tikzcd} q_{\mathsf{drift}} \\ \dot{x}(t) = w(x) \\ \dot{t} = 1 \arrow[d, bend right, "x \in \mathcal{D}"'] \\ q_{\mathsf{actuate}} \\ \dot{x}(t) = w(x) + a^{\star}(x) \\ \dot{t} = 1 \arrow[u, bend right, "x \in \mathcal{L}"'] \end{tikzcd}$$

Figure 3.6: On-board controller hybrid automaton diagram.

On-board controller

Within this framework, our on-board on-off control system can be encoded by a hybrid automaton $H = (Q, X, R, f, \Sigma, \mathcal{U})$ [51, 52]. Q is the set of the two modes $\{q_{\text{drift}}, q_{\text{actuate}}\}$, which the drifter alternates between during its mission. X is the domain of the drifter's location, $\mathbb{R}^2$. Σ is the set of discrete events which can trigger a transition between modes, and in our case $\Sigma = \{x \in \mathcal{L}, x \in \mathcal{D}\}$. $R : (Q, \Sigma, X) \mapsto (Q, X)$ is the transition function, encoded by Figure 3.6. $f_q : X \mapsto X$ are the dynamics experienced while in each mode, also shown in Figure 3.6. Finally, $\mathcal{U}$ is the set of inputs afforded to the drifter $\mathcal{U} = \mathbf{A} = \{a(\cdot)\colon \forall t \; \|a(t)\|_2 \leq \bar{a}\}$.

The *target region* $\mathcal{L}$ and *unsafe region* $\mathcal{D}$ are defined as the following:

$$\mathcal{L} \triangleq \{x\colon V_{\text{center}}(x) \leq 8 \text{ seconds}\} \cap \mathcal{C}$$
$$\mathcal{D} \triangleq \left\{x\colon V_{\text{shore}}(x) \leq 300 \text{ seconds}\right\} \setminus \mathcal{L}$$

Thus, when the position of the drifter is in danger, i.e. $x \in \mathcal{D}$, the controller turns on the motors and seeks the optimal trajectory back to safety. When the drifter reaches safety, i.e. $x \in \mathcal{L}$, the controller turns off the motors and resumes passive drifting. Throughout the experiment, we record the state of the controller alongside the GPS measurements in order to later discard measurements taken while the drifter was actuating. This ensures that only data corresponding to passive drifting is used for later estimation (assuming, as in our case, that the estimator does not need a continuous trajectory, only point velocity measurements).

Simulated results

Obstacle avoidance

For the following two simulations, we introduced a viscous force on the drifter towards the east. This could, for example, be due to wind. This was necessary to force the drifter into dangerous regions and cause our control to be activated.

Figure 3.7 illustrates two trajectories of the simulated drifter. The left is the ideal situation in which REALM provides a reasonably accurate flow field estimate. In the event that we know REALM will be inaccurate for a region, a compromise is to not use any flow field. This case is shown on the right, where the river flow is set to 0 at all points. As apparent from the results shown here, our algorithm was found to be robust against potential inaccuracies in MTTR calculations when desired actuation is nearly orthogonal to external input.

Path selection

Figure 3.8 shows the result of a simulation experiment to test the functionality of the path selection algorithm. In this simulation, 20 drifters were deployed, with 10 of them configured to proceed down the eastern path of the river and the other 10 configured to proceed down the south path. A previously generated flow estimate was used along with a some random Gaussian uncertainty to provide the simulated flows. The uncertainty was generated by generating a $50m$ grid of random vectors whose components are sampled from $\mathcal{N}(0, 0.02^2)$. The standard deviation was chosen to be one-tenth of the maximum disturbance we expect to see in the field. The uncertainty at any arbitrary location is then a bilinear interpolation from this grid.

The results show that the fleet of drifters correctly separate into two groups and proceed down the two branches of the river. Since the two groups are using a particular adaptation of the obstacle avoidance technique, they also act to avoid crashing into the shoreline. This activity explains the abrupt changes in direction as the drifters drive back to the center of their assigned path when they approach the shore too closely.

3.4 Field operational tests

Ultimately, the Generation 3 drifter design, as well as the proposed navigation methods, were proved through real data-collecting deployments in the Sacramento - San Joaquin River Delta. Figure 3.9 shows pictures of the drifter units undergoing a field operation. In this section, we present the results of two such experiments demonstrating the obstacle avoidance and path selection tasks.

Obstacle avoidance

A field operational test was carried out targeting the Sacramento - San Joaquin River Delta in California (approximately Latitude 38.03 N, Longitude 121.58 W. The controller described by Section 3.3 was tested for approximately five hours in the river. Two boat teams were responsible for monitoring the drifters and retrieving trapped units if necessary. Retrieved drifters were placed back in the river at safe locations to continue their mission. One goal of the experiment was to determine if the controller presented in this article effectively prevented the drifters from heading into dangerous areas, therefore reducing the number of necessary retrievals.

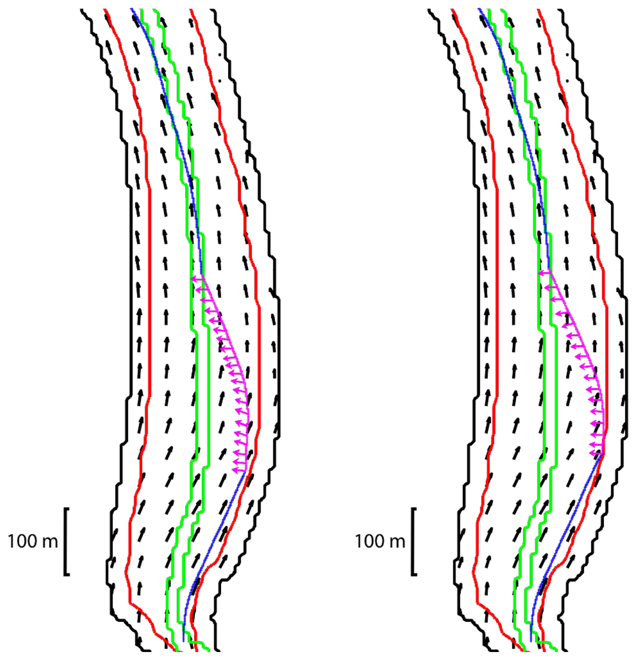


Figure 3.7: Simulated Trajectories with presence or absence of REALM flow field estimate. Left: known estimate. Right: no estimate incorporated.

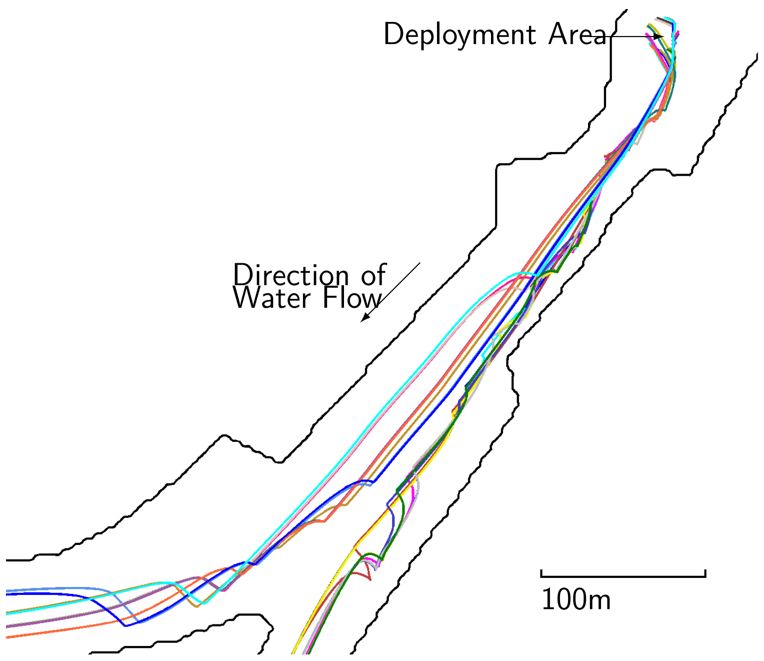


Figure 3.8: Trajectories of 20 simulated drifters performing path selection in Walnut Grove, CA. Drifters were deployed in a 20m diameter circle.

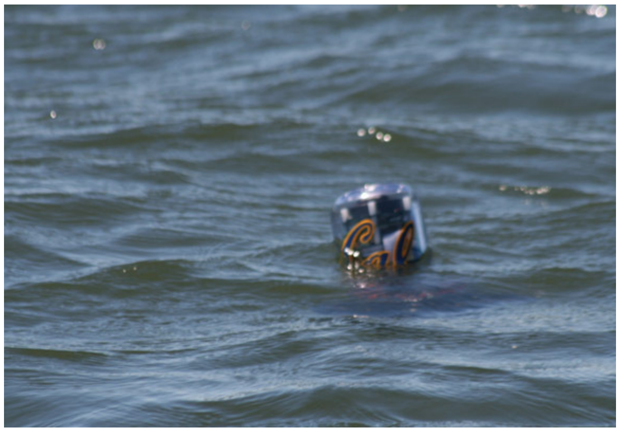
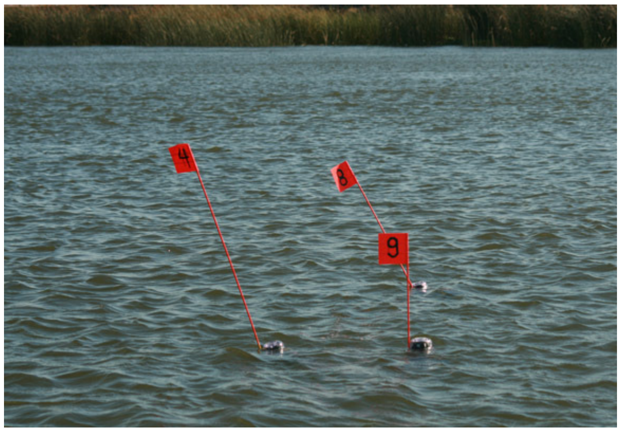



Figure 3.9: Left: drifter under motor power. Right: several drifters participating in experiment.

Figure 3.10 shows data from the field deployment that was gathered by one of the units. The trajectory of the unit has been reconstructed from GPS positions recorded on-board and plotted by the solid magenta and dotted blue lines, where the magenta lines and dotted blue lines indicate when the controller was in q_{actuate} and q_{drift}, respectively.

This result demonstrates behavior similar to that predicted by previously simulated results. During deployment, an easterly wind threatened to beach the drifters. Here, this drifter floats north with the river current, but is also being pushed towards the eastern shoreline. Upon crossing the V_{shore} threshold (red contour), it begins to maneuver back to safety. Once it reaches the V_{center} threshold (green contour), it transitions back to drifting without actuation. The current implementation appears to be sufficient for preventing collisions with the shoreline, but more advanced obstacle avoidance has yet to be proven in the field.

Path selection

In our second experiment, we operated at an interesting fork in the Sacramento river near Walnut Grove, California USA (approximately Latitude 38.24 N, Longitude 121.52 W. See Figure 3.5). Here, the Georgiana Slough meets the Sacramento river and diverts some quantity of water from it to be used in the delta. The large majority of the water, however, continues down the Sacramento river, taking any non-actuated drifters with it. Hence, our actuated drifters are needed to ensure some part of the fleet ends up traveling down the Georgiana Slough to measure that environment.

The primary goal of this experiment was to divert 10 out of 30 actuated drifters down the Georgiana slough, with the remaining 20 actively remaining in the Sacramento. By designing the proper obstacle map, $\mathcal{T}_{\text{shore}}$, we formed two parallel lanes for the drifters to split and stay within, before the actual split happened. This caused the group of drifters to clearly split into two groups and allows us to retrieve any malfunctioning units before they are in danger. In practice, the algorithm does not require that these lanes are drawn, only that an obstacle be drawn across unselected paths.

Figure 3.11 is a plot of the trajectories of two drifters, one from each group. The data represents a GPS location taken every two seconds by each drifter and passed through a two element moving average filter to remove sensor noise from the GPS system. Note that, unlike Figure 3.10, the drifters clearly enter the unsafe region, however, this does not indicate a failure to satisfy (3.2), since the drifter remained within $\mathcal{C}$. The figure demonstrates that the drifters have successfully actuated in a manner placing them in the correct lane and simultaneously avoiding obstacles.

3.5 Conclusions

The Floating Sensor Network has addressed the practical problem of mobile sensing in obstacle-laden rivers by building a fleet of motorized floating sensors. In this chapter, we described a successful technique for controlling our autonomous floating sensor platforms so that they avoid obstacles and the shoreline during a mission. We demonstrated, through simulation and field testing, the efficacy of the algorithm for the scenarios in which the unit must actively avoid running against the bank of a river, and in which the unit must drive itself down a particular fork of the river.

At the conclusion of this study, we achieved our primary goal which was to prevent obstacle collisions in the presence of a water current which was stronger than the control authority of the drifter itself. In fact, the primary reasons for failure of the drifters to avoid obstacles were hardware failure (e.g. a compass losing calibration or a motor failure), and unmodelled hindrances, such as seaweed. The amount of failures could be significantly reduced by redesign of the hardware and more detailed mapping of the region prior to an experiment. However, the results we have achieved clearly demonstrate that the technique is feasible. The Hamilton-Jacobi safety control techniques prove to be a powerful and effective tool in mobile environments where control authority is limited, as it can predict dangerous positions and act accordingly and optimally.

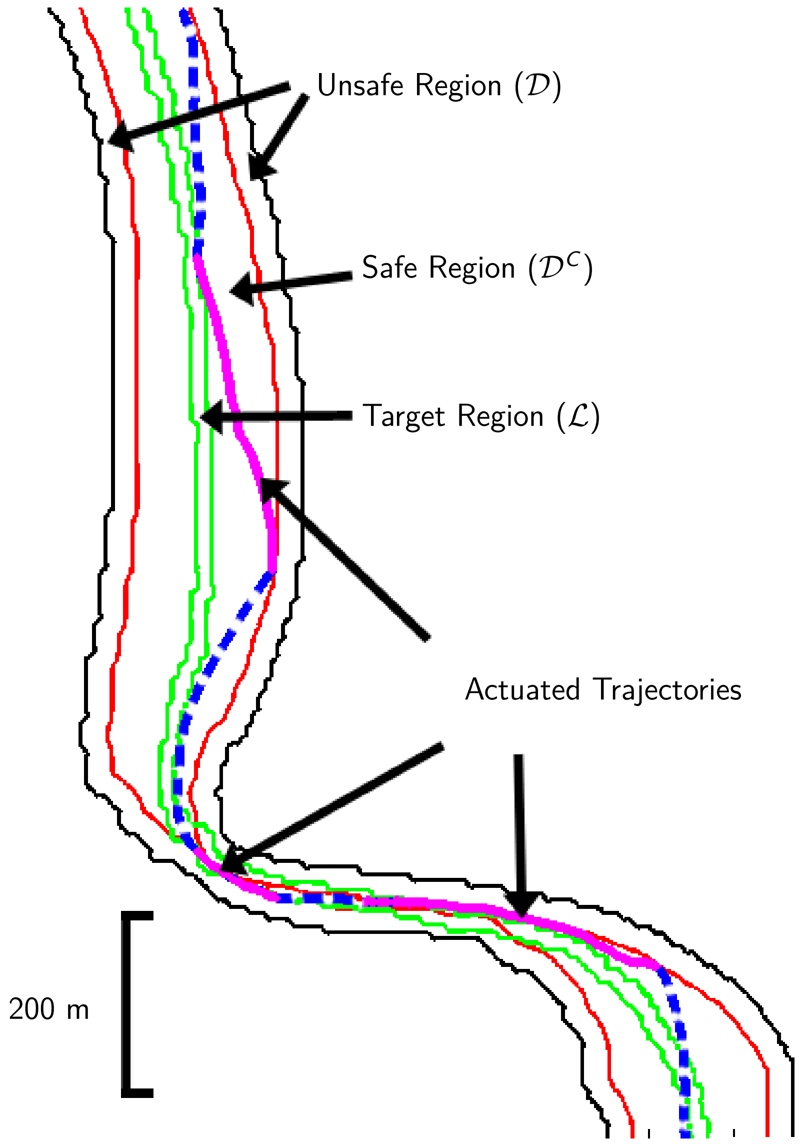

Unsafe Region ($\mathcal{D}$)
Safe Region ($\mathcal{D}^C$)
Target Region ($\mathcal{L}$)
Actuated Trajectories
200 m

Figure 3.10: Drifter GPS Trajectory during northward tidal flow. The red line is a contour of V_{shore} and denotes the edge of the unsafe region. The green line is a contour of V_{center} and define the target region. Along the drifter trajectory, dotted lines indicate unactuated motion.

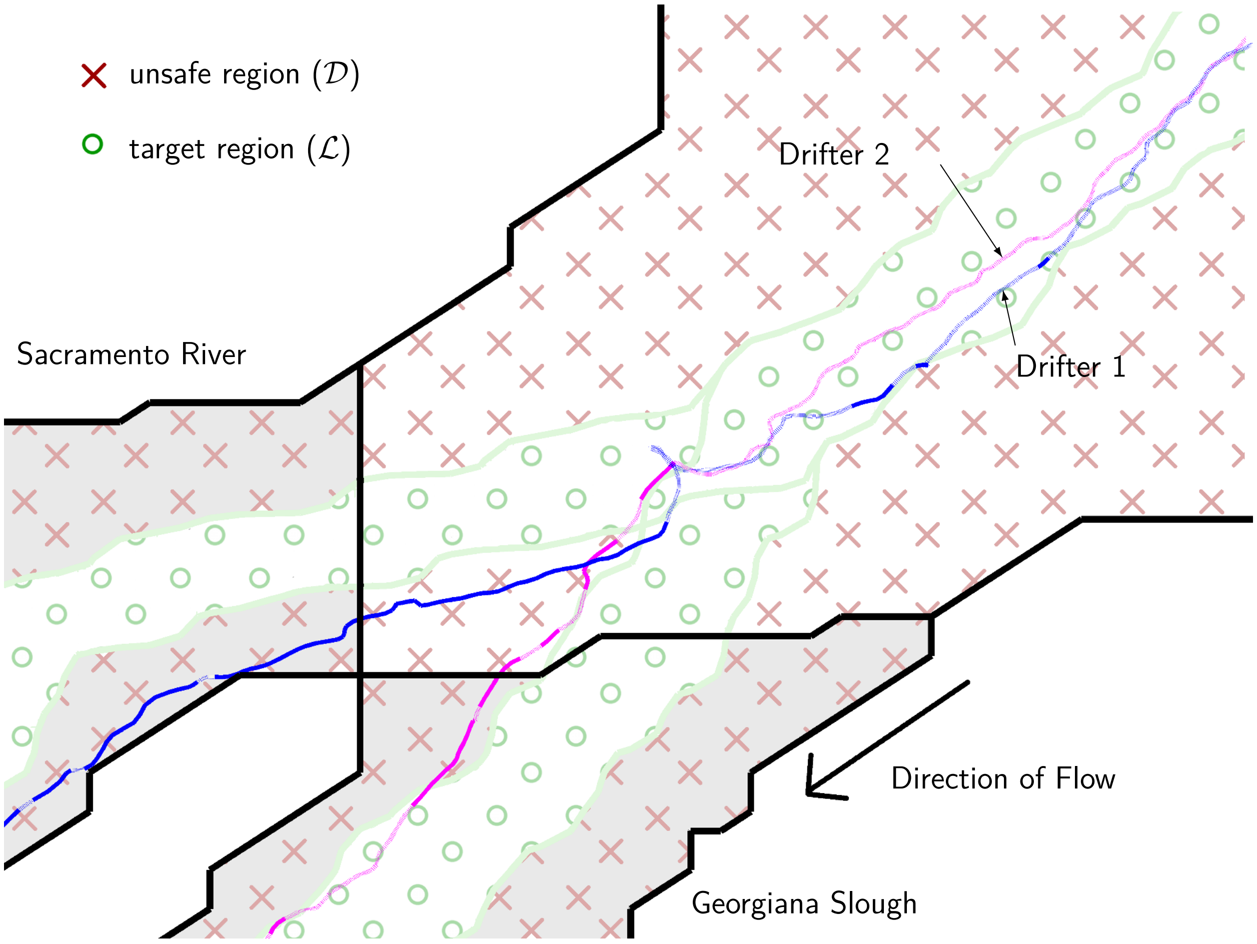


Figure 3.11: GPS Trajectories of two drifters performing the path-selection algorithm during a field test in Walnut Grove, CA. One of the drifters, shown by the blue trace, is tasked with proceeding down the Sacramento River, while the other drifter, in magenta, is tasked with proceeding down the Georgiana Slough. Along the trajectory, the faded segments indicate passive motion where the motors are off.

Chapter 4

Indoor Environmental Sensors for Mobile Sensing

4.1 Introduction

Mobile elements, particularly occupants, of indoor spaces play an important role in the operation of a building. The main purpose of an office building is to ensure the safety and comfort of its occupants, and recently, the goal is to accomplish this while consuming the least energy possible. Tailoring services such as Heating, Ventilation, and Air Conditioning (HVAC), lighting, and electrical power, has the potential to save a significant amount of energy consumed. HVAC and lighting respectively comprise 48% and 22% of the total energy use of buildings in the USA [53]. To illustrate the impact of occupant-awareness in controlling these services, occupant-aware control schemes have been shown to save between 10-15% [54], 8.3-28.3% [55], or even 42% [56], depending on factors such as outdoor climate and control strategy. Having a more detailed view of occupancy, by knowing where specific occupants are located, i.e. tracking, can enable control strategies on a person-by-person basis, and opens the door to more energy savings as well as building services which are tailored to specific occupants.

It is clear that, treated as individuals, or as a mass quantity, occupants are an import mobile element to be observed in a building. Therefore, we seek to answer how many occupants are in a space (i.e. occupancy estimation, Chapter 5) and where individual occupants are located (i.e. occupant tracking, Chapter 6). In the next three chapters, we will discuss the technological and theoretical developments which answer these questions. The latter two chapters discuss the estimation frameworks and techniques. As in the river current estimation problem, special tools were developed in order to sucessfully apply the estimation problem to the real-world. In this chapter, we introduce one of these tools, an environmental sensor platform which collects measurements needed for the occupancy estimation and tracking frameworks.

We have invented indoor sensors to support continuous and long-term estimation of occupancy and occupants. The sensor is a low-cost, battery-powered sensor which is small and light enough to be unobtrusively installed in an office space. The sensor can optionally be outfitted with CO_2 or particulate matter sensors, which are inputs to the estimation of occupancy level (See Chapter 5). The sensor could also be a "tag" which is carried in an occupant's pocket, providing information about that occupant's position. This information could be combined with other information via the sensor fusion capabilities of the particle filter (See Chapter 6).

Paramount to the adoption of an indoor sensing platform is battery lifetime, as it is costly to replace the batteries of a sensor, especially if the sensor is installed in a difficult to access location. For sensors which operate for less than a year, this cost can be unacceptable to a business and the platform will not be installed. We use a combination of strategies to reduce the power consumption of the battery-powered elements in our platform. First, we select peripheral components with very low standby current consumption (e.g. $< 10\,\mu A$) and relatively low active current consumption (e.g. $< 250\,mA$). Second, the components are heavily duty-cycled, leaving them in their standby state for long periods of time. For example, the component with the greatest power consumption, the radio, is used for about 5 seconds out of every hour. We also implement an efficient binary communications protocol which includes rudimentary data compression. Our stated goal is to achieve a battery lifetime of over 5 years, however we achieve a theoretical lifetime of over 6 years, calculated from current measurements taken from the device.

Our estimation framework benefits greatly from having frequent and regular measurements of environmental variables. Therefore, we stress the importance of reliable and high-volume communications between the sensor and server. We chose a WiFi-compatible radio transceiver which supports data rates over 200 times higher than IEEE 802.15.4 technology. Additionally, we chose the TCP/IP protocol stack to leverage its built-in retry mechanism for reliability. However, this choice necessitates a radio which consumes much more active power than traditional sensor nodes which use low-power IEEE 802.15.4 technology. The energy cost per bit can be reduced by storing a large amount of data and transmitting relatively infrequently (e.g. once per hour), thus, our sensor node sacrifices latency of receiving real-time measurements.

Many sensor network implementations are custom-tailored to the application, however there have been efforts to standardize architectures amongst the various efforts, such as the ZigBee Alliance [57], WirelessHART which have sucessfully achieved multi-vendor and multi-national interoperability between devices. We also recognize academic projects such as OpenWSN [58] and TinyOS [59] which incorporate the latest advancements in protocol design. Our resulting sensor architecture is differentiated from these other implementations primarily by this use of WiFi versus IEEE 802.15.4 technology. Additionally, these architectures usually provide for the transport of bytes from the sensor to the server, but provide only a small amount of guidance as far as what those bytes should signify (i.e. they do not specify the application layer). Therefore, we document the application protocol of our sensing architecture, which was designed to efficiently accomplish the sensing goal, while still flexible enough to incorporate additional types of lightweight sensors and actuators.

4.2 System architecture

Overview

There are three main agents involved in measuring, communicating, and storing the environmental readings: the environmental sensor devices, the local server, and the Internet server. Figure 4.1 illustrates how these agents are connected, as well as the relevant internal components of the local and Internet servers.

The *Environmental Sensor Devices*, described in detail in Section 4.5, connect to the *Local Area Network (LAN)* through an access point (the type depends on the wireless technology used). For our demonstration deployment, which uses WiFi technology, we chose the Ubiquiti Network's UAP-LR platform, which can deploy multiple APs managed by software on the Local Server.

The *Local Server* hosts the AP management software and a *Dynamic Host Configuration Protocol (DHCP)* server which assigns local IP addresses to the environmental sensors once they associate with the network. In our example deployment, we use a computer based on a Shuttle DS61 book-size PC which is physically small, yet powerful enough to process information from many remote sensors. Messages are both interpreted (via the "Protocol Interpreter") and forwarded to the Internet Server (via the "message splitter"). Measurements are stored on an on-board PostgreSQL server which also contains metadata about the incoming measurements. Therefore, the Local Server can act independently from the Internet Server, such as when deployed in a location without Internet access. A custom web frontend provides visualization of data as well as the ability to configure the remote sensors.

The *Internet Server* is similar in structure to the Local Server, however, instead of PostgreSQL, it uses an sMAP [60] database to store the collected readings, since it is more well-suited to the volume of measurements being collected. Additionally, the sMAP project provides a web frontend to allow quick visualization of the received data.

Local server

The purpose of developing a self-sufficient local server is to support the idea of a "building in a suitcase" sensor suite, where a sensor network can be rapidly deployed and begin long-term monitoring, without the need for external infrastructure. Our solution is a custom-built PC using a Shuttle DS61 motherboard and enclosure which neatly contains the features needed for our application:

- The small physical size ($190 \times 165 \times 43\,\mathrm{mm}$) and weight ($1.3\,\mathrm{kg}$) makes it easily carried in a handheld container to be brought onto the site in question.
- A metal chassis with dedicated mounting holes were originally designed as VESA-compatible in order to mount the computer on the back of a LCD monitor, however, we can use these holes to mount the computer inside a container, or on a wall or server rack.

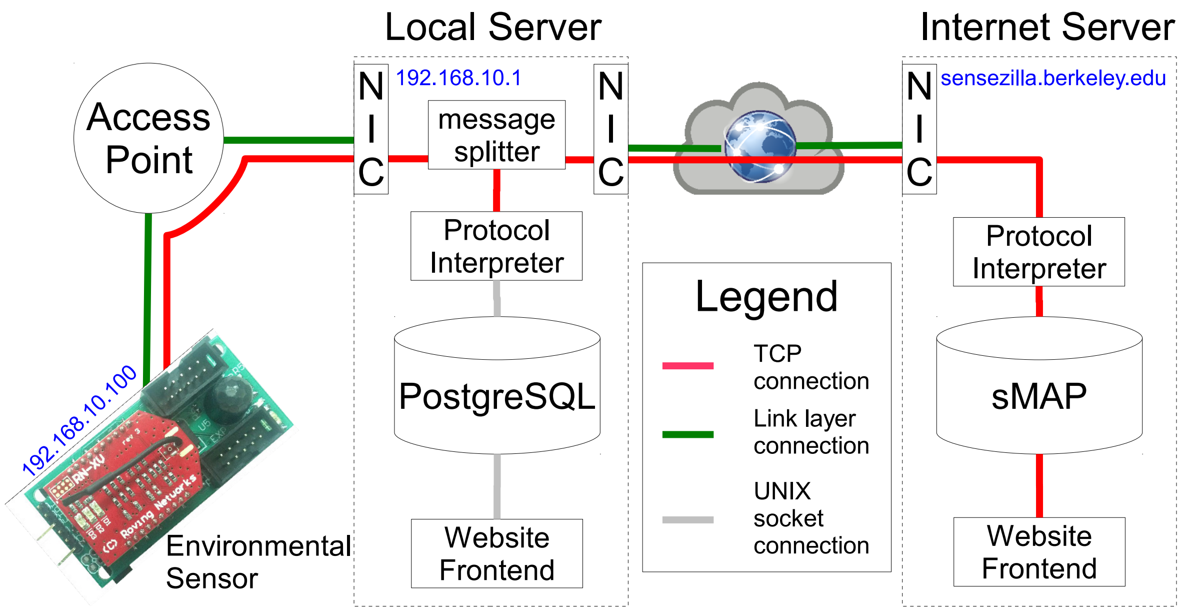


Figure 4.1: Network diagram showing the connections made during normal operation. The TCP connection from the sensor to the local server is only active while the sensor is transmitting a report.

- Two Ethernet *Network Interface Cards (NIC)* are included: one connects to the Access Point, and the other connects to the outside Internet.
- The low power consumption (maximum 100 W at the plug) of the computer allows the feasibility of solar or wind generation to power the computer if used in a remote environment where electrical power is unavailable.
- Additionally, we installed: An Intel Core i3-3220 (3.3GHz dual-core) CPU, a 1 TB hybrid hard drive (including 8 GB of FLASH memory), and 8 GB of system memory.

We rely heavily on open-source software to perform the needed functions and attempt to reduce the amount of custom-code to as little as necessary to perform application-specific functions. Notable software packages installed on the computer are:

Operating System: Ubuntu 13.10, a widely-used Linux distribution with many supported software packages.

DHCP Server: Dnsmasq, a lightweight DHCP, DNS, and TFTP server for routers.

Persistent Storage: PostgreSQL, a high-performance SQL Database.

Process Monitor: Supervisor, a process control system to start, stop, and log the output of processes. It is also capable of restarting processes when they crash.

Our custom code is written in **Python** and uses **SQLAlchemy** as a *Object Relational Mapper (ORM)* to map Python objects to rows in the PostgreSQL database. One piece of custom code, the "Protocol Interpreter" receives and interprets the messages from the sensor devices and inserts the decoded measurements into the database. It also uses the database to retrieve metadata about the devices, for example, which timeseries IDs are associated with a given WiFi MAC address.

Another piece of custom code, the "Website Frontend" is built on **Flask**, a light weight web application framework. Figure 4.2 shows some screen captures of the web application. Client-side scripting (such as plotting and animating timeseries) is written in **Javascript** and extensively uses the **JQuery** and **Flot** libraries. The frontend allows a person managing the network to view, at a glance, the sensor devices associated with the network and when the last report was received. There is also an interface to create configuration tuples consisting of **Name**, **Fields-to-Report**, **Sample Interval**, and **Report Interval** (See **Configure Sensor Command** in Section 4.4). This is useful, for instance, to easily switch between a "development" configuration (where nodes report in frequently at the cost of energy), and a "deployment" configuration (where nodes report in long intervals to reduce power consumption).

Internet Server

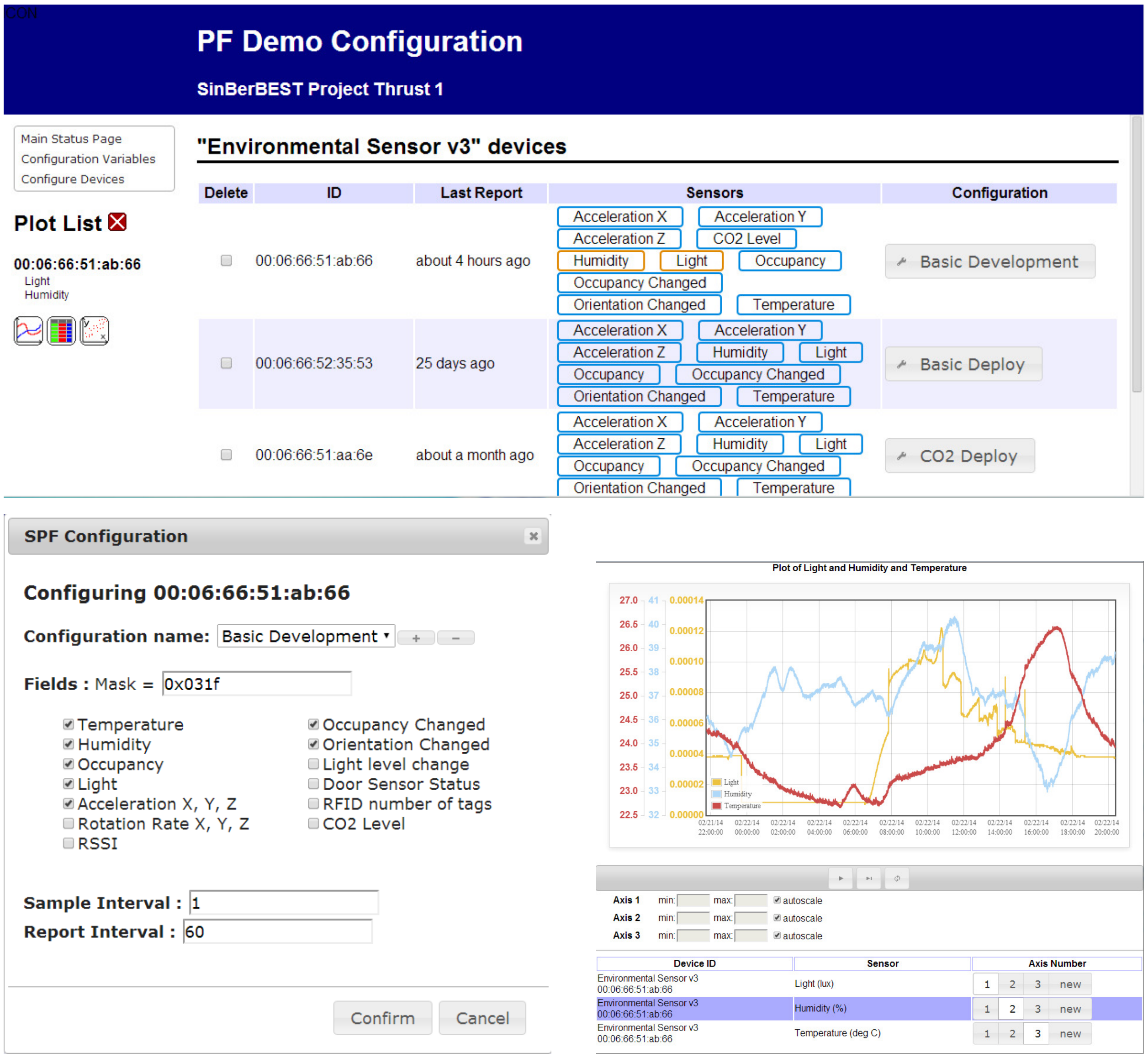
Like the Local Server, the Internet Server also runs an Ubuntu installation. However, the Internet Server is actually a virtual machine hosted by the Berkeley Information Services and Technology (IST) Cloud Computing services. This reduces the maintenance, setup effort and costs associated with provisioning a new server platform, as well as being more reliable, since generator-backed power and redundant storage is provided.

Unlike the Local Server, the measurement storage and website interface features are provided by the **sMAP** [60] project. The database used by sMAP, **ReadingDB** [61], is a database designed to efficiently store large amounts of timeseries signals, unlike a database such as PostgreSQL which is designed to store relational data. ReadingDB compresses and indexes the timeseries data which reduces the storage requirements, yet maintains quick insertion and retrieval of the data and minimizes corruption by using write-ahead logging.

Our custom software manages receiving messages forwarded from one or more Local Servers, interpreting the messages, and inserting any parsed data into the sMAP database. We use a PostgreSQL database installed on the Internet Server to store necessary metadata about the device, such as mapping WiFi MAC Addresses to sMAP stream identifiers.

4.3 Recordstore data format

Our sensor device relies on an inexpensive, low-cost, and low-power microcontroller with limited computational resources. Since we must store sampled measurements to be later



PF Demo Configuration

SinBerBEST Project Thrust 1

Main Status Page
Configuration Variables
Configure Devices

Plot List

00:06:66:51:ab:66
Light
Humidity

"Environmental Sensor v3" devices

Delete	ID	Last Report	Sensors	Configuration
☐	00:06:66:51:ab:66	about 4 hours ago	Acceleration X, Acceleration Y, Acceleration Z, CO2 Level, Humidity, Light, Occupancy, Occupancy Changed, Orientation Changed, Temperature	Basic Development
☐	00:06:66:52:35:53	25 days ago	Acceleration X, Acceleration Y, Acceleration Z, Humidity, Light, Occupancy, Occupancy Changed, Orientation Changed, Temperature	Basic Deploy
☐	00:06:66:51:aa:6e	about a month ago	Acceleration X, Acceleration Y, Acceleration Z, Humidity, Light, Occupancy, Occupancy Changed, Orientation Changed, Temperature	CO2 Deploy

SPF Configuration

Configuring 00:06:66:51:ab:66

Configuration name: Basic Development

Fields : Mask = 0x031f

- ☑ Temperature
- ☑ Humidity
- ☑ Occupancy
- ☑ Light
- ☑ Acceleration X, Y, Z
- ☐ Rotation Rate X, Y, Z
- ☐ RSSI
- ☑ Occupancy Changed
- ☑ Orientation Changed
- ☐ Light level change
- ☐ Door Sensor Status
- ☐ RFID number of tags
- ☐ CO2 Level

Sample Interval : 1

Report Interval : 60

Confirm Cancel

Plot of Light and Humidity and Temperature

Axis 1 min: max: ☑ autoscale
Axis 2 min: max: ☑ autoscale
Axis 3 min: max: ☑ autoscale

Device ID	Sensor	Axis Number
Environmental Sensor v3 00:06:66:51:ab:66	Light (lux)	1 2 3 new
Environmental Sensor v3 00:06:66:51:ab:66	Humidity (%)	1 2 3 new
Environmental Sensor v3 00:06:66:51:ab:66	Temperature (deg C)	1 2 3 new

Figure 4.2: Screen captures from the website frontend of the Local Server. Top: Shows the list of associated devices and when they last reported. Also allows selection of variables to plot and links to configure devices. Bottom-left: Configuration screen to set-up and change between device configurations. Bottom-right: Timeseries plotting screen which allows zooming, panning, and real-time animation to track incoming data.

reported to the server, limited memory space can restrict the amount of power we can save by limiting the number of measurements that can be stored.

For example, the ATmega1284 microcontroller that we use contains 16384 bytes of RAM, 2416 of which are used by our program. The average size of a message containing a measurement is 29 bytes (see **Sensor Data Report** format in Section 4.4), therefore, without compression, the maximum number of messages stored is 481. If a message is generated every 10 seconds, this corresponds to about 80 minutes of data stored. Thus, the data must be *successfully* delivered to the server within 80 minutes, otherwise some measurements must be lost.

Thus, we develop the *recordstore* data format which the device uses to store messages to be later transmitted to the server. This provides simple and efficient compression, taking advantage of the fact that sensor data messages are very often the same length and have many repeated bytes, especially at high enough sample rates where environmental conditions do not change much between samples.

In addition to the messages being stored in the devices' RAM in this manner, messages are also transmitted to the server in this format (in fact the contents of the memory are simply "dumped" onto the communications channel). This therefore reduces the communications load. An additional advantage of the recordstore format is that it is stream-based, meaning that previously inserted bytes do not need to be changed (although they are referenced). The primary disadvantage of this format is that it will perform very poorly for rapidly changing data, such as measurements taken with very long sample intervals. In these situations, more overhead bytes may be added than the number which are saved by compression. Another minor disadvantage is that the compression technique limits the size of each message to 128 bytes, however, we currently have no need to send messages this long. In an experiment where 500 measurements were taken, once every 2 seconds, the recordstore method achieved a compression ratio of 1.5:1, or a reduction in memory requirements of 30%. Primarily, this serves to allow more samples to be stored in the device's memory prior to a report being sent. If latency is not a concern, this compression can significantly increase battery life due to less reports needing to be sent to send the same amount of data. Compression of timeseries data is an interesting and useful direction of study, for which the algorithm described is only an initial result. Future work will attempt to compress the data further since it is so important to the battery performance of the sensor.

Figure 4.3 is an illustration of the data structure of the recordstore format. The *memory block* contains a list of *records*, each of which encapsulates one *message* (See Section 4.4 for the types of messages). There are two types of records: *template records* and *delta records*, represented in Figure 4.3 by the solid and hatched blocks, respectively. Additionally, to improve efficiency, a list of template pointers keeps track of the template records for quick referencing.

The template records take the following form:

7	6 0	
1	Length	Message Data

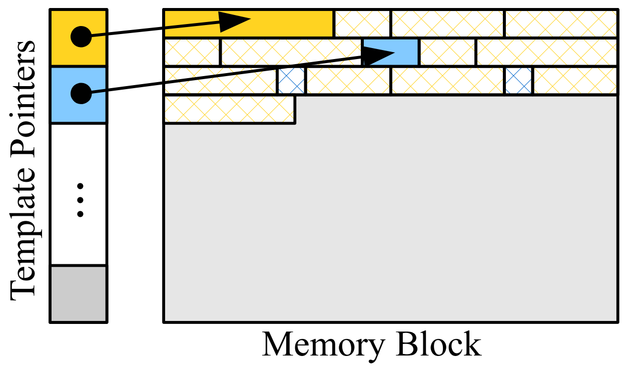


Figure 4.3: Diagram of the recordstore data structure in a typical scenario. There are two sets of records in this scenario, designated by the yellow and blue blocks. The solid yellow and blue blocks are the templates for that type of record and the hatched blocks are delta records which refer to the respective templates.

where **Length** contains the length, in bytes, of the **Message Data** field, which contains an exact copy of the message to be encoded.

The delta records take the form:

7	6 0		
0	Length	Changed Bytes Mask	Changed Bytes List

where **Length** contains the length, in bytes, of the *uncompressed* message to be encoded. This field directly determines which template record that the delta record corresponds to, where the **Length** field matches between the two. The **Changed Bytes Mask** field is a $\left(8 \cdot \lceil \frac{\textbf{Length}}{8} \rceil\right)$-bit bitmask specifying which bytes are different between the message to be encoded, and the message in the corresponding template record. The following **Changed Bytes List** field then provides only the bytes of the message which were not the same as the template. For example, **Changed Bytes List** = 0x11 means that only the first and fifth bytes were different between the template and the message to be encoded. Then, **Changed Bytes List** would contain two bytes: the first and fifth byte from the message to be encoded, in that order.

Encoding

Assume we are given a message, M, whose length in bytes is l, which must be inserted into the recordstore. The encoding procedure is as follows:

1. Look through the template records pointed to by the template pointers list. If the **Length** field of any template, T, is equal to l, then do 1b, however, if there are no such templates do 1a.

a) Create a new template record by prepending l as a 7-bit integer and the 1-bit value `1` to M. Insert this record at the end of the memory block and create a new entry in the template pointers list pointing to this new template record.

b) Create a new delta record starting with the 1-bit value `0` and l as a 7-bit unsigned integer. Create the **Changed Bytes Mask** field with a length of $\lceil \frac{l}{8} \rceil$ bytes and initialize all bits to `0`, and also create **Changed Bytes List** as an empty list. From the first to the last byte, compare the template T with the message M. If the byte at position p is different, then set the pth bit of the **Changed Bytes Mask** field to `1` and append the pth byte from M to the **Changed Bytes List**. After all bytes are compared, insert the record at the end of the memory block.

Decoding

Assume we are given a memory block of length L containing all of the records, in the order they were inserted. Let b_i represent the ith byte from the memory block, and let P : (lengths) $\rightarrow$ (messages) be an initially-empty map. The goal is to generate a list, D, of decoded messages, which is initially empty. The decoding procedure, starting at $i = 1$, is as follows:

1. If $i \geq L$, then we are done. Otherwise, retrieve the length l from the 7 least significant bits of b_i. Look at the MSB of b_i, if it is `1`, then do 1a, otherwise do 1b.

   a) The record is a template record. The decoded message, M are the bytes b_{i+1} through b_{i+l}. Insert M into D as a decoded message and set the map $P(l) \leftarrow M$. Set $i \leftarrow i + 1 + l$ and continue with 1.

   b) The record is a delta record. Retrieve the template message from the map and initialize the decoded message to an exact copy: $M \leftarrow P(l)$ (it is an error if no mapping yet exists). Retrieve the **Changed Bytes Mask** bitfield from the next $\lceil \frac{l}{8} \rceil$ bytes, and create a list $O = \{o_1 \ldots o_n\}$ of the bit-positions set to `1`. Retrieve the **Changed Bytes List** from the next n bytes. For each o_j in O, set the o_jth byte of M to the jth byte of the **Changed Bytes List**. Insert M into D as a decoded message. Finally, Set $i \leftarrow i + 1 + \lceil \frac{l}{8} \rceil + n$ and continue with 1.

4.4 Communications protocol

We developed a custom binary format for sensors to report their readings to the database server. Our motivations are the following:

- The protocol should be simple enough to encode and decode on a microcontroller without consuming very much power. Thus, we chose to use a binary protocol relying on native C data types such as integers and IEEE 754 floating point numbers.

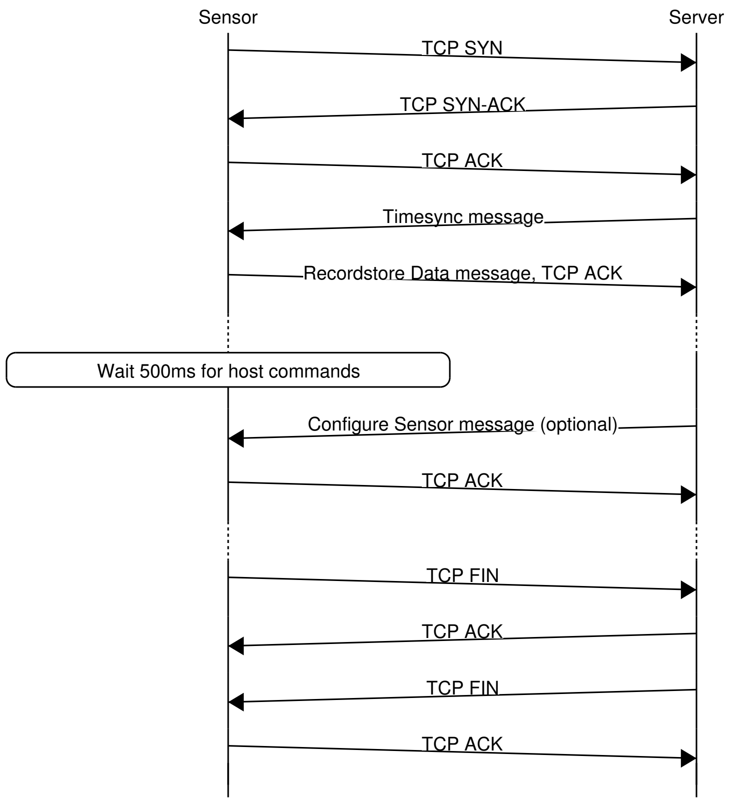

Sensor
Server
TCP SYN
TCP SYN-ACK
TCP ACK
Timesync message
Recordstore Data message, TCP ACK
Wait 500ms for host commands
Configure Sensor message (optional)
TCP ACK
TCP FIN
TCP ACK
TCP FIN
TCP ACK

Figure 4.4: Typical exchange between sensor and server during one report, showing packets at the transport layer and above.

- We should also balance extensibility of the protocol to add new types of messages and sensor data without the protocol becoming overly complex. Our compromise was to have fixed-length fields specifying the type of hardware, type of message, and types of sensor data present in each message. The meaning of the field values must be known across all agents that wish to encode and decode the messages.

- To reduce power consumption, the protocol should reduce the number of packets that need to be sent and minimize the amount of time the radio needs to be active. Thus, our protocol incorporates "batching" many sensor samples into one compressed report which is sent to the server infrequently. Section 4.3 describes the recordstore technique designed to accomplish this.

With these considerations in mind, we develop a simple, yet easily extensible, protocol for the environmental sensors. Our protocol is essentially a custom binary format relying on the TCP/IP networking stack to deliver bytes from the sensor to the server and from the server to the sensor. Figure 4.4 illustrates a typical packet exchange between the sensor devices and the server (at the transport layer and above) for one sensor report interval.

Lower-layer considerations

Our protocol relies on a layered communications model where lower layers ensure certain properties about the connection. Our protocol requires that the lower layers guarantee a reliable and error-free transmission of bytes between agents on the network, or, if this cannot be guaranteed, to inform our application as such. Thus, we chose to build our protocol as an application atop a TCP/IP stack. The TCP/IP stack extensively uses acknowledgement packets to validate the successful transmission of data, which is important to ensure that our sensor data is not lost, since we can later re-send the data if a transmission failed. In practice, if the current link is reliable enough to establish a connection, then we assume the link is also ready to transmit data, whereas if the link does not survive the initial TCP handshake, we assume that the link is *not* reliable and save the data for later retransmission. While we have observed this assumption to be correct while developing the sensors, we have yet to perform any extended tests to validate the assumption.

Thus, TCP is chosen for its built-in mechanisms to ensure data reliably reaches the server, however, this comes with a major drawback. Since TCP relies on many separate control packets to establish and tear-down a connection (at a minimum 7, See Figure 4.4), this requires that the radio to be active for the entire time. Thus, approximately one-third (See Figure 4.9) of the sensor's battery life is expended by the radio alone, even at a very infrequent reporting rate. Future work will be to use the UDP protocol for which packet reliability is not enforced by the protocol. We can add a custom acknowledgment method which requires less overhead than TCP. For example, the server could send a single acknowledgement to the sensor upon receiving a data message, rather then setting up and tearing down an entire TCP connection.

Message framing and escaping

Since TCP is a stream-oriented protocol, it guarantees that bytes are delivered reliably and in order (otherwise delivering an error). However, TCP does not provide framing utilities to designate discrete sets of bytes that should be treated as one message. We considered the following options when faced with the task of framing messages in the TCP stream:

Connection-based framing

The first method is to constrain each TCP connection to only contain one message, therefore, simply initiating and terminating the TCP connection signals the start and end of a message. This is simple to implement, and requires no extra overhead (i.e. extra energy consumed) when only one message needs to be sent per reporting interval. However, the overhead is very significant when needing to transmit more than one message since the TCP connection must be reconnected, and the radio must be awake during that time.

Length-based framing

The second method is to append each message with a 16-bit length field, l, of the message which follows. Thus, the following l bytes are treated as a single message, and the l-th and $l+1$-th bytes are the length field for the next message. Thus, the overhead is exactly 2-bytes per message. The main drawback to this method is that errors such as dropped bytes are nearly irrecoverable, as are errors in accurate transmission of the length fields. This is because any mismatch between the received length field value and the actual length of the following message will cause the wrong bytes to be read and interpreted as the length field of the next message.

Delimiter-based framing

The third method, and the chosen method for framing our messages, utilizes a delimiter (we use the hex value `0x0A`, i.e. the linefeed character) to indicate the boundaries between messages, by appending this delimiter to the end of every message. Thus, there is only one byte of overhead per message, and communications errors are generally recoverable for multiple message streams, as the next correctly received delimiter will be correctly interpreted. The method described is inspired by, and is very similar to, the High-Level Data Link Control (HDLC) protocol [62].

For messages which do not contain the delimiter character, `0x0A`, no further change is necessary, however, since we cannot make this assumption, an additional *escaping* step is necessary to ensure that any part of the payload is not misinterpreted as the delimiter character. Therefore, we make the following substitutions when the payload is sent:

$$\texttt{0x0A} \Rightarrow \texttt{0x7D}, \texttt{0x2A}$$
$$\texttt{0x7D} \Rightarrow \texttt{0x7D}, \texttt{0x5D}$$

The receiver, after splitting the stream into discrete messages using the delimiter characters, then makes the reverse of the above substitutions in order to recover the original messages. In this way, any bytes in the message do not interfere with the framing and escaping mechanisms. The escaping process adds some amount of overhead, about 1 extra byte for every 128 message bytes if the message is random data, or, in the worst case, doubling the message size (if all message bytes are `0x0A` and `0x7D`).

Message descriptions

All messages sent between the sensor and server are in the following format:

BT	**MT**	Message Data

containing the board-type **BT** and message-type **MT** fields, as well as an arbitrary-length data field which varies depending on the value of **MT**. For the environmental sensing platform, **BT** is always equal to 2.

Below, we describe the most commonly used message types, where “Command” refers to messages usually sent from the server to the sensor, and “Report” refers to messages usually sent from sensor to server. We assume that the byte ordering of all integers is little-endian.

Timesync Command (MT=`0x00`)

`0x02`	`0x00`	Unix Timestamp

This command is sent to the sensor to notify it of the current time. The time is given as a 32-bit integer representing seconds elapsed since January 1, 1970. For example, January 1, 2014 is equivalent to a timestamp of `1388563200`. The current implementation does not incorporate timezones, and so all times are given by the local timezone of the server.

Sensor Data Report (MT=`0x01`)

`0x02`	`0x01`	Unix Timestamp	Fields Present	Field 1	Field 2	...

This message is for the sensor to report environmental measurements. The **Unix Timestamp** is in the same format as described above for the **Timesync Command** and represents the time that the sample was taken for the subsequent fields. The **Fields Present** field is a 16-bit bitfield indicating what types of measurements are present in the following list of fields. If a bit is set in the bitfield, then that means that the field is included in the following list of fields, with the lowest bit-indexed field coming first. The meaning of each bit of **Fields Preset** is board-dependent (i.e. the value of **BT**); for the environmental sensor, **BT** = 2, the meaning of the bits are as follows:

Bit index	Type	Units	Binary Format
0	Temperature	°C	32-bit IEEE 754
1	Humidity	RH%	32-bit IEEE 754
2	Occupancy	%	32-bit IEEE 754
3	Ambient Light	lux	32-bit IEEE 754
4	3-axis Acceleration	g	3 fields, 32-bit IEEE 754
5	3-axis Angular Rotation Rate	$^\circ\,\mathrm{s}^{-1}$	3 fields, 32-bit IEEE 754
6	Last Received Signal Strength	dBm	Signed 8-bit integer
7	Not implemented		
8	Occupancy State Changed	boolean	Unsigned 8-bit integer
9	Orientation Changed	Code	Unsigned 8-bit integer
10	Light Level Changed	boolean	Unsigned 8-bit integer
11-15	Not implemented		

The **Occupancy State Changed** field is sent with value `1` whenever human activity is detected (via the PIR sensor) after 10 seconds of inactivity, and sent with value `0` whenever human activity is not detected for 10 seconds, after previously sending the value `1`.

The **Orientation Changed** field is sent whenever the sensor device changes orientation so that any of its axis are aligned, or closely aligned with the gravity vector, and the other two axis are parallel to the ground. The following codes are defined:

Code	Orientation	Code	Orientation
`0x02`	X-axis pointed UP	`0x01`	X-axis pointed DOWN
`0x08`	Y-axis pointed UP	`0x04`	Y-axis pointed DOWN
`0x20`	Z-axis pointed UP	`0x10`	Z-axis pointed DOWN

The **Light Level Changed** field is sent with value `0` any time the ambient light is darker than a certain level (approximately the light level inside a closed desk drawer), and sent with value `1` whenever the ambient light is above that level. There is a small amount of hysteresis applied to prevent chattering.

Typically, the sensor periodically samples the measurements and generates **Sensor Data Report** messages every **Sample Interval** seconds. However, the **Occupancy State Changed**, **Orientation Changed**, and **Light Level Changed** fields are *asynchronous*, meaning that a **Sensor Data Report** message for these fields is generated exactly whenever the appropriate event occurs (to a resolution of 1 second). This allows us to capture events which happen in between the sample intervals.

Configure Sensor Command (MT=`0x02`)

`0x02`	`0x02`	Variables Present	Variable 1	Variable 2	Variable 3

This command is used to configure the sensor, mainly to allow remote configuration of variables that affect the battery lifetime of the device. This message is extensible to allow up to 8 configuration variables to be defined. The **Variables Present** bitfield is currently implemented as:

bit 0 **Fields-to-report** variable present.

bit 1 **Sample Interval** variable present.

bit 2 **Report Interval** variable present.

bits 3–7 Not implemented.

If a bit is set in the bitfield, then the corresponding variable is present in the following fields of the message, with the lowest bit-indexed variable coming first. For example, if

Variables Present = 0x06, then the **Sample Interval** and **Report Interval** fields are present, in that order, following the **Variables Present** field.

The **Fields-to-Report** variable is a 16-bit bitfield instructing the sensor to measure and report the environmental fields corresponding to the bits set. This can allow the sensor to reduce energy consumption by turning off the measurement hardware for the fields that are not needed. The bits have the same meaning as the **Fields Present** bitfield described above for the **Sensor Data Report** message. Therefore, the **Fields Present** field of the subsequent **Sensor Data Report**s coming from the sensor should match the value of **Fields-to-Report** given by the server.

The **Sample Interval** variable request is a 16-bit unsigned integer specifying, in seconds, the time that the sensor should wait before taking periodic measurements from the environment, such as light level or temperature (this does not apply to asynchronous measurements, such as occupancy level change detection). A typical value for this is 20 seconds. Increasing this value will increase battery life at the cost of temporal resolution of measurements.

The **Report Interval** variable request is a 16-bit unsigned integer specifying, in seconds, the time that the sensor should wait before attempting to contact the server to receive commands and send the latest batch of measurements. A typical value is 3600 seconds, or 1 hour. Increasing this value increases battery life at the cost of latency before environmental measurements can be used.

Current Configuration Report (MT=`0x03`)

`0x02`	`0x03`	Fields-to-report	Sample Interval	Report Interval

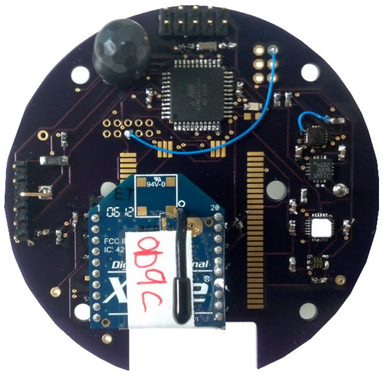
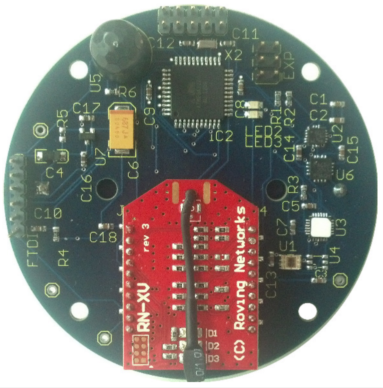
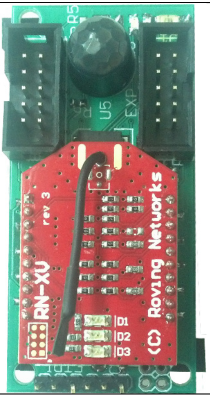
This reports to the server the current configuration of the sensor device. If there is a mismatch between the reported configuration variables and those desired by the user, then a **Configure Sensor Command** is sent by the server to correct the mismatch.

Recordstore Data Report (MT=`0x05`)

`0x02`	`0x05`	Recordstore Data

In this message, **Recordstore Data** is a compressed set of many more messages (See Section 4.3 for a description of the format). In fact, in our implementation, messages from sensors to the server are *always* wrapped in a **Recordstore Data Report** which is decompressed into individual messages on the server. This saves an average of 33% in communication load.

Table 4.1: Iterations of the Environmental Sensing Devices

		
Version 1	**Version 2**	**Version 3**

Device Identifier Report (MT=`0x06`)

`0x02`	`0x06`	ID Type	Device Identifier

This message is used by the sensor to report its ID to the server and is typically included with each sensor measurement message. For the environmental sensor, **ID Type** is set to `0x01`, meaning that the following identifier should be interpreted as a 48-bit WiFi MAC Address, and the device identifier is the 48-bit binary-encoded Wifi MAC Address provided by the wireless module. A future extension will be to allow human-readable names to be set on the sensors, such as "Room 406A, Supply Vent CO_2 Sensor", using **ID Type** = 0x40.

4.5 Environmental sensor devices

The most visible and numerous element of the environmental sensing platform are the environmental sensing devices which are the instruments responsible for collecting physical measurements and delivering them to the server. These are hardware devices designed from the ground-up, although they share design elements found in the hobbyist community, such as the Arduino [63] design. Many wireless sensor nodes have been developed by the research community to accomplish the task of continuously and remotely monitor an environment [64, 65, 66, 67, 68]. We see this plethora of new designs as exciting evidence of iterative improvement in sensor designs, incorporating novel ideas and new technologies.

Our design represents another incremental step towards a permanent and easily-deployable sensing infrastructure that can operate for many years inside a smart building. Notably, by using Microchip's RN-XV [69] low-power IEEE 802.11 (WiFi) module, we are one of the first to use WiFi technology for this application. In the past, WiFi has been overlooked in favor of other radio protocols, such as IEEE 802.15.4 and Bluetooth, due to WiFi's high active power consumption. However, as WiFi hardware becomes lower power (driven by the venerable advancement of smartphones), WiFi is gaining attention as a contender to supply the communication requirements of wireless sensor networks (WSN)[70].

Our hardware design has proceeded through several iterations to reach its current form (See Table 4.1). Since *Version 1* was our first experience with the physical implementation of the design, it had several hardware errors that needed to be fixed. In *Version 2*, we corrected these errors, allowing us to develop a codebase that would be compatible with *Version 3*, in which we added further expansion capabilities, reduced the physical size and significantly reduced the cost of production (by using 2 routing layers instead of 4, and a rectangular outline).

Commercial alternatives

Prior to designing a custom hardware solution, we first evaluated alternative commercial solutions, some of which are listed in Table 4.2. Some platforms, such as the Digi XBee Sensors[71], are commercially packaged, produced and sold through major distributors, which makes it extremely quick and convenient to build a platform upon. However, none fulfilled all our requirements of variables sensed, battery life, or cost. Moreover, proprietary solutions prevent us from extending the sensors to measure more variables, installing experimental networking protocols, and are also vulnerable to becoming unsupported by the company.

There are also many WSN-centric development kits intended for research and development, perhaps the most referenced and studied being the Telos and TelosB platforms[66]. These are somewhat unfinished products and require some development of software, hardware, and mechanics to fit the parameters of our deployment. They also rely on an experimental and changing code base, such as TinyOS, and requires a higher level of knowledge to configure, as opposed to simple and user-friendly interfaces such as the X-CTU utility by Digi. However, with this cost comes the benefit of being able to keep up with improving protocols and a high amount of customization.

Finally, there are proof-of-concept and short-run products such as the Powercast WSN-1101[72] (and derivatives), which feature remarkable improvement in battery life and package size. Sensor nodes like these are usually developed by companies to demonstrate an underlying technology innovation (in this case, wireless charging, low-power micro-controller, and energy storage improvements). In many cases, the true product is an OEM module that is sold to other companies to integrate into a commercially packaged product.

Hardware design

Designing the environmental sensor from the ground-up afforded us a great amount of flexibility in choosing state-of-the-art components available. As an overview, the block-level diagram of the Environmental Sensor node is shown in Figure 4.5.

Table 4.2: Feature table of sample of environmental sensors.

Digi XBee Sensors[71]	**Telos Platform**[66]	**Powercast WSN-1101**[72]
Measurements Temperature, Humidity, Ambient Light	**Measurements** Temperature, Humidity, Ambient Light	**Measurements** Temperature, Humidity, Light, CO_2 (optional)
Battery Type 3 Alkaline AA Cells (4.5V 2700mAh)	**Battery Type** 2 Alkaline AA Cells (3V 2700mAh)	**Battery Type** Integrated Lithium OR Radio Power Transfer
Battery Lifetime 1.5 years (1/30 Hz rate) 2.5 years (1/60 Hz rate) 6 years (1/3600 Hz rate)	**Battery Lifetime** 3 years (1% duty cycle)	**Battery Lifetime** (1/60 Hz rate) 25+ years (battery) Perpetual (RF-powered)
Communications 2.4GHz IEEE 802.15.4 ZigBee mesh network	**Communications** 2.4GHz IEEE 802.15.4 TinyOS mesh network	**Communications** 2.4GHz IEEE 802.15.4 Proprietary mesh
Cost 109 USD	**Cost** 110 USD	**Cost** 200–400 USD

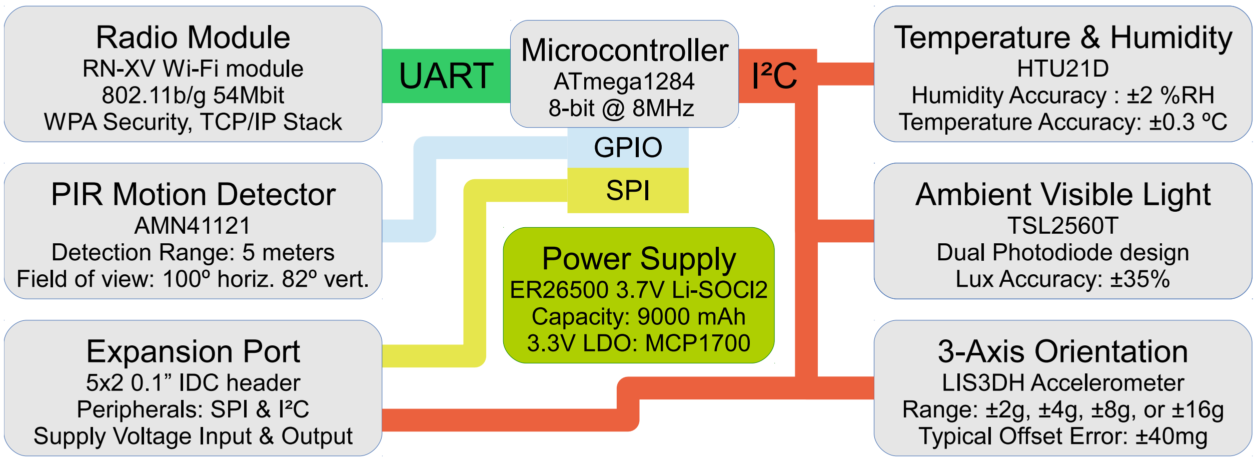


Figure 4.5: Environmental Sensor system diagram illustrating the major on-board instruments and expansion capabilities. Communications buses are indicted by solid-color paths.

Microcontroller

At the logical center of the design is Atmel's ATmega1284 microcontroller. This particular choice has a high amount of program memory (128KB) and RAM (16KB), which allows us to add in many code features without running out of space. However, most importantly, the microcontroller has the peripheral features required (I^2C, SPI, and two UART ports), and the capability of low-power sleep while operating a timer from a 32.768KHz crystal. Even though there are many microcontroller families (e.g. TI's MSP430, Microchip's PIC) which satisfy our requirements, an additional reason for choosing the ATmega family is due to its reliance on an open-source and free toolchain (GNU GCC) and extensive community support through the Arduino community (e.g. the community provides driver code for many sensors).

Communications

A necessary component for a wireless sensor is the wireless transceiver. Initially, we did not seek to tie our design to a specific wireless module, therefore we included a universal 20-pin socket, popularized by the ubiquitous XBee module. This is the connector footprint that is defined by many of Digi's OEM XBee modules. Due to the popularity of the XBee module, many other companies produce modules which adhere to the same connector footprint and signal locations, therefore we allow ourselves to evaluate these alternative offerings without a hardware redesign of the board.

Of these alternatives, we are particularly interested in integrating the OpenMote [73] since it incorporates the IEEE 802.15.4e stack via the OpenWSN project [58]. IEEE 802.15.4e uses Time Synchronized Channel Hopping (TSCH) to combat narrow-band challenges and achieve 99.999% end-to-end reliability [74]. Therefore, when the OpenMote has reached maturity, it offers to address our reliability concerns while achieving much lower latency and power than our WiFi selection.

Ultimately, we chose Microchip's RN-XV[69] module which, in our experience, proved more reliable than the XBee offering. Although the power consumption of this WiFi transceiver is up to 10 times greater than Bluetooth or IEEE 802.15.4 products, the high data rate (over 200 times faster) allows us to transmit many measurements in one "burst", thus limiting the time that the radio must be turned on.

Power supply

In the standard configuration, the sensor is powered via a 3.7-volt Lithium-Thionyl Chloride C-cell battery with a nominal battery life of 9Ah. This particular chemistry is intended for powering long-lifetime devices, having a low self-discharge of less than 1% per year. However, the capacity of the battery is significantly reduced when pulling more than 2 mA of continuous current or more than 400 mA of pulsed current (which could damage the battery). The capacity of the battery is also reduced at lower current levels, such that at a 100 µA continuous current draw (roughly the average draw of the sensor), the effective capacity is roughly 8Ah. In our design, a 100 µF capacitor is used to lightly buffer the current draw during the times that the radio is active.

The 3.7V battery voltage is dropped to the system voltage of 3.3V via a linear regulator. Therefore, instead of the battery, the system can also be supplied with another power source of up to 6V and as low as 3.5V. For proper operation, the external supply should be able to supply current up to 250 mA (@12 dBm) or 140 mA (@0 dBm) during transmission, and, on average 200 µA.

Temperature and humidity sensing

Measuring Temperature and Humidity from the local environment is accomplished by Measurement Specialties' HTU21D [75] instrument. This instrument is attached to the shared I^2C bus and can be queried to provide 12-bit and 14-bit digital readings of relative humidity and temperature, respectively. The stated accuracy of the humidity sensor is ±2%RH typical over the 20%RH to 80%RH range, and up to ±3%RH outside of this range. The maximum error tolerance is stated to be ±5%RH over the whole range. Although the instrument is calibrated at the factory, the relative humidity reading must be temperature-compensated to achieve the stated accuracy. The coefficients and formula needed for this correction are specified by the datasheet. The stated accuracy of the temperature sensor is typically ±0.3 °C and maximally ±0.4 °C over the range of roughly 5 °C to 60 °C which well covers the expected range of temperatures encountered indoors.

Ambient light sensing

Ambient visible light is measured by AMS' TSL2560 instrument [76]. This design of the IC incorporates two light-sensing photodiodes: one measures visible and IR light from 300 nm to 1100 nm, and the other measures IR light from 500 nm to 1100 nm. Therefore, the reading from the second (IR only) photodiode can be used to compensate for the light energy that the first (visible and IR) photodiode measures, but is not visible to the human eye. An additional feature of the instrument is its ability to change the integration time of the on-board analog-to-digital converter (ADC). This is typically changed to adapt to various light conditions, such as a long integration time for dark environments, and short integration time for outdoor environments. In our implementation, which is mostly to be deployed indoors, we have fixed the integration time to 101 ms. The light sensing instrument is attached to the shared I^2C bus for configuring the instrument and reading the 16-bit values measurements of the two photodiodes. The microcontroller uses the two measurements to calculate a lux reading, based on a relation described in the TSL2560 datasheet. We also used the interrupt feature of the instrument, which detects when the light level crosses above or below a pre-configured threshold. When enabled, this allows our sensor to timestamp events such as when the sensor is put inside a drawer or the lights are turned on and off.

A particular integration note about this instrument is that the responsivivity to light is somewhat directional (120° beam-width at half-maximum). Therefore, the choice of enclosure and mounted configuration of the environmental sensor can dramatically affect the ambient light readings. For practical studies, it would be prudent to ensure that all of the sensors are mounted in a similar way to properly measure the light levels in the indoor space. Alternatively, we should study the changes in light intensity over time as opposed to emphasizing the absolute lux values of light.

Orientation sensing

Although capable of accurately measuring rapid acceleration, we include the LIS3DH Accelerometer by ST Microelectronics [77] primarily to measure the orientation of the device, with respect to gravity, and secondarily to measure high-acceleration "bumps", such as footsteps. Detecting orientation and bumps provides information in a mobile environment, such as if the device is in the pocket of a mobile human occupant. The number of footsteps taken and whether the person is sitting or standing can be extremely valuable pieces of information in an occupant tracking or occupancy estimation problem. In these situations, the accuracy of the instrument is not of large concern.

This instrument features low-power (6 µA at 50 Hz sample rate) continuous operation, which allows us to capture and timestamp events which occur in between periodic samples. For orientation purposes, the typical magnitude of acceleration we are measuring is 1 g, so the stated offset error of 40 mg represents about a 4% error. Additionally, there is some sensitivity to temperature which is stated as 0.01 %/°C. The 3 16-bit values of acceleration are read from the device over the shared I^2C bus. Additionally, we use the interrupt capabilities of the device to inform the processor when the orientation has changed (the x, y, or z measurement axis is aligned with the gravity vector).

Motion detection

We measure motion using Panasonic's AMN41121 passive infrared (PIR) motion detector module [78]. This module utilizes a pyroelectric element to monitor small changes in infrared black-body radiation emitted by humans (7-14 µm). There is a specialized lens which forms a pattern of discrete detection zones corresponding to one of four sensing regions of the pyroelectric element. Further circuitry within the device monitors changes in the infrared light measured by the four regions to determine whether a detection event has occurred. Ultimately, the sensor will detect when an infrared-emitting body, such as a human moves across the detection zones, but remain insensitive to overall temperature increases or decreases within the field-of-view. In practice, we have found that the sensor is also insensitive to an unmoving human, i.e. some movement across the detection zones is necessary to signal a detection.

The field-of-view of the sensor is stated to be 100° along one axis and 82° along the other, and the detection range is at least 5 m from the sensor. Within this cone, there are 64 discrete detection zones distributed somewhat uniformly. There are three other models within the family line of sensors, each having different lenses and correspondingly different fields-of-view and detection ranges. Since these other models are pin-compatible, we could easily switch out the AMN41121 for one of these alternative lens configurations, depending on the end application. A drawback of the module is that it needs at least 7 seconds for the circuit and sensor to stabilize before the detection is reliable. Therefore, we realistically cannot duty-cycle the sensor to save power, however, since the static power consumption is only 46 µA, we can leave the sensor powered and still achieve a long battery lifetimes.

The module interfaces to the microcontroller via a single output which connects the attached signal to V_{dd} (HIGH) when a detection occurs. An external pull-down resistor pulls the signal to GND (LOW) otherwise. Although the datasheet does not specify the timing of the signal during a detection, we have found that, the signal will stay HIGH as long as there is activity, but can sometimes intermittently go LOW, even while humans are moving in front of the sensor. Therefore, we require some intelligent interpretation of the signal, rather than simply assuming that a HIGH signal means humans are present and a LOW signal means humans are not present.

Our strategy is to report two calculated measurements: the first is the *occupancy* percentage, which is the percentage of time (with a resolution of 1/256 seconds) that the signal was HIGH over the last sample interval. The second is the *occupancy state changed* event, which sends a value of `1` if the signal goes HIGH after being LOW for 10 seconds, or sends a value of `0` if the the signal is LOW for 10 seconds after previously sending a `1`. We believe these measurements will be sufficient to determine whether the space in front of the sensor is occupied by one or more humans at any time.

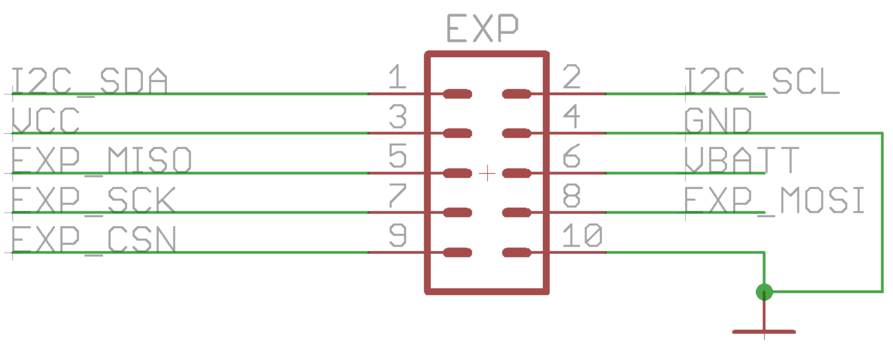


Figure 4.6: Pin configuration of the expansion port of the environmental sensor, allowing adaptation of a wide variety of additional sensors.

Expansion capabilities

The expansion port (see Figure 4.6) is a 10-pin male IDC connector with standard 0.1" spacing. The port exports an SPI Master interface, including clock (SCK), Master-Out-Slave-In (MOSI), Master-In-Slave-Out (MISO), and a single slave Chip Select (CS) line. These four signals can also be used as General-Purpose Input Output (GPIO) pins, including the ability to timestamp changes in voltage. We have also written a software-defined serial driver using these pins which emulates an asynchronous serial port at 9600 baud. The shared I^2C bus is also available on the expansion port, composed of the Serial CLock (SCL) and Serial DAta (SDA) signals. It should be noted that all of these pins operate using 3.3V digital logic levels and are not intended to communicate with 5V logic levels. Finally, the expansion port includes four power-supply pins: two GND pins, VCC, which is the system voltage of 3.3V, and VBATT which is either 3.7V from the lithium primary battery, or can be used as a 3.5V to 6V power supply input when the battery is not being used.

In addition to the expansion port, there are an additional two analog inputs exposed as unpopulated test points which may be used for interfacing devices which have analog outputs.

Firmware design

The firmware of the ATmega microcontroller was programmed in "bare metal" (i.e. no underlying operating system) C code and debugged using the on-board JTAG connector. The code is logically organised into:

- *Device drivers* – code which knows how to configure and extract data from the instruments.

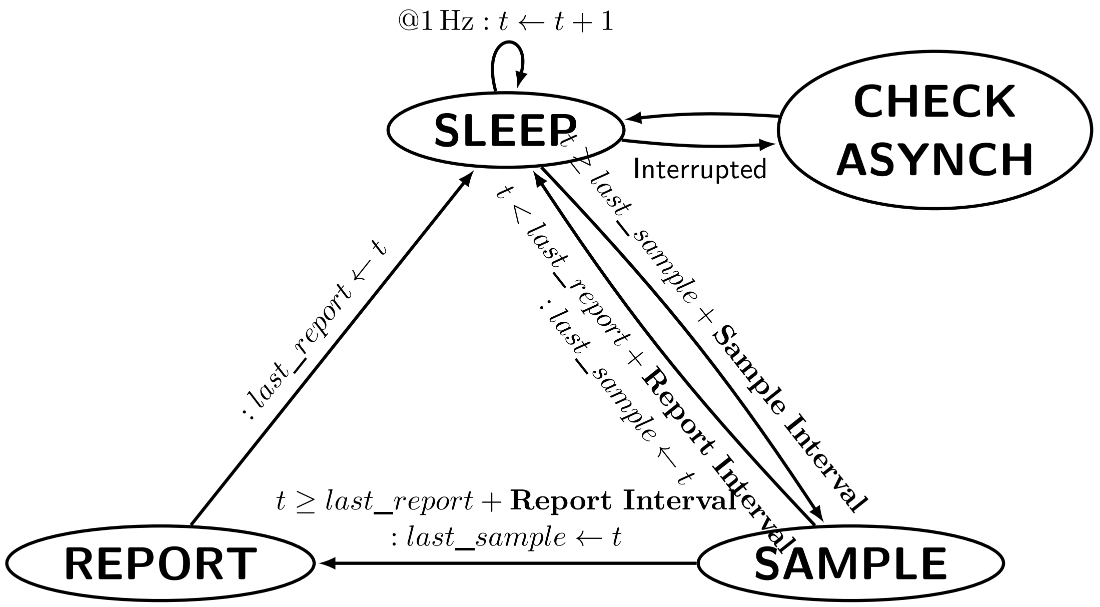

@1 Hz : $t \leftarrow t + 1$
SLEEP
CHECK ASYNCH
Interrupted
$t \geq last\_sample +$ **Sample Interval**
$t < last\_report +$ **Report Interval**
$: last\_sample \leftarrow t$
$: last\_report \leftarrow t$
$t \geq last\_report +$ **Report Interval**
$: last\_sample \leftarrow t$
REPORT
SAMPLE

Figure 4.7: State machine diagram of the firmware running on the ATmega microcontroller of the environmental sensor. The variables *t*,*last_sample*, and *last_report* represent time variables and initialized to 0. **Sample Interval** and **Report Interval** are configuration variables.

- *Peripheral drivers* – code which knows how to configure and use the ATmega hardware peripherals which are shared between multiple modules. Drivers for hardware peripherals which entirely used by one code module are usually included within that module.
- *Radio drivers* – code which knows how to configure and use the attached radio transceiver (such as the RN-XV, or XBee modules). This code also handles the escaping and wrapping routine described by Section 4.4.
- *Utilities* – generic, non sensor-related, utilities such as CRC generation and a task scheduler.
- *Sensor Logic* – Computational code, such as scheduling when samples are taken and reported, constructing reports and following commands specified in Section 4.4, and implementing the recordstore data structure (see Section 4.3).

We illustrate the simplified normal operation of the sensor by Figure 4.7 as a finite state machine with four states:

- The machine begins in the **SLEEP** state, which represents the lowest-power state.

- The machine may be interrupted (e.g. by the PIR sensor) and transition to the **CHECK ASYNCH** state which interprets the interrupt and possibly creates a **Sensor Data Report**.
- The machine may also transition to the **SAMPLE** state if it is time to read the next periodic sample and create a **Sensor Data Report**. A transition to this state triggers the *sampling schedule* to execute.
- Finally, the machine transitions to the **REPORT** state if it is time to transmit the stored samples to the server over the radio.

When the machine transitions to the **SAMPLE** state, the processor must activate the peripheral instruments, instruct them to make a conversion, wait for the conversion to complete, then retrieve the conversion result. To be efficient, while waiting for a conversion to complete for one instrument, the processor can be simultaneously communicating with another instrument. To make this process straightforward, a simple real-time scheduler is provided for the device drivers to use to schedule their operations. Starting from when the **SAMPLE** state is entered, and the sampling scheduler is triggered, the scheduler allows drivers to specify the *earliest* time that a task should be executed (but it could be executed later than specified).

Table 4.3 is the sampling schedule for the instruments on the environmental sensor. The first column gives the time offset from when the sample is triggered. For example, the light and the temperature/humidity instruments require a conversion time, evidenced by the choice of time offset to start and read the conversions. If more than one task is specified to execute at the same time, such as those at the 1 ms time offset, the scheduler executes them in the order they were placed into the queue. Finally, there is a special value of time offset, `LAST`, which instructs the scheduler to execute the task after running all other tasks which are not at time offset `LAST`. Additionally, there are times when no tasks need to be run (e.g. from 67 ms to 106 ms) where the scheduler turns off the CPU to save power (approx. 4 mA of current consumption saved).

Battery lifetime

One of the most important motivations in designing the environmental sensor was to achieve a multi-year battery lifetime in order to reduce the maintenance cost of the network (i.e. labor and supplies needed to frequently replace batteries). Therefore, we utilized several strategies to bring down the average current consumption to a target of less than 200 µA. This included:

- Extreme duty-cycling of the processor and sensors, waking only to take a sample from the instruments or communicate using the radio. A single sample requires that the processor is active for a little over 100 ms, whereas a single radio transmission requires approximately 1 s of radio activity (associate, transmit, then wait for host commands).

Table 4.3: Standard schedule of tasks executed to take one sample

Time (ms)	Component	Description
0	Reporting	Start constructing new **Sensor Data Report**
1	Temp/Humid	Start humidity conversion
1	PIR	Calculate PIR occupancy percentage value and reset
1	Light	Wake light sensor to start conversion
1	Accel.	Read latest acceleration reading
17	Temp/Humid	Read converted humidity & start temperature conversion
67	Temp/Humid	Read converted temperature
106	Light	Read converted ambient light reading
`LAST`*	Reporting	Store **Sensor Data Report** into recordstore memory

* Special signal to the scheduler to *always* run these tasks last.

Therefore, a sample taken every 10 s represents a sampling duty cycle of 1% and radio transmission made every hour represents a communication duty cycle of 1.7%.

- Selection of components with low current requirements. Not only does this reduce the amount of energy consumed, it reduces the peak load needed to be supplied by the linear regulator and battery. Typically, linear regulators with higher peak current capability also have a higher leakage current, and, additionally, high peak current draws can damage or reduce the effective capacity of the battery. The most important component, in this respect, is the choice of radio transceiver. For example, modules such as the XBee-PRO and XBee Series 6 (WiFi) are not used due to their high (over 300 mA) peak consumption.

To determine the feasibility of having a multi-year battery life, we create a power consumption worksheet, shown in Table 4.4, listing the power requirements for each device during their sleep and active modes, and the typical amount of time they are required to be active to perform their functions.

Using the information in Table 4.4, we simulate and plot, in Figure 4.8 the battery lifetime of an environmental sensor node powered by a lithium battery (See “Power supply” in Section 4.5), over varying values of sample intervals (i.e. when a measurement is gathered from the instruments) and report intervals (i.e. when a set of measurements are sent to the server). We simulate current consumption from the light sensor, humidity and temperature sensor, accelerometer, PIR sensor, microcontroller, and radio. We also simulate the reduction of battery capacity at low average current draw. We do not simulate leakage currents or the power consumption of the 3.3V regulator, or atypical conditions which would cause devices to be powered longer than expected, in particular, when communications problems may cause the radio to remain active for up to 5 seconds per report.

We believe that we can achieve a battery lifetime of over 5 years by using a 10 second sample interval and 60 second reporting interval. In this configuration, the average current

Table 4.4: Power consumption worksheet

Device	Sleep power consumption	Awake power consumption	Active time per sample
ATmega1284	0.6 µA	CPU on: 4.5 mA CPU off: 0.5 mA	At least 101 ms
RN-XV*	4 µA	RX: 40 mA TX@0 dBm: 135 mA TX@12 dBm: 240 mA	Association: 15 ms Authentication: 50 ms – 250 ms Application: 500 ms – 5 s
AMN41121	No sleep mode	46 µA	Always-on
HTU21D	20 nA	450 µA	Temperature: 44 ms Humidity: 14 ms
TSL2560	3.2 µA	240 µA	101 ms
LIS3DH**	0.5 µA	6 µA	1 ms

* Not active for every data sample. Typically only active once per hour.
** Typically always-on to do asynchronous orientation change detection.

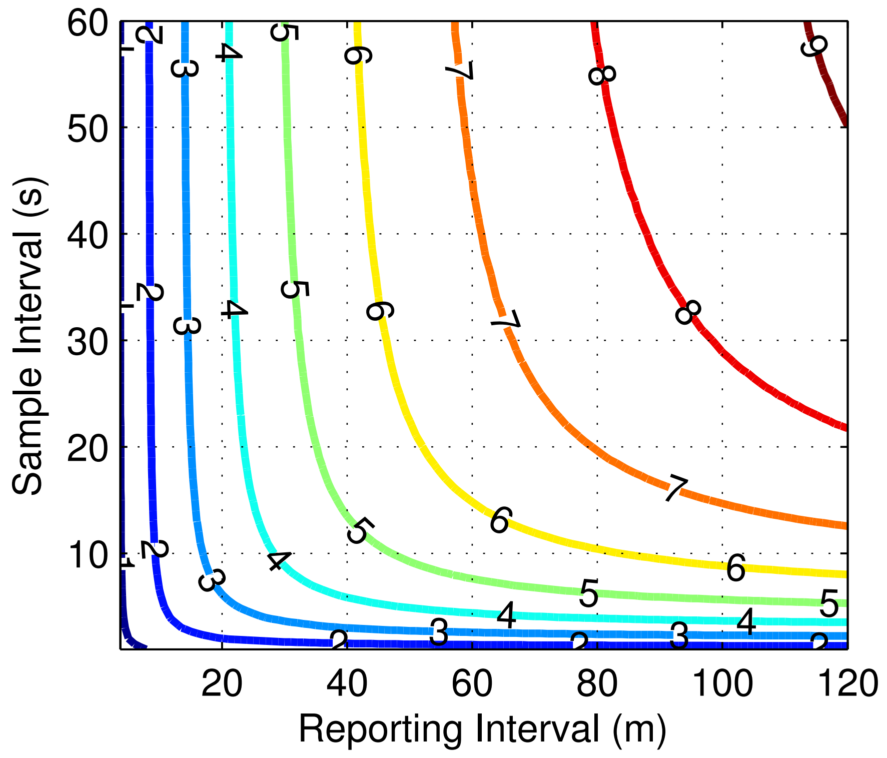


Figure 4.8: Battery Life (surface contours, in years) given by varying the amount of time between samples (y-axis) and time between transmitting the data to the server (x-axis).

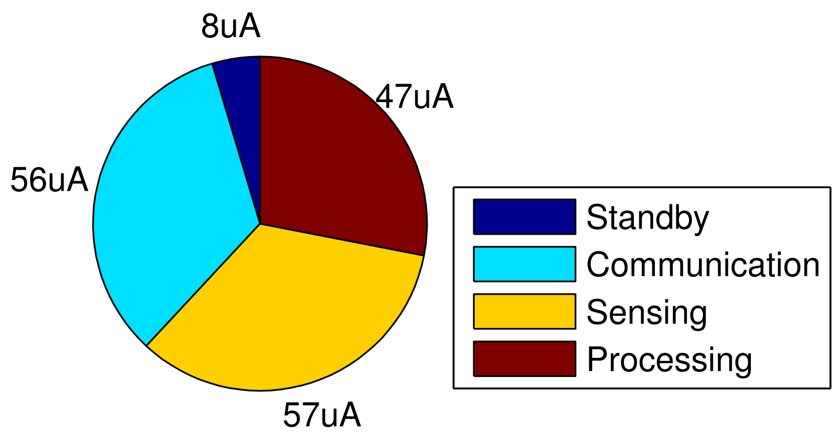


Figure 4.9: Breakdown of average current for target battery lifetime of 5.4 years. Sample interval is 10 seconds and Reporting Interval is 1 minute .

consumption is 168 µA, and the effective battery capacity at this current is 8.03 Ah. Figure 4.9 shows a breakdown how this current is used in this configuration, with relatively equal amounts of current being used for communication (56 µA), sensing (57 µA (46 µA of which is the always-on PIR sensor), and processing (47 µA). A relatively minuscule amount of current (8 µA) is used for inactive devices while they are in their respective sleep modes.

Supported extensions

Although there is a rich suite of instruments already fixed on the main board, we included the 10-pin expansion connector (See "Expansion Capabilities" of Section 4.5) to allow virtually limitless expansion possibility to interface other sensors. Below are examples of external devices that we have interfaced to the environmental sensor board to allow other types of measurements:

CO_2 sensing

Sensing CO_2 has many important uses in intelligent building research, particularly in occupancy estimation [79]. We have successfully interfaced the K-30 CO_2 sensor from CO2Meter [80] to the expansion port of the environmental sensor board (See Figure 4.10), utilizing the shared I^2C bus. The K-30 sensor achieves an accuracy of ± 30 ppm $\pm$ 1% using a self-calibration procedure called *Automatic Baseline Correction (ABC)* which adjust the readings such that the lowest value in the last 7.5 days is equal to 400 ppm (the assumed outdoor air CO_2 concentration).

In our experience with data gathered from multiple K-30 sensors co-located with a lab-grade reference CO_2 meter, we have found that offset errors of up to 100 ppm are common. However, the K-30 sensors were likely not powered long enough for their ABC algorithm to adjust and correct the incorrect baseline. For our experimental work, we have manually corrected the baseline by knowledge of when the environment has been unoccupied for a long time and adjusting the results so that the measurement is 400 ppm at this time.

The K-30 requires a power input of 4.5 V to 9 V at an average current of 40 mA and maximum current of 300 mA, therefore, we require that an external power source (between 4.5 V and 6 V) is applied. We connect the VBATT input of the environmental sensor to the power supply of the K-30 device, so that they share the same power source.

We found that when the K-30 sensor is attached, that it impedes the operation of the HTU21D instrument, causing its measurements to fluctuate (See Figure 4.11). We believe the cause is due electrical noise generated by high peak currents pulled by the K-30 sensor every time it makes a CO_2 measurement (which happens automatically and not under our control).

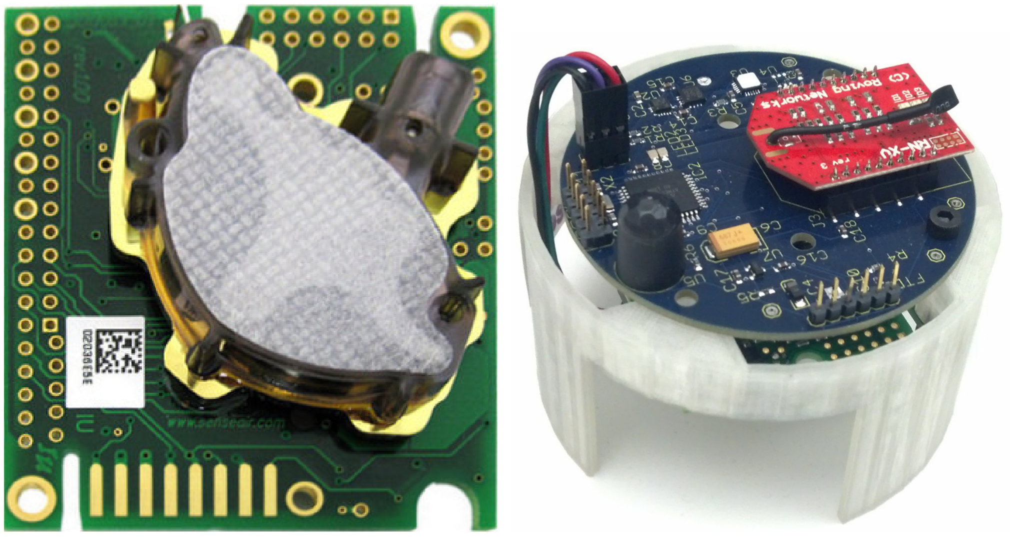


Figure 4.10: Left: The K-30 CO_2 sensor from CO2Meter [80]. Right: Sensor attached to the expansion port of a "Version 2" environmental sensor. This used the software-defined serial driver to read from the sensor, however support for the I^2C interface has been added in "Version 3" of the environmental sensor's expansion connector.

Door opening detection

The door open detector expansion utilizes the GPIO capabilities of the expansion port by attaching a magnetic switch and pull-up resistor. A schematic of the simple circuit is shown by Figure 4.12. When the magnetic switch is closed by the magnet being close by, the signal goes to 0V, whereas when the magnet is far, the magnetic switch is open and the signal is pulled to VCC by the pull-up resistor. The current consumption of the circuit is 10 µA, when the switch is closed, and the leakage current of the ATmega's GPIO input driver (maximum 1 µA), when the switch is open. The interrupt capabilities of the GPIO port allow us to timestamp exactly when the voltage level changes, to within one second.

To install the sensor, the magnetic switch is installed on either a door or its frame, and the magnet is installed on the other side, such that when the door is closed, the magnet comes within 1 cm of the switch, and when the door is open, the magnet is far from the switch. Note that we could similarly install the switch and magnet combination on windows, draws, or cabinets which, given the context, could provide information about where occupants are and which objects in the office are being manipulated.

4.6 Evaluation

Given that a long battery life is one of the primary features of the platform described, it is necessary to validate that the platform actually consumes the theoretical amounts that we calculated from the data sheets. Therefore, we designed a circuit, shown in Figure 4.13

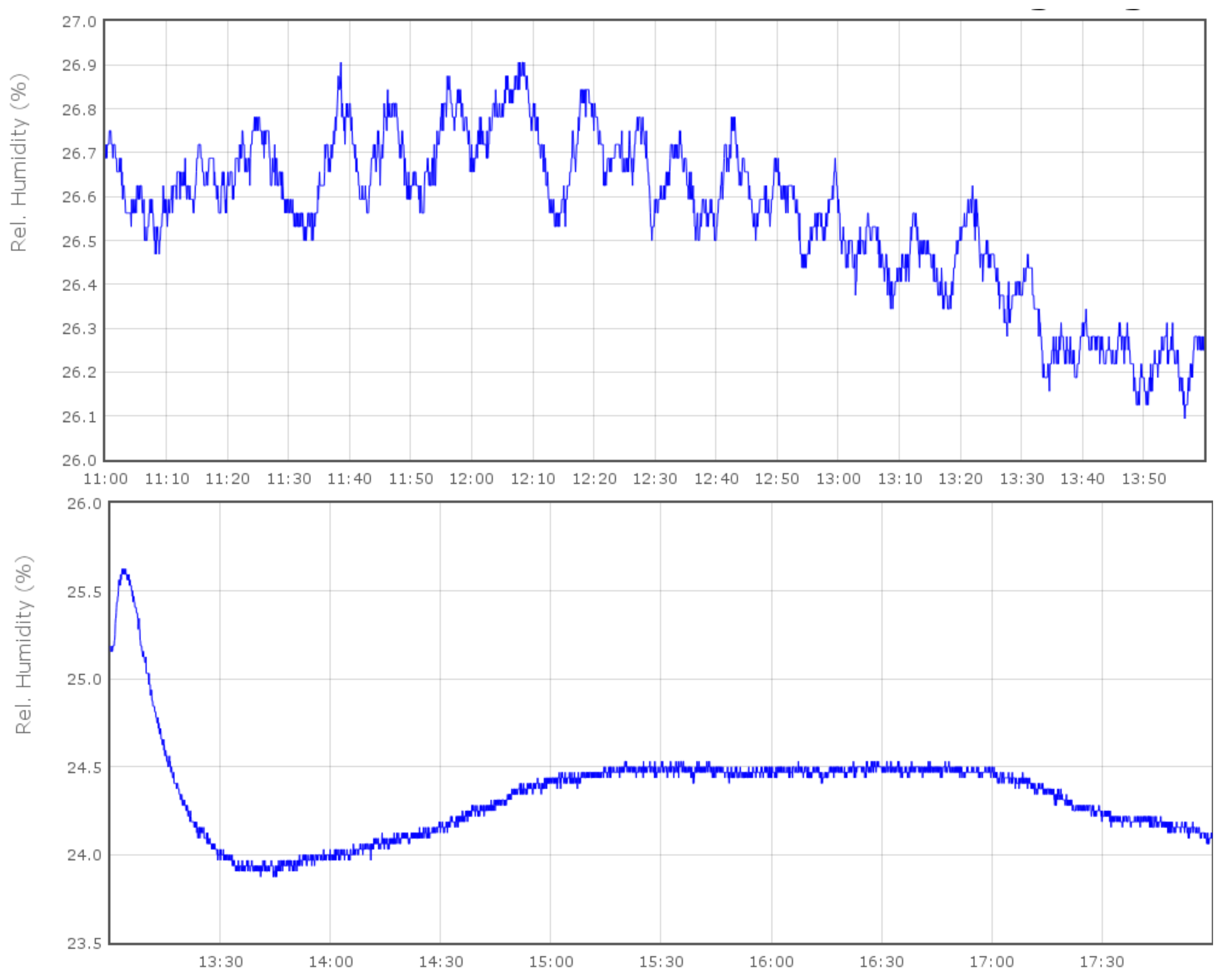


Figure 4.11: Example 2 hour timeseries plots showing effect of CO_2 sensor adding noise to the humidity measurement. The top plot shows the humidity readings from the environmental sensor *with* the K-30 sensor attached and the bottom plot shows the humidity readings *without* the K-30 sensor attached.

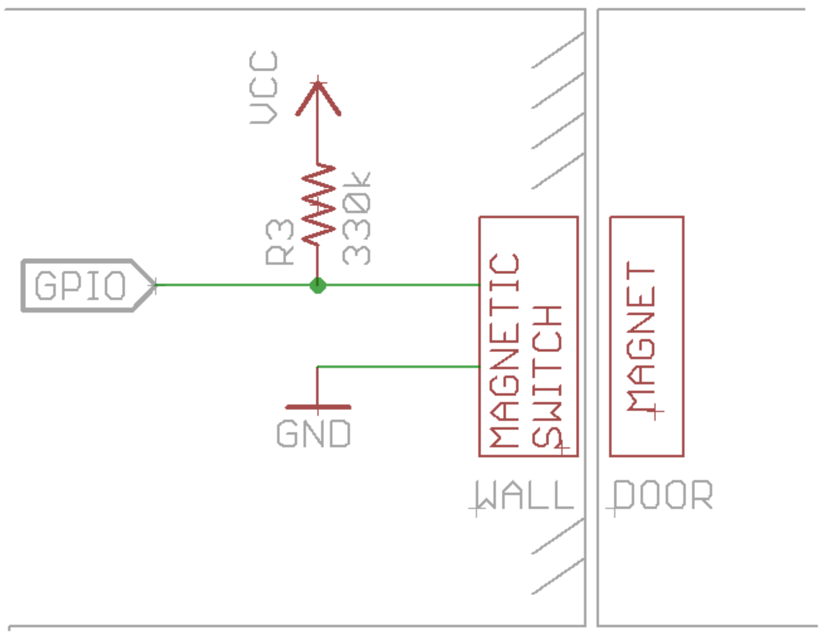


Figure 4.12: Schematic of door opening detection based on a magnetic switch attached to one of the GPIO pins of the expansion port.

to measure the current consumed by our prototype. An operational amplifier produces an regulated 3.75 V output voltage at up to 250 mA (i.e. $(12\,\text{V} - 3.75\,\text{V}) \div 33\,\Omega$). The voltage, v, at the output of the amplifier is captured with an oscilloscope and we can calculate the supply current, i, as

$$i = \frac{v - 3.75\,\text{V}}{33\,\Omega}.$$

The standby current consumption was too low to precisely measure using the tools at our disposal (theoretically, the voltage difference across the 33 Ω resistor would be $< 2\,\text{mV}$ at 60 µA). Since we cannot reliably measure the standby current, for the following plots we plot the current *differential*, that is, the amount of *extra* current that doing a task consumes in addition to the standby current.

We measured the current consumption while the device took one environmental sample and plotted the results in Figure 4.14. There are 5 spikes in current consumption, corresponding to the 5 times when the processor is active, according to the schedule in Table 4.3. We also calculated that the energy capacity drained from the battery for one sample is 1.09 µAh. Note that this is much lower than expected by the numbers previously calculated in Section 4.5, indicating that our previous estimates were conservative.

Figure 4.15 provides similar plots of the current consumption profile while the sensor is transmitting the collected measurements to the server. In this plot, the spikes represent

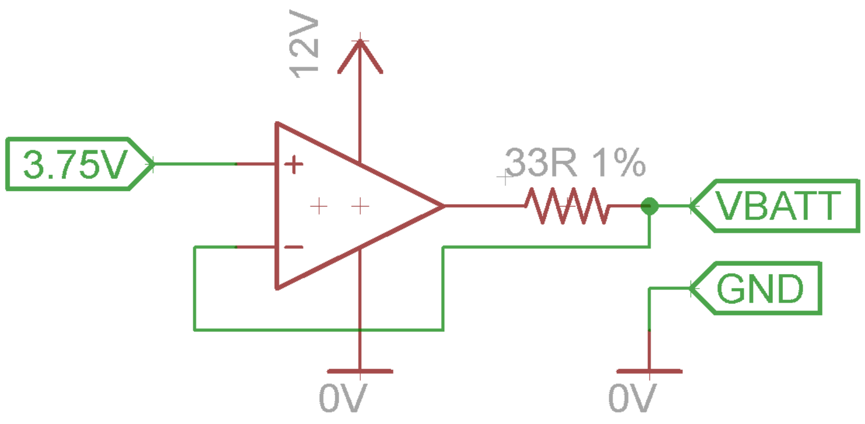


Figure 4.13: Schematic of measurement circuit used to measure currents up to 250 mA consumed by the environmental sensor.

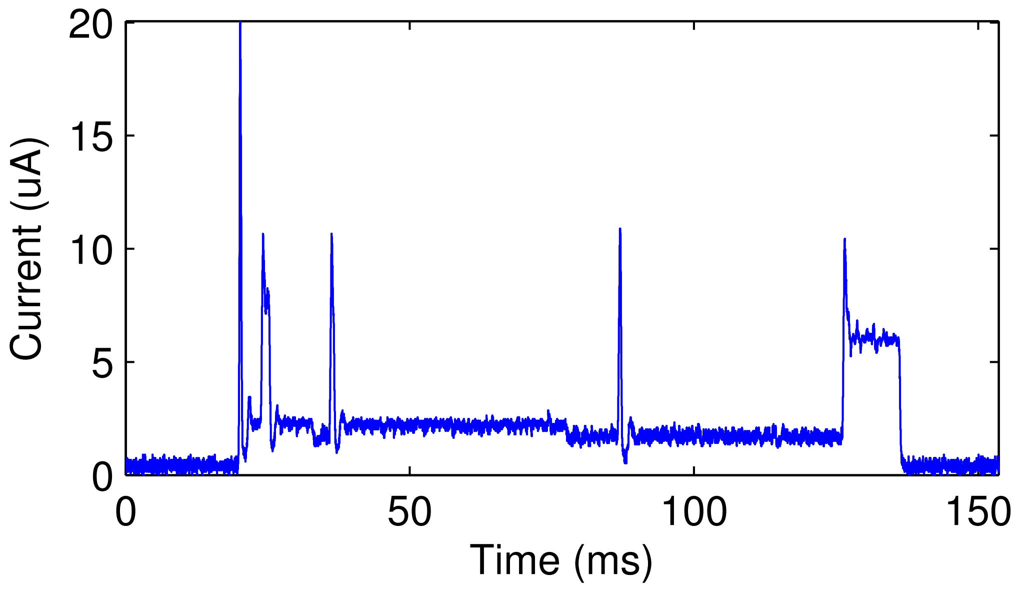


Figure 4.14: Current consumption while one environmental sample is being measured. In this example, 85 pAh was drained from the battery (excluding standby current).

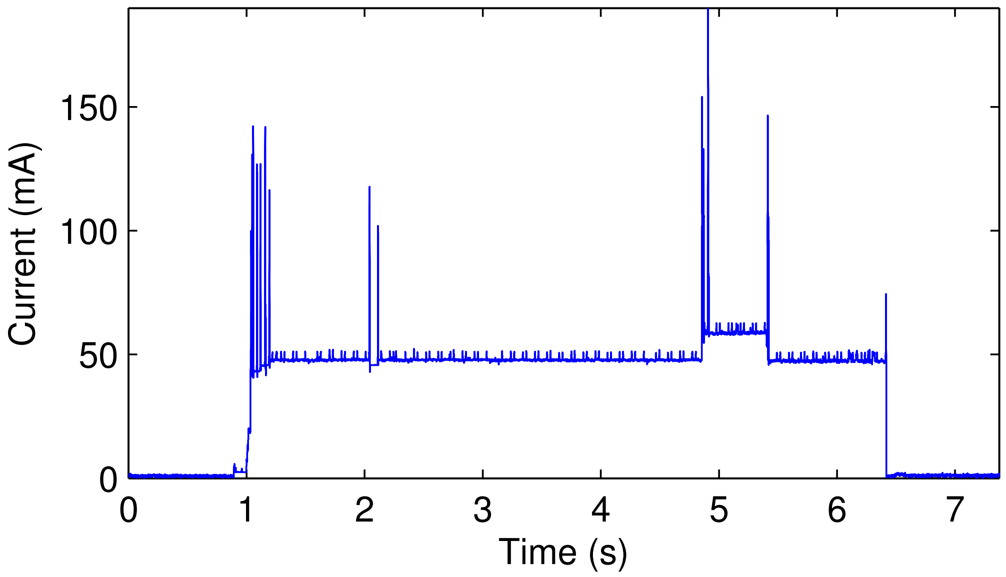


Figure 4.15: Current consumption during transmission of stored measurement data. In this example, 75 µAh was drained from the battery (excluding standby current).

times when the radio is transmitting and consuming up to 240 mA of current. The sensor consumes around 50 mA of total current when the radio is active, but only listening (itself consuming 40 mA).

To estimate the battery lifetime, T, we assume a battery capacity, $C_{\text{batt}} = 8\,\text{Ah}$ (derated from 9 Ah after 5 years of age) and a standby power consumption of 60 µA. We also assume the sensor is configured to take samples every 10 seconds and send a report (approx. 6.9kB each) every hour. We assume that each sample and report consumes the amount of energy as measured above. Therefore, we have the relations:

$$\begin{aligned}
n_{\text{samples}} &= \frac{T}{(10\,\text{s})} \\
C_{\text{samples}} &= n_{\text{samples}} \cdot (85\,\text{pAh}) \\
n_{\text{reports}} &= \frac{T}{(3600\,\text{s})} \\
C_{\text{reports}} &= n_{\text{reports}} \cdot (75\,\mu\text{Ah}) \\
C_{\text{standby}} &= T \cdot (60\,\mu\text{A}) \\
C_{\text{batt}} &= C_{\text{samples}} + C_{\text{reports}} + C_{\text{standby}}
\end{aligned}$$

Solving these equations gives a battery life of approximately 6.7 years, after taking 21 million environmental samples and sending 59 thousand reports. Of the 8 Ah battery, approximately 4.4 Ah (56%) is used for communication, 1.8 mAh (0.02%) for synchronous samples (not including consumption from the always-on PIR or accelerometer instruments), and 3.6 Ah (44%) used for standby consumption, including the PIR and Accelerometer instruments.

Clearly the battery lifetime could be extended by increasing the report interval. Additionally, there does not appear to be a significant power penalty to increasing the sample rate of measuring light, humidity, and temperature. However, we are limited by the memory space of the device, as these samples must be stored until the next report is sent. Therefore, for future designs, increasing the amount of memory, perhaps via an externally connected memory chip, could prove fruitful in improving the battery lifetime. An alternative is to perform some computation on the sensor node to determine whether a sample is useful to be held or discarded. For instance, if the sensor detects that a quantity is rapidly changing, it could choose to store more of the samples, however, if the measurements are not significantly changing, many of the high-rate samples could be discarded.

Secondarily, we could increase the battery lifetime by reducing the standby power consumption of the device, especially by turning off the PIR detector which consumes 46 µA. For instance, we could place two sensors with overlapping PIR detection regions and turn only one PIR detector on at a time, or other sensors could be used to determine whether the PIR detector is needed. For instance, if a door-open detector senses that a door is closed to an empty room, then there is no reason for a PIR detector to be monitoring that room since there is no possibility of humans entering without opening the door.

4.7 Conclusions and Future Work

Ultimately, we achieve our objective of a 5+ year lifetime for the environmental sensor devices and it is comforting to know that there is room for unaccounted factors. Therefore, the environmental sensing framework, as a whole, enables practical deployment of a permanent sensing architecture in the office space. The sensing devices themselves can operate for over 5 years while consuming a lifetime average of 140 µA. Therefore, the costs of deploying and maintaining such a network is significantly reduced in comparison to networks which require wiring or frequent battery replacements.

Although we have achieved our power consumption targets using WiFi technology, we recognize that other standards-based technologies exist, such as IEEE 802.15.4e and WirelessHART, which offer high-reliability, low-latency, and low-power relative to WiFi technology. For example, the SmartMesh line of products from Linear Technology offers roughly the same order of power consumption (less than 100 µA) for communications, but at a latency of less than 10s and achieves 99% reliability (for 100 nodes) [81]. A future goal is to integrate the OpenMote [73] which implements an open-source implementation of IEEE 802.15.4e, OpenWSN [58]). Using the same standards-based protocols, OpenWSN theoretically achieves similar performance to SmartMesh. The primary challenge in incorporating this technology, however, is that it is not well suited to mobile environments, since nodes must undergo a lengthy and costly rejoining process if they frequently leave and enter the radio range of their neighbors [82]. To address this requires careful adjustment of the network parameters and a network topology that assists mobile nodes in staying associated within the mesh network.

From our observations of current, the WiFi module consumes most of its power in idle receive mode, not during transmission of a packet. These times are spent either when the ATmega microcontroller is communicating the data packet to the RN-XV module, or when the sensor is waiting for a command to be sent to it from the Local Server. Therefore, we could significantly reduce power consumption with a faster data connection to the RN-XV and a protocol which does not rely on the module to wait for a possible command from the Local Server. The first change would require selection of a WiFi module which supports the SPI communications bus, which can be much faster than an asynchronous connection. The second change could be accomplished by sending a flag with the TCP acknowledgement packet, indicating whether the server has a command waiting for the sensor.

Another way to reduce the data throughput, and thus the communication and memory requirements, would be to process the data on the ATmega prior to storage. This processing step would be highly dependent on the end application. For example, instead of recording the temperature every 10 seconds, we could record only when the temperature rises or falls below some threshold. However, since we want to avoid constraining the sensors to one application, and because WiFi affords a relatively high data bandwidth, we choose to send the measurements with very little processing.

Using the measurements collected from the on-board instruments, as well peripheral instruments, such as CO_2 and Particulate Matter sensors, we can gather information about the occupancy of building spaces. For instance, in the following chapter, we will describe how measurements of CO_2 concentrations inside of a conference room can be used to accurately estimate the occupancy of that room. We were able to accomplish this non-trivial feat by modelling the CO_2 concentrations within the room and building an estimator for the rate of CO_2 production. To apply the estimation theory, it was necessary to develop the CO_2 sensor platform shown in Figure 4.10.

If the sensors are carried by occupants, this will provide information needed to track the occupant using the particle filtering technique described by Chapter 6. We collect the signal strength from an occupant-carried device and find the most likely position of the occupant by mapping these measurements to previously constructed statistical distributions. However, in addition to signal strength measurements, the framework can be adapted to accept any type of measurement that is affected by the occupants' positions. For instance, the particle filter can glean information from light measurements, such as knowing that the occupant is near or far from a sun-facing window. Further examples are discussed in Section 6.5.

As well as providing the inputs to our occupancy estimation and occupant tracking techniques, the system also proves to provide valuable inputs to other smart building projects. The temperature, humidity, particulate matter, and CO_2 readings are used to control and study the indoor air quality of the space, and the light-level readings are likewise used to determine if an adequate and healthy amount of light is being provided. Since readings are taken frequently and stored permanently, we enable future studies by collecting a rich and long-running data set to be analyzed.

Chapter 5

Models of indoor occupancy

5.1 Introduction

In the previous chapter, we briefly described the advantages of estimating indoor occupancy, especially in regards to significantly reducing energy usage of a building. To detect activity and movement in the building, the most straightforward methods are light-based, such as passive infrared (PIR) sensors, which detect movement of body heat, or emitter-detector pairs which trigger when a light beam is broken. Ultrasound sensors are also popular to determine whether a space is occupied by emitting ultrasonic chirps and measuring differences in the response. Magnetic switches can also be added to doors to determine when they are open or closed. These types of sensors provide accurate detections of occupants, however, the information they provide is limited. For instance, these light-based and ultrasound-based sensors usually have a small detection volume and cannot distinguish the number of occupants or the amount of activity that is occurring [83].

More recently, there has been a host of indoor positioning systems (IPS) [84, 85, 86, 87] developed to obtain a person's location (i.e. x-y coordinates) continuously. For instance, an IPS is used in Chapter 6. From the positions of individual occupants in the building, a distribution can be constructed by counting the number of occupants in a given area. However, a wireless IPS infrastructure requires that the occupants carry radio tags, such as custom RFID tags. It is difficult to enforce that occupants carry exactly one tag with them at all times, such as when visitors are present. This is somewhat alleviated by WiFi-based IPSs, which sniff the transmissions of occupants' cell phones without any prior association. However, the method still requires that the occupants carry exactly one WiFi device, that the device has WiFi enabled at all times, and that the device consistently transmits over the WiFi channel. Therefore, IPSs have these challenges which can prevent them from accurately tallying the true number of occupants in a space.

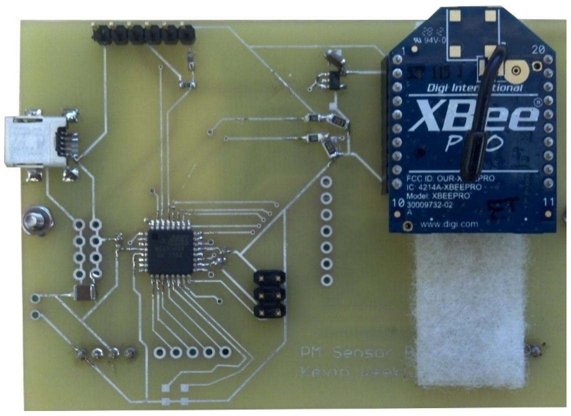
To explore techniques which do not have these limitations, we directly estimate the occupancy level of indoor spaces. This has several advantages over counting the number occupants in a space from the results of an IPS: The first is that the focus of many IPSs is to obtain the highest positioning accuracy (i.e. distance between true and estimated position), irrespective of the logical boundaries which divide the building into spaces. Therefore, even an accurate IPS could result in poor occupancy estimation, such as if occupants are positioned on the other side of a wall from their true position. The limitation demonstrated by this particular example is addressed in Chapter 6 with an IPS that integrates obstacle information into the estimation. A second advantage of directly estimating occupancy level is that other types of sensing technologies are available to add information into occupancy estimates. These technologies measure environmental variables, such as CO_2 and Particulate Matter (PM) concentrations which are correlated to occupant activity, and can thus be used as a proxy. Moreover, heating, ventilation, and air conditioning (HVAC) accounts for 48% of the total energy usage in US office buildings [53]. Therefore, it is natural to investigate airborne phenomena as a proxy for occupancy, since we would like to control the same air spaces as we are estimating the occupancy of (e.g. we would like to estimate whether a room is occupied to control the HVAC services for that room—both the control and the estimation problem operate on the same air space). Finally, as a practical advantage, direct occupancy estimation techniques often require no occupant-carried sensors.

We also specifically choose to investigate indoor CO_2 and PM concentrations due to their importance for monitoring Indoor Air Quality (IAQ) which must be maintained for the health and productivity of the occupants. In fact, scientists have termed Sick Building Syndrome (SBS) [88] as symptoms, such as headache, fatigue, and rash, which are brought on by low-level exposure to non-industrial indoor airborne contaminants. Diluting the concentration of these contaminants to a safe level has a direct energy cost, since fresh air must be brought in from the outside, filtered, and conditioned. Referred to as ventilation, this process is nevertheless very important as ventilation rate per person is strongly correlated to the occupants' productivity [89, 90, 91]. In current buildings, the ventilation rate is often insufficient since regulations only specify minimum rates for the ventilation system design, but do not specify rates during operation [91]. To properly satisfy the IAQ requirements, the ventilation rate should be increased to reduce contaminants while the building is occupied, and to save energy, decreased while the building is unoccupied. Therefore, we need to monitor both the occupancy, and the pollutant concentrations of spaces within the office. Our goal is to estimate the occupancy using the same sensing infrastructure that is needed to monitor pollutants, which will further add value to deploying these important sensors.

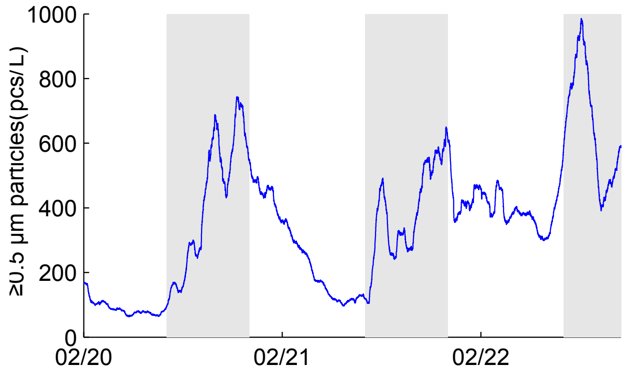
5.2 Correlation of occupant activity to coarse particulate matter

Indoor particulate matter sensing

As previously discussed, acceptable IAQ is an occupant need and should be sensed and controlled for. The need for PM sensors in consumer devices which control air quality, such as air purifiers, is an example where manufacturing advances have made the sensors much less



(a) The wireless PM sensor, mounted on the back of the PPD-20V, which used for PM measurements for particles $\geq 0.5\,\mu m$.



(b) PM detected inside office on UC Berkeley campus. Shaded area represents times between 10:00 PST and 20:00 PST each day.

Figure 5.1: Real-time PM sensing infrastructure sensor (a) and example data (b).

expensive than laboratory equipment. PM sensors measure the particulate concentration, which, at high levels, correlates to health problems [92]. Beyond their original intended use for IAQ, they can also be used for indoor occupancy estimation by observing the effect of occupants on local PM concentrations. While conducting continuous monitoring of PM concentration for IAQ, we noticed trends intuitively correlating to occupant activity. For example, Figure 5.1b shows a time-series of PM detected in an office space at the UC Berkeley campus over two full 24-hour periods. Clearly, there is a trend of a higher concentration of particles during the working hours of the day, and we can even see when there is a surge of activity inside of the workday, as indicated by the rise in PM. This trend is also consistent with previous work that showed significant increases of PM concentrations due to occupant activity such as walking [93].

In this section, we use a low-cost ($<$ 8 USD) PM sensor to infer the local movement of occupants in a corridor by sensing the resuspension of coarse ($\geq 2.5\,\mu m$) particles. To obtain meaningful values from the inexpensive sensors, we have calibrated them against a laboratory-grade instrument. After calibration, we conducted a 7.8 hour experiment measuring coarse PM within a pedestrian corridor of a heavily-used office area. Comparing against ground truth data obtained by a camera, we show that the PM sensor readings are correlated with occupant activity, thus enabling statistical methods to infer one from the other.

Our hypothesis is that PM concentration is an additional method that can be used to detect occupancy. In the present study, we focus on the effect within a corridor, allowing us to easily obtain a ground-truth via camera. However, motivated by the results plotted in Figure 5.1b, we believe that occupancy and activity for an entire room could be estimated.

Procedure

Real-time monitoring

To enable long-term experiments, we designed and built a wireless-enabled PM sensor based on the PPD-20V sensor, shown in Figure 5.1a. The circuit board has the same dimensions as the PPD-20V sensor and is attached to the back by two bolts and two sets of header pins. An on-board ATmega128 microcontroller reads the digital output of the PM sensor and calculates the PM sensor ratio over 10 minutes. Every 10 minutes, the microcontroller uses an 802.15.4 transceiver (Digi XBee series) to send the reading to a sink node containing an embedded Linux computer (Beaglebone) and its own 802.15.4 transceiver. Python software communicates with the transceiver and forwards the raw sensor packet data to our Internet server for interpretation. Software running on our Internet server receives and interprets the data packet, then inserts the data into an sMAP [60] server instance running on itself. Thus, software for our future research projects can retrieve the data using the standardized sMAP protocol. Figure 5.1b shows one example of data that was collected with the device in Figure 5.1a and retrieved using sMAP. We can also use the sMAP web interface to quickly visualize the data as it arrives. Given the prohibitive expense of the PPD-20V sensor, our plan is to adapt the design to accommodate the DSM501A sensor and deploy many of these in the office for future coarse PM studies.

Particulate matter sensing

Our experiments consisted of measuring particle concentration using 8 PM sensor modules. The specifications of the two models tested are given in Table 5.1. Figure 5.2 shows the principle of operation of the PPD-20V sensor and the DSM501A sensors operate using a similar principle. Essentially, the sensor employs a Light-Emitting Diode (LED) aimed at a small point inside the device. When a particle of sufficient size passes by this point, a detector picks up the scattering of the LED light and outputs a digital signal on the output pin. A feature of the DSM510A device is that it provides two outputs with different sensitivities. A resistive heater causes convective airflow which ensures a small flow of air is passed through the device. We have programmed an ATmega168 microcontroller to receive the 13 digital signals from the PPD-20V and DSM501A sensors and record the data onto a Secure Digital (SD) memory card. It is programmed to sample every signal at 2.5 kHz to see if it is 0 V (logic 0) or 5 V (logic 1). Every 50 ms, it logs the number of samples that were logic 1 during the last period. This data set can be processed into a ratio of the time that the signal is 0 V within a time window. After calibration against an accurate sensor, the GT-526S laser particle counter, this ratio value can be mapped to meaningful units (described in Section 5.2).

Table 5.1: Sensor models tested.

Model	Qty. Tested	Cost	PM Size detected
DSM501A[94]	5	< 8 USD	Two outputs: $\geq \{1, 2.5\}$ µm
PPD-20V[95]	3	~ 700 USD	≥ 0.5 µm
GT-526S[96]	1 (reference)	2990 USD	Six outputs: $\geq \{0.3, 0.5, 1, 2, 5, 10\}$ µm

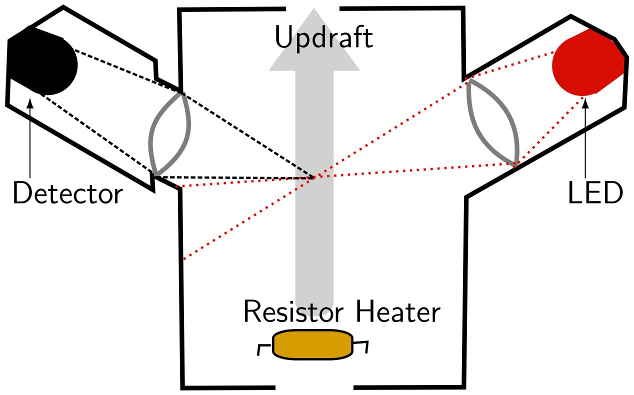


Figure 5.2: Operation of a low-cost PM sensor measuring scattering of LED light (Shown: PPD-20V).

Physical configuration

The experiment was carried out in the main corridor of 490 Cory Hall (also known as the SWARM lab) on Berkeley campus, which is a heavily used and trafficked office location. Figure 5.3 gives a picture of the corridor prior to the experiment being run.

The camera and PM Sensors were mounted on an aluminum structure leaned against one side of the corridor and visual landmarks were placed on the other side of the corridor. The sensors and landmarks were arranged according to Figure 5.4. Figure 5.5 is a picture of the apparatus we constructed to achieve this configuration. Although the analysis in this article does not look into differences in PM readings due to spatial variations, the apparatus allows us to investigate this later. The most valuable contribution of the apparatus was allowing us to run a multiplicity of sensors with all of them relatively close and in the same orientation, and we found that averaging the readings from several sensors gave the smoothest results.

Ground truth sensing

The goal of the ground truth observation was to determine when occupants or other objects passed in proximity of the PM sensor. Ground truth data was collected using a Veho VCC-003-MUVI digital video recorder. After replacing the stock memory card with a 16 GB card, and the battery with a much larger one, the camcorder was capable of recording approximately 10 hours of video at a resolution of 640×480 pixels. We placed visual landmarks

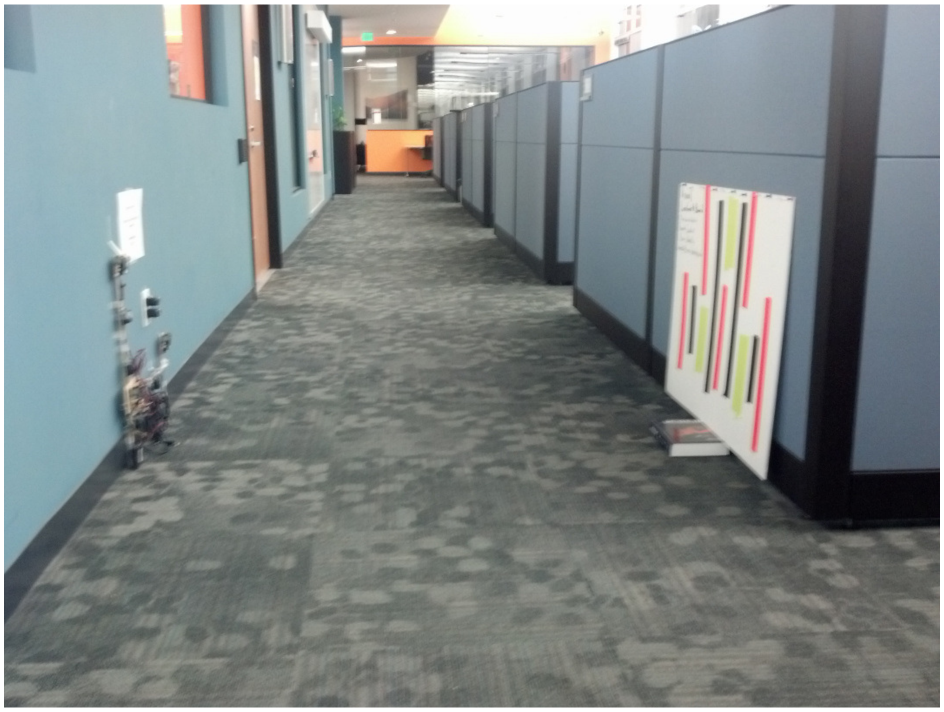


Figure 5.3: Main corridor of 490 Cory Hall, prior to experiment being started. Rooms on the left are lab areas and cubicles on the right are occupied work spaces.

(bright orange tape strips) in the field-of-view of the camera, to make post-processing more straightforward. Examples of the images captured are shown in Figure 5.6.

To obtain the ground truth, the video was later post-processed to detect when something obscured the visual landmarks. We made the determination based on whether the hue, saturation, and value of the color at the landmark locations fall outside a specified range. Since the video is also archived, we can validate the computer vision manually.

Results

Filtering

Data recorded by the apparatus was taken at a high sample rate relative to the effects we intended to measure. By inspection, the individual PM samples are not easy to interpret as they are mostly either a ratio of 1 or 0. By doing several filtering steps we are able to extract a meaningful signal.

The raw data of the experiment comes in the form of a text file from the PM sensor apparatus as well as a video file from the digital video recorder. The video file was processed by a simple computer vision algorithm to give a set of times, $\mathcal{C} = \{t_1, t_2, \ldots, t_n\}$, when occupant activity occurred in front of the camera. There were a total of 312 such occurrences

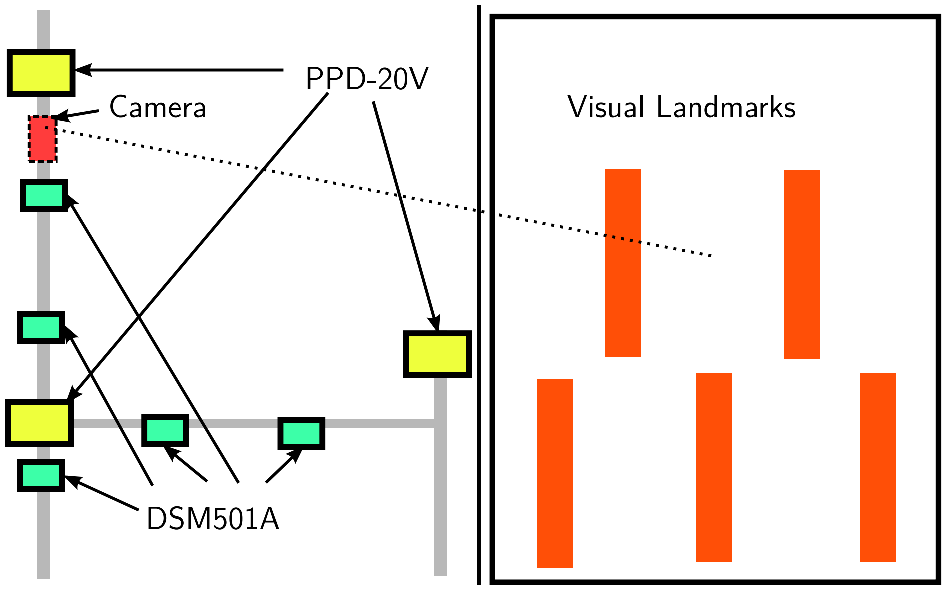


Figure 5.4: Physical Configuration of the sensors and visual landmarks. Dotted line indicates center of view of camera.

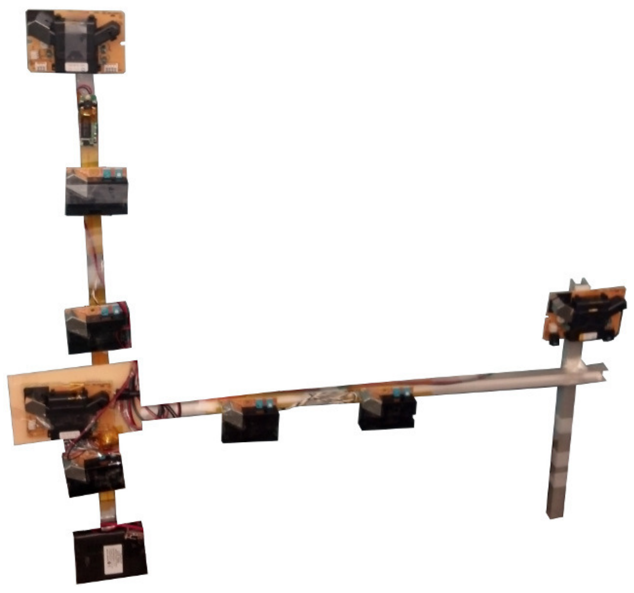
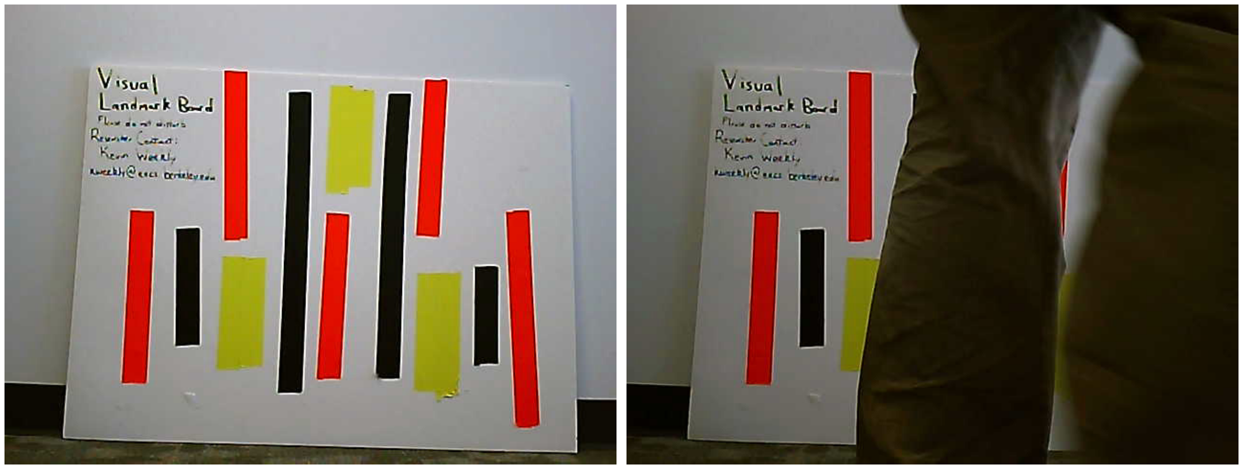


Figure 5.5: Physical implementation of the design in Figure 5.4.



(a) Clear View (b) Obscured View

Figure 5.6: Example images showing detection of occupant in corridor from obscured visual landmarks (orange strips).

during the experiment. We calculated a *camera occurrence rate*, x_i, by the following:

$$x_i = \frac{|\{t | t \in \mathcal{C} \wedge t \geq W_s(i - 0.5) \wedge t < W_s(i + 0.5)\}|}{W_s}$$

where W_s is the "sample period" and is set to 10 s for these results. Thus, x_i is a discrete signal representing the number of camera occurrences per second, evaluated every 10 s.

The PM sensor signals, originally at a sample period of 50 ms, are down-sampled to a sample period of W_s (10 s), by averaging every 200 readings.

We then passed each signal, both PM readings and camera occurrence rate, through a sliding window average filter with a window size of 30 samples. Thus, the resulting samples still have a sample period of 10 s, but each sample represents the average of 300 s, or 5 min, worth of data.

Selection of variables

The text file provides timeseries of 13 variables, 2 per DSM501A sensor and 1 per PPM-20V sensor, so we sought to find which signals were relevant. We found that the coarse outputs correlated most with camera occurrences, an effect which is also supported by [97], which found that occupancy is a large factor in supermicron particle concentration, but an insignificant factor for finer particles. Intuitively, this is because larger particles are overcome by gravity and settle faster than finer particles which are kept afloat by air currents. Sensors which are sensitive to smaller particles are actually less effective for inferring occupancy since their readings will ultimately be dominated by submicron particles (which have a concentration almost 10 times as large as supermicron particles). To verify, we computed the Pearson's correlation coefficient between the camera data and PM readings, grouped

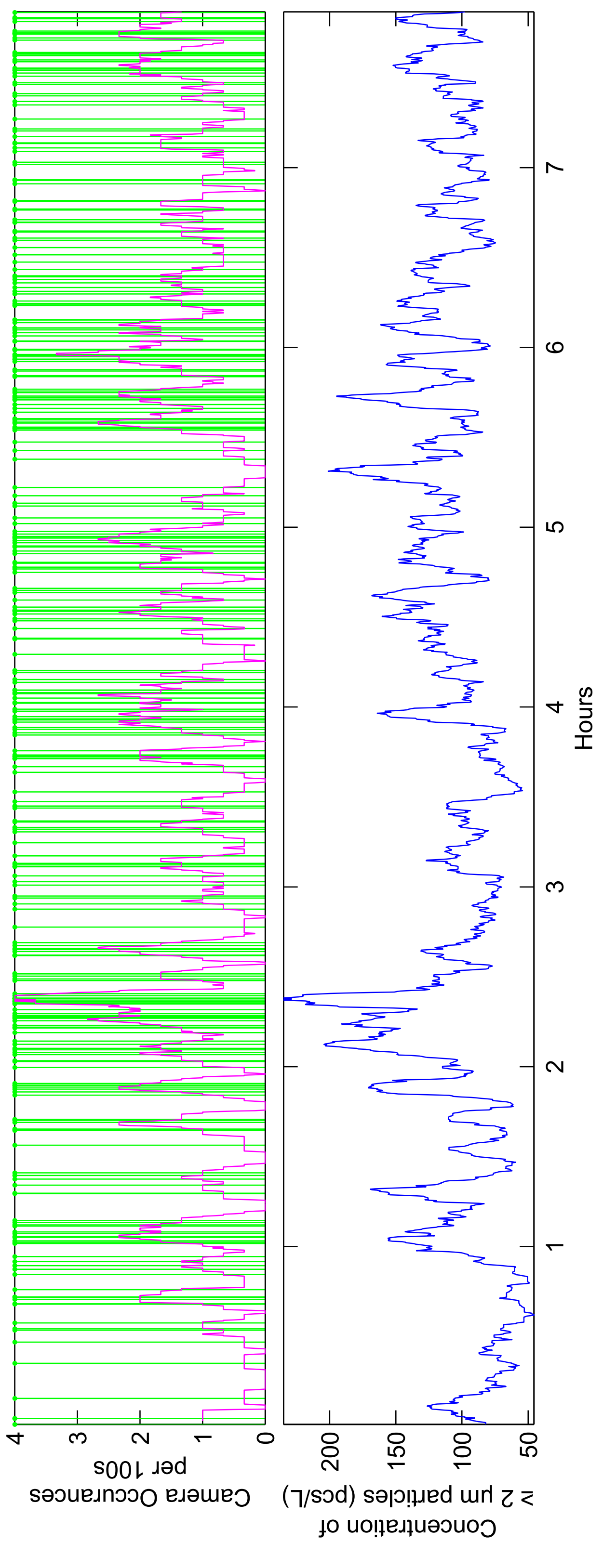


Figure 5.7: Timeseries of filtered data over 7.8 hr experiment. Top: Green lines mark camera obstruction occurrences and magenta line is the filtered camera occurrence rate. Bottom: Filtered $\geq 2.5\,\mu$m outputs from DSM501A (average of 5).

Table 5.2: Pearson's Correlation Coefficient for different outputs.

Output	r
$\geq 0.5\,\mu\text{m}$ from PPD-20V	-0.03
$\geq 1\,\mu\text{m}$ from DSM501A	0.28
$\geq 2.5\,\mu\text{m}$ from DSM501A	0.49

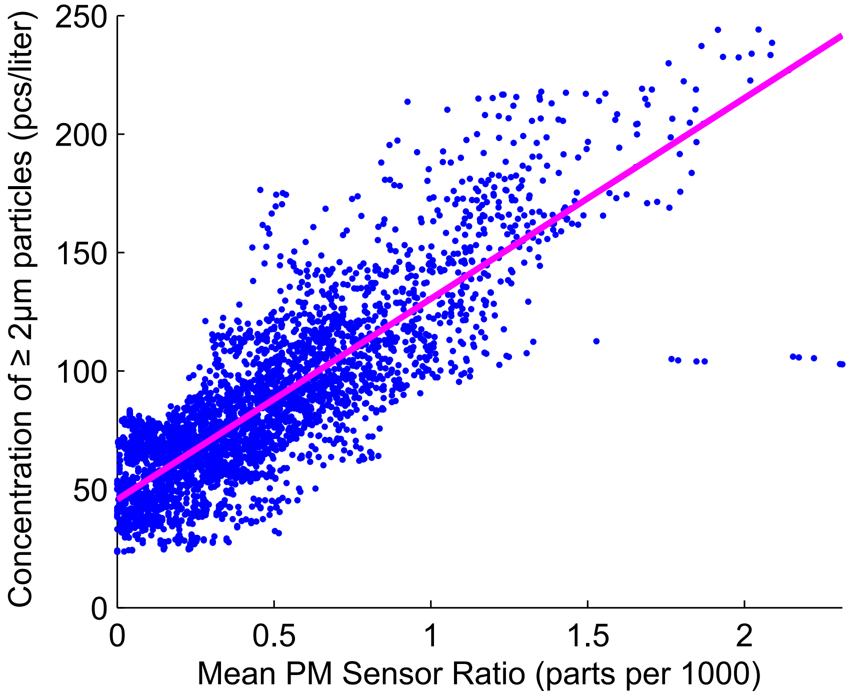


Figure 5.8: Scatter plot of $\geq 2\,\mu\text{m}$ particle concentration from OPC against $\geq 2.5\,\mu\text{m}$ output from the DSM501A.

by the size of particles sensed. The results, presented in Table 5.2, show that the amount of linear correlation between camera occurrences and PM readings is the most when the particles are 2.5 µm or larger.

Ultimately, we decided to construct a signal, y_i, composed of the average of all 5 DSM501A $\geq 2.5\,\mu\text{m}$ outputs. That is,

$$y_i = \frac{1}{5}\left(\sum_{k=1}^{5} \mathrm{PM}_i^{k,2.5\,\mu\text{m}}\right)$$

where $PM_i^{k,2.5\,\mu\text{m}}$ is the ith reading of the $\geq 2.5\,\mu\text{m}$ output from the kth DSM501A sensor. This signal, by inspection, was smoother than any individual PM output and showed more correlation with the camera data.

Table 5.3: MSE of linearly calibrated outputs compared to OPC

Output	MSE
$\geq 0.5\,\mu\text{m}$ from PPD-20V	4.7%
$\geq 1\,\mu\text{m}$ from DSM501A	19.4%
$\geq 2.5\,\mu\text{m}$ from DSM501A	9.5%

Calibration

The calibration procedure consisted of a 29.5 hour experiment where data was collected by the test apparatus while the OPC measured accurate values of PM concentration. For both the $\geq 0.5\,\mu\text{m}$ and $\geq 1\,\mu\text{m}$ outputs, we used the corresponding channel from the OPC to compare against. For the $\geq 2.5\,\mu\text{m}$ output of the DSM501A sensor, we used the $\geq 2\,\mu\text{m}$ channel of the OPC to compare against.

For each output we performed a linear fit against the reference data. The scatter plot in Figure 5.8 illustrates the data collected and the fitted line for the $\geq 2\,\mu\text{m}$ output. To determine the accuracy of the linearly fitted model, we also calculated the relative Mean Squared Error (MSE) of the modelled concentration against the real concentration, shown in Table 5.3. Examining the time series, we found that there was a spike of airborne PM at around 00:00 hours during the experiment. The spike could be seen mostly in the micron or lower sized particles and was picked up by the PPD-20V unit. However, the DSM501A did not pick up the spike in PM, perhaps because some characteristic of the particles such as surface (changing the refraction characteristics) or composition (causing them to not be caught in the heater updraft) caused them to not be detected by the DSM501A. This complicated the calibration results significantly as can be seen by the $\geq 1\,\mu\text{m}$ results of Table 5.3.

Fortunately, as described in the previous section, we opted to use the $\geq 2\,\mu\text{m}$ outputs of the DSM501A for correlation with activity. This output could be calibrated reasonably accurately and thus we used the following linear model to map the PM sensor raw units to real-world values:

$$\text{PM}_{\geq 2\,\mu\text{m}} = \left(8.48 \times 10^4\right)(\text{PM Sensor Ratio}) + 45.7$$

Synchronization of PM sensor and camera data

We also shifted the PM sensor data earlier in time so that it more closely correlates to the camera data. We calculate the shift amount as the peak location of the cross-correlation of the two signals. For this experiment, the shift amount was found to be 30 s. Physically, this represents the time it takes for activities happening in front of the camera to affect the sensors via air circulation or diffusion. It could also account for experimental errors in synchronizing the time offset of the two signals.

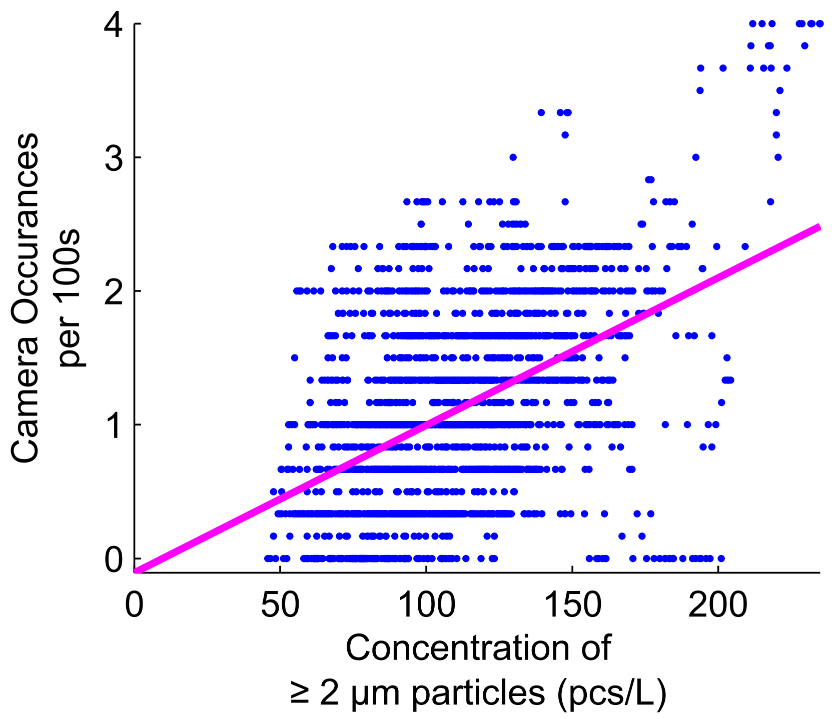


Figure 5.9: Scatter plot of the data shown in Figure 5.7. Magenta line is the linear fit to the data.

490 Cory corridor experimental results

Figure 5.7 is a plot of the two signals x (Top) and y (Bottom) after the filtering steps described above. The green lines are plotted at the times in the set $\mathcal{C}$. Visually, we can see a clear correlation of the two signals, particularly noting that many of the sharp increases in PM are correlated with increase in camera occurrences. There are some cases (e.g. at 5.5 hours) where there is a spike in PM concentration, but no corresponding spike in camera occurrences. One explanation is that there were significant resuspension events in the nearby cubicle without any persons moving in front of the camera.

The same data, is presented in a scatter plot in Figure 5.9 and we also plot the linear fit to the data. The linear fit is given by: $y_i = (2.6 \times 10^{-2})\, x_i + 4.6 \times 10^{-4}$.

This data could be used to construct the statistical models: $\Pr(x_i, y_i)$ (i.e. joint probability), $\Pr(x_i|y_i)$ (e.g. for maximum likelihood detection), or $\Pr(y_i|x_i)$ (e.g. as a sensor noise model). In particular, the noise model, $\Pr(y_i|x_i)$, is an input to Baysian estimation algorithms such as the particle filter [98, 99].

An example which demonstrates using the correlated data is a simple binary detector. Consider a detector which outputs true if the camera occurrence rate is less than once per 100 s, using the information of whether the $\geq 2\,\mu\text{m}$ concentration is less than 100 per liter.

Used on this data set, the detector would be correct 66% of the time with a 42% false positive ratio and 28% false negative ratio. While this alone is far from ideal, it could be combined in a multiple agent decision framework containing other sensors such as passive infrared detectors.

Future work and connections

We have shown via experimentation that local occupant activity, measured visually by a camera, is correlated to the concentration of coarse particles, particularly those $\geq 2.5\,\mu\text{m}$. These types of particles can be easily sensed with low-cost PM sensors such as the DSM501A. Furthermore, if smoother data based on an average over numerous measurements is needed, more low-cost sensors could be added, while still being economical. We have also described the hardware developments which enable real-time reporting of PM sensor data to a central server using wireless technology. These hardware modules, while primarily intended for air quality monitoring, can also indicate occupant activity as we have shown by this article. We believe it will prove to be yet another piece of valuable information for estimating the occupancy of smart buildings.

An important next step will be to investigate the causality of the variables we are measuring. Of particular concern is describing the hidden variable which is measured by both the camera and PM sensor. That is, we have not determined whether how dependent the PM measurement is on the occupancy of the entire room versus the local occupancy directly in front of the camera. An experiment which could disambiguate the two would be to also place several PM sensors away from the test apparatus and measure the PM level of the entire room. For a permanent deployment, multiple sensors in the same room will be useful for estimating this background PM. We would then look at deviations of the local PM from the background level.

Further study into using coarse PM readings as a means of estimating occupancy directly is motivated by these results which show that there is a correlation between the two variables. Our method would be to fuse the partial information provided by PM readings with the information gathered from other sources, such as CO_2 concentration measurements. There is extensive literature[100] on information fusion techniques which can take redundant or complementary information and fuse them into a better estimate than either information source would provide on their own. For instance, in the next section, we discuss a method which provides information about occupancy using CO_2 measurements from the supply and return air vents of a conference room. In Section 6.5, a unified particle filter is introduced which can incorporate all sources of information into an occupant tracking and occupancy estimation framework.

5.3 Effect of occupant on room-level CO_2 concentrations

Indoor CO_2 monitoring

Another metric for measuring indoor air quality (IAQ) is the airborne concentration of CO_2. For example, a study was conducted by assessing the performance of 22 participants while controlling the environments' CO_2 concentration [101]. In this study, moderate decrements in decision-making performance were observed for CO_2 levels of 600 and 1000ppm, and large decrements for CO_2 levels of 2500 ppm. Another study also found the risk of developing sick building syndrome (SBS) is significantly decreased when CO_2 concentrations are below 800 ppm [91]. Therefore, there is a compelling argument to monitor and reduce indoor CO_2 concentrations in order to maximise the productivity of the occupants. However, there is also pressure to reduce ventilation (the primary means of removing CO_2) to save energy. Our objective is to estimate occupancy from the measured indoor concentrations of a conference room, so that building services including ventilation, have this information when making control decisions.

Although we have discussed other phenomena which can be a proxy for occupancy, using CO_2 concentrations has a particular advantage in that human breath is the primary source of CO_2 in the building due to humans' metabolic activity [91]. Human breath contains a concentration of over 1.7% CO_2, depending on the metabolic rate of the occupant and the amount of time a breath is held before exhaling [102]. For comparison, outdoor air CO_2 concentration contains around 400 ppm [103]. In our CO_2 monitoring studies, we have mainly observed CO_2 concentrations from 400–500ppm, although concentrations above 1000 ppm are observed when a room is highly-occupied [79].

Modeling CO_2 dynamics is challenging, due to the complexity of air dynamics. Most recently, two categories of models are used: Zonal models and *Computational Fluid Dynamics* (CFD) models. CFD models provide the most rich and detailed view of air motion in a space, however, they are beset by arduous work in modeling the physical space (e.g. providing locations of all walls, furniture, and occupants) and identifying all parameters that are needed for the model. CFD models also suffer from lengthy computation times to solve the necessary PDEs at a high resolution, especially near boundaries [104], [105]. Zonal models relate the movement of air between discrete and well-mixed spaces, such as rooms and parts of rooms. Generally, zonal models rely on ODE mass-balance laws between these spaces, which, in comparison to CFD models, can be solved very quickly [104]. However, this comes at the expense of not modeling the distributed nature of airborne contaminant transfer within a single space, and complex local phenomena such as jets of air coming from a vent [106].

Yet, for designing and implementing estimation algorithms for the CO_2 concentration, one has to develop a simple, and at the same time, accurate PDE-based model that retains the distributed character of the system. Based on this model, one can then design an observer for estimating the unknown CO_2 input that is produced from humans. The observer design has to be developed using the minimum number of sensors, in order to reduce cost and increase reliability. It is also crucial to develop online identifiers for the parameters of the model, since these parameters change with time due to their dependency on time-varying quantities such as heat generation [107].

Tuning and verifying the applicability of the model requires experimental data to be collected. We conduct two experiments. In the first, a regulated amount of CO_2 gas is released in the conference room for specific time periods while CO_2 concentrations are measured at eight different locations in the room. We use the measurements of the CO_2 concentration in order to develop a model that reproduces the measured CO_2 concentrations at the eight different locations, given the known CO_2 release. We also use the measurements from this experiment in order to identify in which sensing locations the measured CO_2 concentrations are more sensitive to an external CO_2 source. In the second experiment, we monitor the evolution of the CO_2 concentrations at three different locations in the room (the ventilation system's input and output, and at a table located in the conference room) as two researchers enter and exit the room at recorded times. The purpose is to verify the model that we develop in the first experiment under a CO_2 input that is generated by humans.

By conducting the two experiments, we aim to develop a data-driven model whose output matches the output of the actual system, when the same inputs are applied to both the model and the system, which is simple enough for estimation and identification purposes. We do not attempt to modeling the exact physical phenomena govern the dynamics of the CO_2. Yet, our model provides some insight on the actual spatially distributed dynamics of the CO_2 concentration since it is a PDE model.

We model the dynamics of the CO_2 concentration in the room using a convection PDE with a source term which is the output of a first-order ODE system driven by an unknown input which models the human's emission rate of CO_2. The source term represents the effect of the humans on the CO_2 concentration in the room. In our experiments, we observe a delay in the response of the CO_2 concentration in the room to changes in the human's input. For this reason, the source term is a filtered version of the unknown input rather than the actual input. We assume that the unmeasured input from the humans has the form of a piecewise constant signal. This formulation is based on our experimental observation that humans contribute to the rate of change of the CO_2 concentration of the room with a filtered version of step-like changes in the rate of CO_2.

The value of the PDE at the one boundary of its spatial domain indicates the CO_2 concentration inside the room at the location of the air supply. At this location, incoming air is entering the room, and hence, one can view the CO_2 concentration of the fresh incoming air as an input to the system. The value of the PDE at the other boundary of its spatial domain indicates the CO_2 concentration at the air return of the ventilation system. The air at this point is mixed with CO_2 that convects from the air supply towards the air return, and with CO_2 that is produced from humans. We consider the CO_2 concentration at this point as the output of our system. Any value of the PDE on an interior point of its spatial domain is an indicator of the concentration of CO_2 at the ceiling in a (non-ratiometric) normalized distance along an axis from the supply to the return vent.

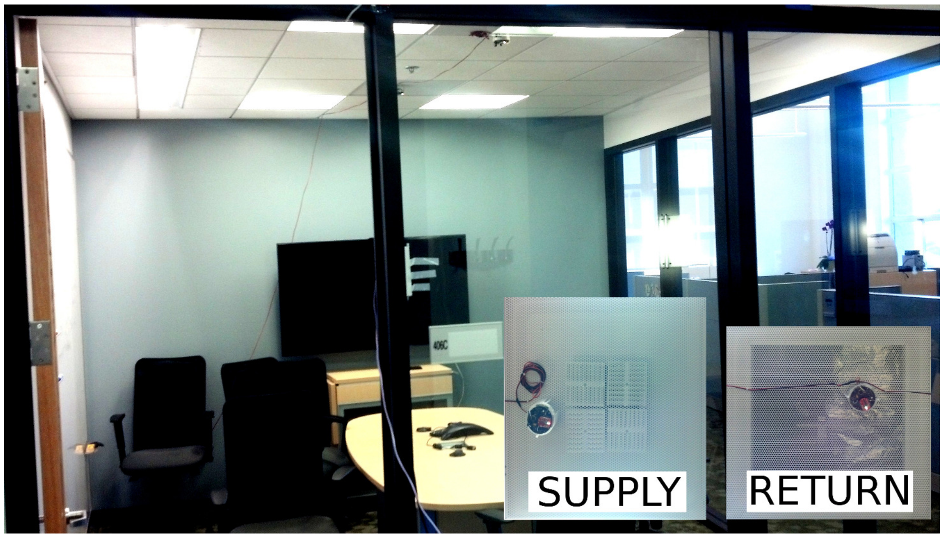


Figure 5.10: The conference room under study.

At the conclusion of our study [79, 108], we develop a model whose output closely matches the observed concentration and develop an estimator and identifier with proven convergence properties. These results are demonstrated to be applicable by running them against experimental data gathered during two experiments inside a conference room.

Procedure

Our experimental work takes place in a $44\,\mathrm{m}^3$ conference room, shown in Figure 5.10. The room is completely interior within the building and has no outside walls. On the ceiling there is one air supply vent with a diffuser and protective grate, and there is also an air return vent with a protective grate.

We measure CO_2 concentration using the K-30 Sensor Module [80] which comes with specifications of $\pm 30\,\mathrm{ppm} \pm 3\%$ accuracy and repeatability of $\pm 20\,\mathrm{ppm} \pm 1\%$. Since we expect the nominal CO_2 concentrations of the room to be no more than $1500\,\mathrm{ppm}$, this gives a repeatability error bound of $\pm 35\,\mathrm{ppm}$.

Constant errors in CO_2 readings are corrected by a method called *Baseline Correction*. Essentially, sensor readings are taken over a time period and the lowest concentration seen during that period is assumed to be $400\,\mathrm{ppm}$, corresponding the outdoor air concentration (i.e., the steady-state value if the room is ventilated and no occupants are present). The sensor itself performs this correction automatically over a seven and a half day interval. We manually perform this correction by operating all of the sensors overnight, then subtracting an offset from each data set so that the minimum readings from each sensor equaled each other. This ensures that all of our sensor readings are using the same baseline, even if this baseline is up to $(-30\,\mathrm{ppm} - 400\,\mathrm{ppm} \cdot 3\%) = -32\,\mathrm{ppm}$ away from the real value.

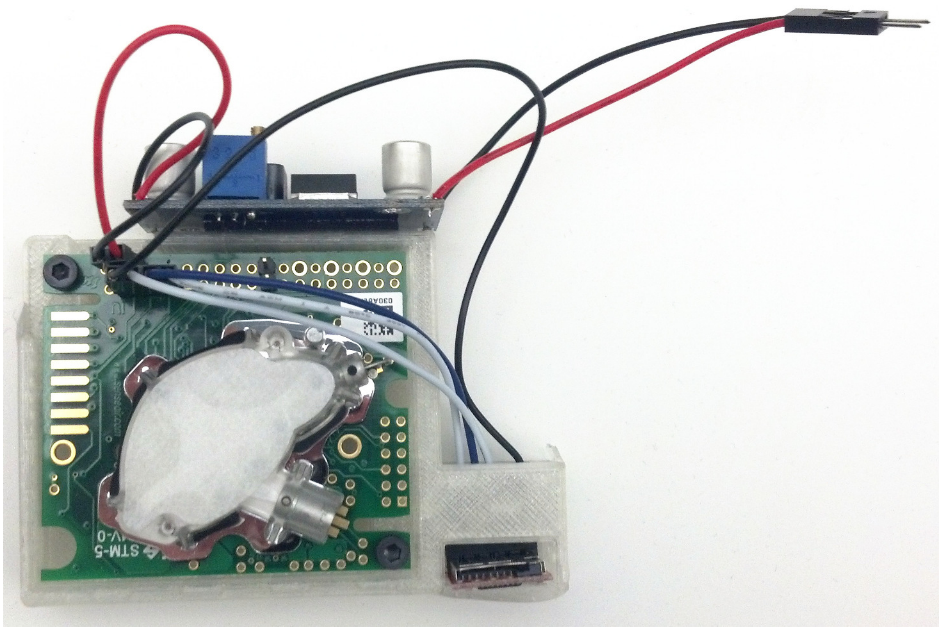


Figure 5.11: A CO_2 data-logging sensor which was used for the experiments.

Figure 5.11 depicts a close view of the data-logging CO_2 sensor used for these experiments. An external microcontroller is attached to the K-30 sensor and, every seconds, polls the current CO_2 concentration reading from the K-30 device. These recordings are written to an SD-card where they are stored until retrieved at the conclusion of the experiment.

Experiment I: Controlled CO_2 release

In the first experiment, we have two goals:

- The first goal is to examine the spatial dependence of CO_2 concentration in the room, in particular how well-mixed the air is. If there is a spatial dependence, we would like to identify the sensor which exhibits the most dependence on CO_2 generation in the room.
- The second goal is to collect data that can be used for manual or automatic identification of the parameters of a model whose output matches the measured data, when the same CO_2 input is applied to the model and the conference room.

Therefore, our testing methodology is to add a controlled disturbance of CO_2 into the room and measure the resulting response on the sensors placed in the room.

The disturbance input consists of beverage grade (99.9% purity) CO_2 gas being released via a flow regulator at approximately 2 CFM, and passed through a small 200 W personal heater, to simulate warm breath. A mechanical timer is used to switch the regulator and heater on and off with a 2 hr period (1 hr on, 1 hr off).

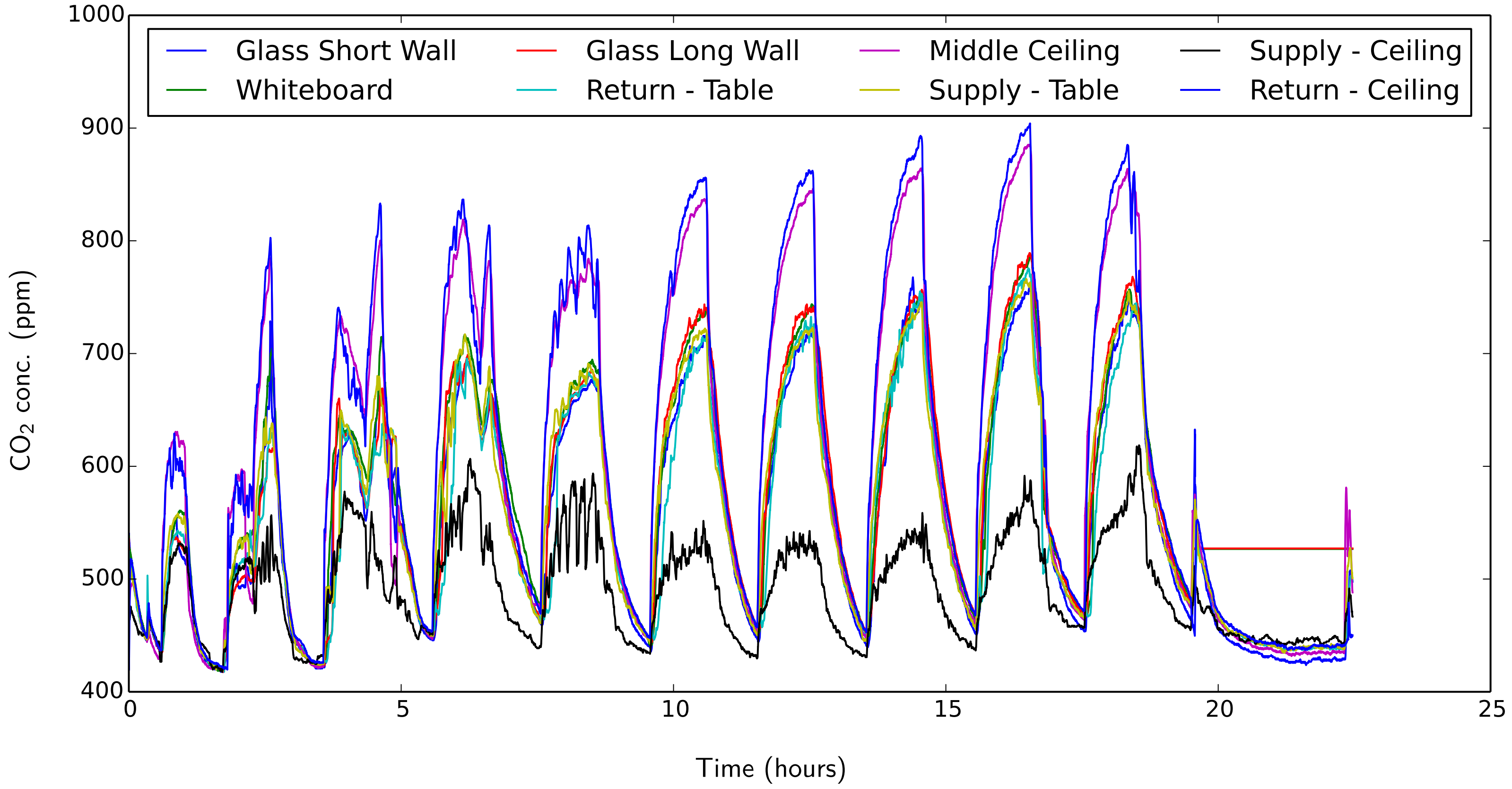


Figure 5.12: CO_2 concentrations during Experiment I. Showing measurements from all 8 locations over the approximately 22 hour experiment.

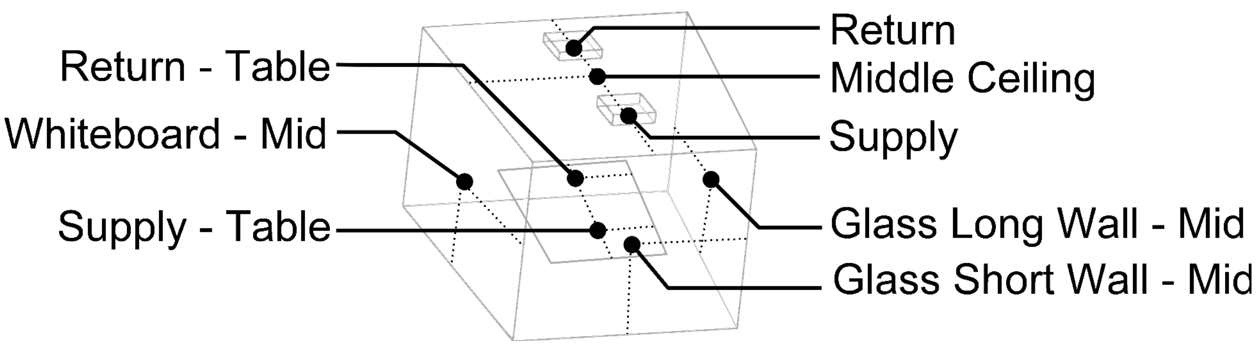


Figure 5.13: The locations of the CO_2 sensors during Experiment I.

We deploy a total of fifteen CO_2 sensors in the conference room at eight different locations as it is shown in Fig. 5.13. At seven of the eight locations, two CO_2 sensors are co-located for redundancy in case hardware failure made a reading invalid. We do not encounter any hardware failures during the experiment, so we instead take the mean of redundant measurements.

Figure 5.12 gives the sensor readings from this test. When the CO_2 injection is turned off, all of the measurements settle to a steady-state value, which is almost the same for all sensors. However, when CO_2 injection begins, we see clear spatial differences in CO_2 concentrations. During injection, the highest concentrations of 900 ppm are seen by sensors placed at the air return vent and sensors placed on the ceiling at the midpoint between the supply and return vents. The lowest concentrations are seen at the supply vent, which stays below 600 ppm. All of the other sensors, which are placed between chest and waist level in the room, exhibit similar behavior in response to the CO_2 injection. In general, besides transient behavior due to ventilation turning on and off, the CO_2 concentrations from different points in the room react the same, albeit with different magnitudes.

From this experiment, we conclude that, when CO_2 is being generated in the room, the concentration of CO_2 local to the air supply represents a mixture of the room's CO_2 concentration and that of the fresh air (about 400 ppm). Other than at the supply vent, we observe that there are large variations on the CO_2 concentration between points at the ceiling and points at table height. This is explained by the fact that a warm breath from an occupant acts as a "bubble" of gas that rises to the ceiling, since it is more buoyant than the ambient, cooler air. Also, we observe that there are smaller variations on the CO_2 concentration between different points at the ceiling.

Furthermore, we also conclude that, of all the sensors, the measurements most affected by the production of CO_2 were those taken at the air return vent. Therefore, these measurements will be most useful to observe and perform system identification with.

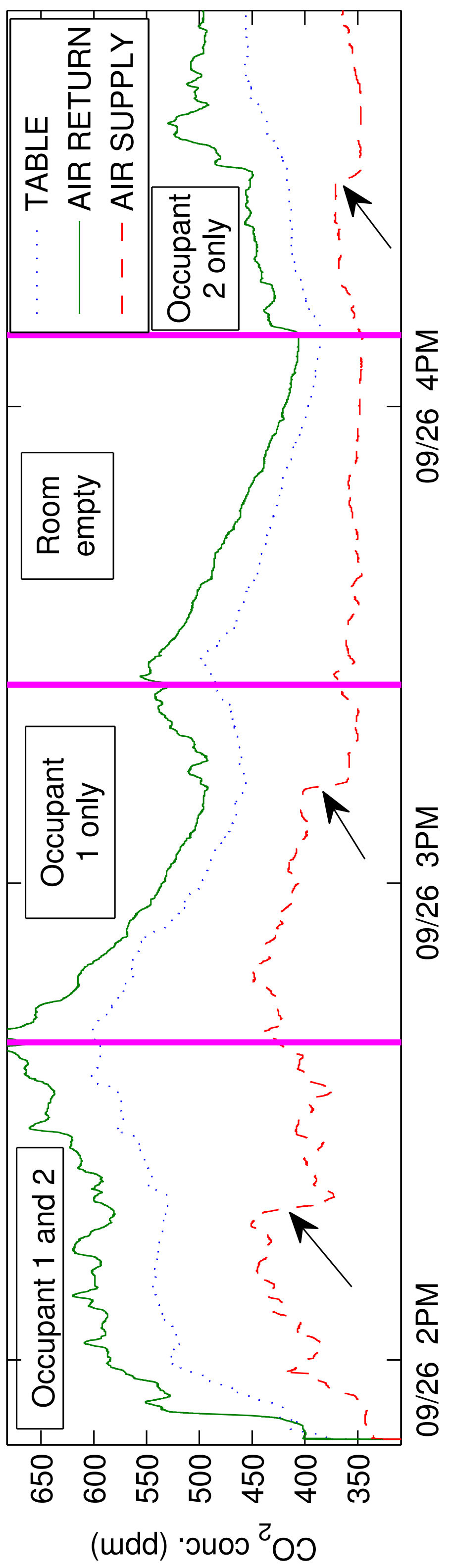


Figure 5.14: CO_2 concentrations during Experiment II. Showing measurements from 3 locations in the conference room over 3 hour experiment. Magenta lines indicate when occupancy changes occurred. The arrows indicate the time instants at which the ventilation rate increases[1].

Experiment II: Release of CO_2 from occupants

In the second experiment, the goal is to determine the effect of real occupancy on the concentration of CO_2 in the room. For this experiment, three CO_2 sensors are deployed: one each at the air supply and return vents, and one on the conference table at the center of the room. Our excitation procedure consists of adding or removing one of two participants of the experiment, and noting the time that the occupancy changes. Fig. 5.14 is a plot of the data gathered from this experiment and when the occupancy transitions occur.

From this plot, we can see the general trend that CO_2 concentration at the conference room table and at the return vent increases when occupants arrive and decreases when occupants leave the room. We also conclude that the concentration at the air supply vent is much less dependent on occupancy. This can be attributed to the constant fresh air ventilation that is provided by building services, so that fresh air concentration dominates the concentration in the area near the vent.

We can also see an interesting effect starting at the approximate times of 1:30PM, 2:20PM and 3:40PM, where the CO_2 measurement at the air supply sharply drops and corresponds to a rise in CO_2 in the other two measurements. We hypothesize this is due to the ventilation rate increasing. Near to the supply vent, a greater quantity of fresh air would mix with the air near the sensor, driving the concentration down. A higher air velocity in the room will also impart more turbulent mixing of pockets of CO_2 concentration within the room, pushing them out of the air return and increasing the concentration at that point. The mixing of these pockets also causes an increase in the CO_2 concentration near the table.

Modelling the CO_2 dynamics

Model design

The model for the CO_2 dynamics of the conference room are developed in [79]. Our model consists of a PDE and an ODE part. The ODE part is given by

$$\dot{X}(t) = -aX(t) + V(t) \tag{5.1}$$

$$\dot{V}(t) = 0, \tag{5.2}$$

where, $X(t)$, in ppm, models the source term of occupant CO_2 production on the relative concentration (in ppm) of the room in the local vicinity of the occupant (the evolution of which is described later on by a PDE), and $V(t)$ is a step-valued function, in ppm $\cdot$ s^{-1}, representing the level of the occupant CO_2 production rate within the vicinity of occupants. Parameter, $\frac{1}{a}$, in units of 100s, represents a time constant specifying how fast changes in occupancy affect the CO_2 concentration in the room, in the local vicinity of the occupant.

[1]We assume that the air ventilation rate increased at the marked points because an increased proportion of fresh air (typically 400 ppm) would cause a drop in the supply CO_2 concentration. The validity of this hypothesis could be verified by measuring the flow of incoming air at the supply with an anemometer.

The ODE is coupled with a PDE that models the CO_2 concentration in the room given by

$$u_t(x,t) = bu_x(x,t) + b_X X(t) \tag{5.3}$$
$$u(0,t) = U_\mathrm{e} - \Delta U(t), \tag{5.4}$$

with $\Delta U(t) = U(t) - U_\mathrm{e}$, where $u(x,t)$, in ppm, is the concentration of CO_2 in the room at a time $t \geq 0$ s and for $0 \leq x \leq 1$, $-b > 0$, in $\frac{1}{100\mathrm{s}}$, represents the rate of air movement in the room, and $b_X > 0$, in $\frac{1}{10^4\mathrm{s}}$, specifies the rate of diffusion of CO_2 from the local vicinity of the occupant to the room. The spatial variable x is unitless and represents a normalized distance along a horizontal axis that connects the air supply and air return. The air supply and air return are located at $x = 0$ and $x = 1$ respectively. Therefore, $u(0,t)$ is the CO_2 concentration inside the room at the location of the air supply and $u(1,t)$ is the CO_2 concentration inside the room at the location of the air return. The input $U(t)$ is the measured ppm concentration of the fresh incoming air. We do not simply specify the boundary condition at $x = 0$ as $u(0,t) = U(t)$. The reason is that during our experiments we observe that a sudden drop in the measured CO_2 concentration at the air supply results in an increase of the CO_2 concentration at the air return. Our explanation for this effect is that a drop in CO_2 concentration at the supply from its equilibrium value corresponds to increased airflow at the vent, i.e. more fresh air gets mixed in the local vicinity. The increased airflow has the effect of pushing pockets of CO_2 air out of the return vent. One way to capture this effect is to multiply the difference of the CO_2 concentration from its equilibrium value $\Delta U(t) = U(t) - U_\mathrm{e}$ with minus one, where U_e, in ppm, is the steady state input CO_2 concentration at the supply ventilation.

In Figure 5.15, we illustrate the geometrical representation of our model. The PDE portion of the model, $u(x,t)$, represents convection of air from the air supply to the air return vent near the ceiling. Note the absence of a diffusive term, which we have omitted since it plays a relatively minor role in dispersing indoor pollutants [107]. We choose to model the CO_2 concentrations near the ceiling since this is where we see most effect from occupant-generated CO_2 (see Section 5.3). This is explained by the fact that a warm breath from a occupant occupant acts as a "bubble" of gas that rises to the ceiling, since it is more buoyant than the ambient, cooler air. Thus, the air coming from lower in the room is modeled as a source term on the PDE across its entire length. The ODE portion of the model is intended to model the fact that this bubble of air does not immediately rise to the ceiling but only gradually.

Simulations

In Fig. 5.16 we show the concentration of CO_2 at the air return and the air supply measured by the CO_2 sensors for our first experiment in which we periodically release CO_2 every one hour. We also show the output $u(1,t)$ of our model with parameters as shown in Table 5.4[2]

[2]In this section we manually tune the parameters of model (5.1)–(5.4) in order to match the measured CO_2 concentration at the air return with $u(1,t)$. In Section 5.3 we design identifiers for online identification

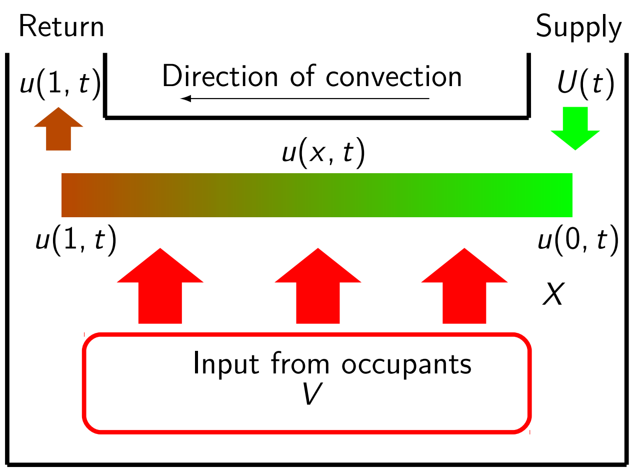


Figure 5.15: The geometrical representation of our model. Fresh air (U) enters the room from the supply ventilation. Air near the ceiling (u) convects from the air supply to the air return vent. The occupants produce CO_2 (V) which rises (X) to the ceiling.

and initial condition $u(x, 0) = 400$ ppm. The input V to our model, with which we emulate the behavior of the CO_2 that is released from the pump, is the square wave that is shown in Fig. 5.17.

In Fig. 5.18 we show the CO_2 concentration from Experiment II measured from the CO_2 sensor and predicted from model (5.1)–(5.4) with parameters shown in Table 5.5, initial condition $u(x, 0) = 400$ ppm, and input V that is shown in Fig. 5.19, with which we emulate the behavior of the CO_2 that is produced by occupants.

Estimation of the occupant-generated CO_2

Observer design

The observer for the plant (5.1)–(5.4) is developed in [108], assuming measurements of $u(1, t)$ and $U(t)$. This observer assumes that the parameters of the model are known, since they can either be manually identified (as in Section 5.3), or they can be identified using parameter identifiers (as in Section 5.3). From [108], we have the following observer, which is proven to be asymptotically stable:

$$\hat{u}_t(x,t) = b\hat{u}_x(x,t) + b_X\hat{X}(t) + p(x)\left(u(1,t) - \hat{u}(1,t)\right) \tag{5.5}$$

$$\hat{u}(0,t) = -U(t) + 2U_{\mathrm{e}} \tag{5.6}$$

$$\dot{\hat{X}}(t) = -a\hat{X}(t) + \hat{V}(t) + L_1\left(u(1,t) - \hat{u}(1,t)\right) \tag{5.7}$$

$$\dot{\hat{V}}(t) = L_2\left(u(1,t) - \hat{u}(1,t)\right). \tag{5.8}$$

of the parameters of model (5.1)–(5.4).

Table 5.4: Parameters of the model (5.1)–(5.4) for Experiment I.

Physical Parameter	Model parameter	Value
Convection coefficient $\left(\frac{1}{100\text{S}}\right)$	$-b$	0.8
Source term coefficient $\left(\frac{1}{10^4\text{S}}\right)$	b_X	0.2
Time constant of the occupants' effect (100s)	$\frac{1}{a}$	10
Equilibrium concentration at the air return (ppm)	U_e	450

Table 5.5: Parameters of the Model (5.1)–(5.4) for Experiment II.

Physical Parameter	Model parameter	Value
Convection coefficient $\left(\frac{1}{100\text{S}}\right)$	$-b$	0.8
Source term coefficient $\left(\frac{1}{10^4\text{S}}\right)$	b_X	0.16
Time constant of the occupants' effect (100s)	$\frac{1}{a}$	10
Equilibrium concentration at the air return (ppm)	U_e	370

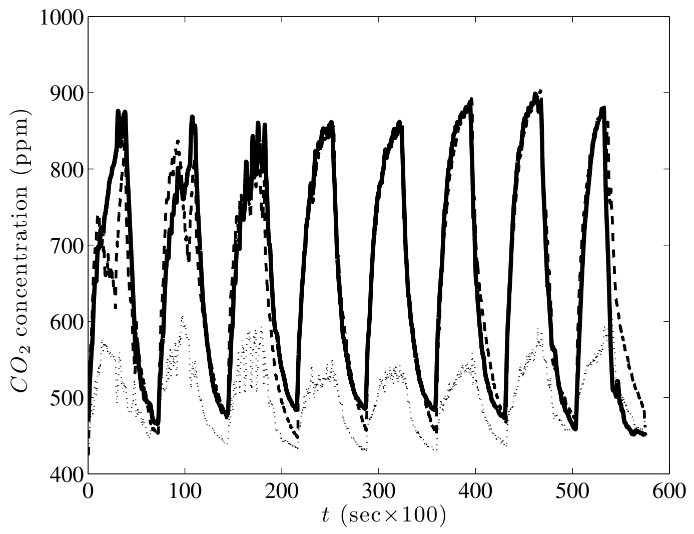


Figure 5.16: Solid line: The simulated concentration of CO_2 at the air return $u(1,t)$ given by the model (5.1)–(5.4) for Experiment I. Dashed line: The concentration of the CO_2 at the air return measured by the CO_2 sensor. Dotted line: The concentration of CO_2 at the air supply measured by the CO_2 sensor.

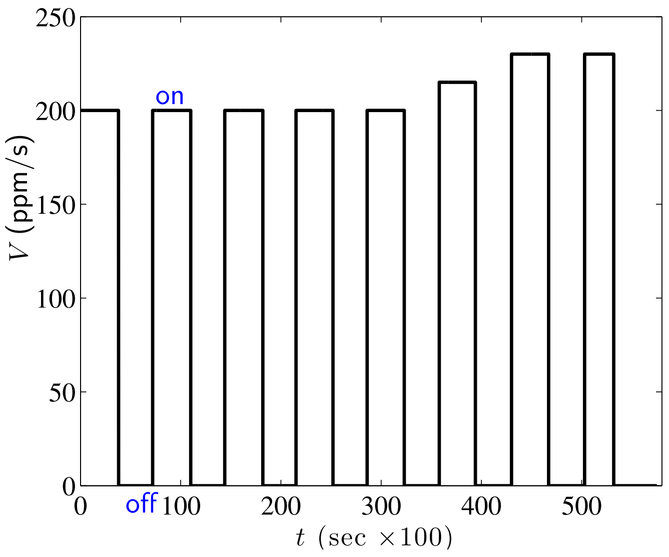


Figure 5.17: The input V to the model (5.1)–(5.4) from Experiment I modeling the concentration of CO_2 that is released from the pump. When $V = 0$ the CO_2 pump is turned off and when $V \neq 0$ the CO_2 pump is turned on.

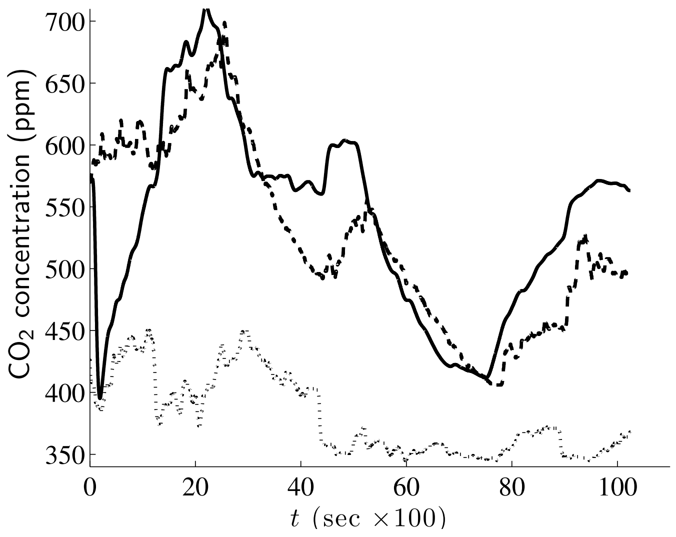


Figure 5.18: Solid line: The simulated concentration of the CO_2 at the air return $u(1, t)$ given by the model (5.1)–(5.4) for Experiment II. Dashed line: The concentration of the CO_2 at the air return measured by the CO_2 sensor. Dotted line: The concentration of CO_2 at the air supply measured by the CO_2 sensor.

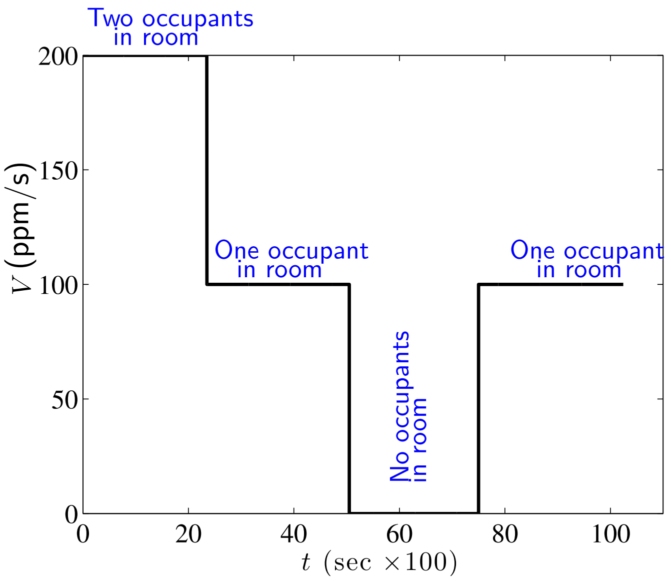


Figure 5.19: The input V to the model (5.1)–(5.4) from Experiment III modeling the input concentration of CO_2 from the occupants.

Simulations

We test our observer design for the model (5.1)–(5.4) by applying the input U that is measured from the sensor, and the input V which is shown in Fig. 5.17 and 5.19 for each of the two experiments. The initial conditions for the observer are $\hat{u}(x,t) = 400$, for all $x \in [0,1]$, $\hat{X}(0) = \hat{V}(0) = 0$. We choose the observer gains as $L_1 = 9.5$, $L_2 = 4$. In Fig. 5.20 we show the estimation of the state u together with the estimation error, and in Fig. 5.21 we show the estimation of the input V from the pump for the first 110×100 seconds for Experiment I. Using the same initial conditions for the observer and the same observer's gains as in Experiment I, we show in Fig. 5.22 the estimation of the state u together with the estimation error, and in Fig. 5.23 the estimation of the input V produced by the occupants for Experiment II.

Online model parameter identification

Identifier design

We also use the identifier developed in [108] for online identification of the parameters b, b_X and a. We start by assuming that the ODE and PDE states are measured. Directly measuring these quantities in an actual implementation might be impractical. Yet, our online parameter identifiers can be in principle combined with a state-estimation algorithm in order to simultaneously perform state estimation and parameter identification. However, in order to identify (offline) the parameters of our model this assumption is reasonable since we can use the data that we collect from our controlled-CO_2-release experiment in Section 5.3.

To define the identifier, we deal first with the identification of b and b_X. Define the "estimation" error

$$e(x,t) = u(x,t) - bv(x,t) - b_X p(x,t) - \eta(x,t), \tag{5.9}$$

between the measured state u and the signals v, p, η, where v is a filter for u_x, p a filter for X and η is an input filter, given by

$$\begin{aligned}
v_t(x,t) &= \hat{b}v_x(x,t) + u_x(x,t) && (5.10)\\
v(0,t) &= 0 && (5.11)\\
p_t(x,t) &= \hat{b}p_x(x,t) + X(t) && (5.12)\\
p(0,t) &= 0 && (5.13)\\
\eta_t(x,t) &= \hat{b}\eta_x(x,t) - \hat{b}u_x(x,t) && (5.14)\\
\eta(0,t) &= -U(t) + 2U_e. && (5.15)
\end{aligned}$$

The goal of the filters (5.10)–(5.15) is to convert the dynamic parametrization of the plant into a static one. This is the main attribute of the swapping identification method [109],

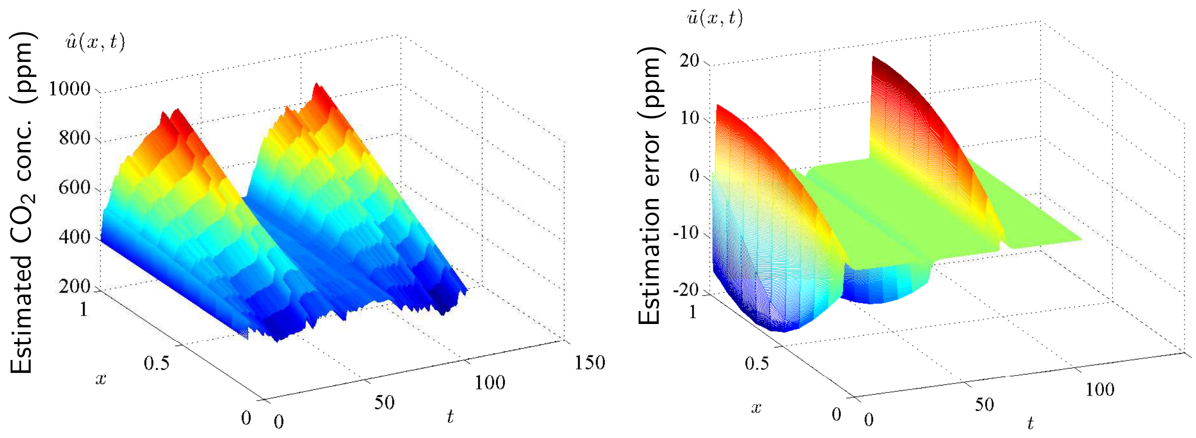


Figure 5.20: Left: The estimation of the CO$_2$ concentration $\hat{u}(x,t)$ in the room for Experiment I. Right: The estimation error $\tilde{u}(x,t) = u(x,t) - \hat{u}(x,t)$ of the CO$_2$ concentration in the room for Experiment I (shown after 1×100 sec for a better visualization).

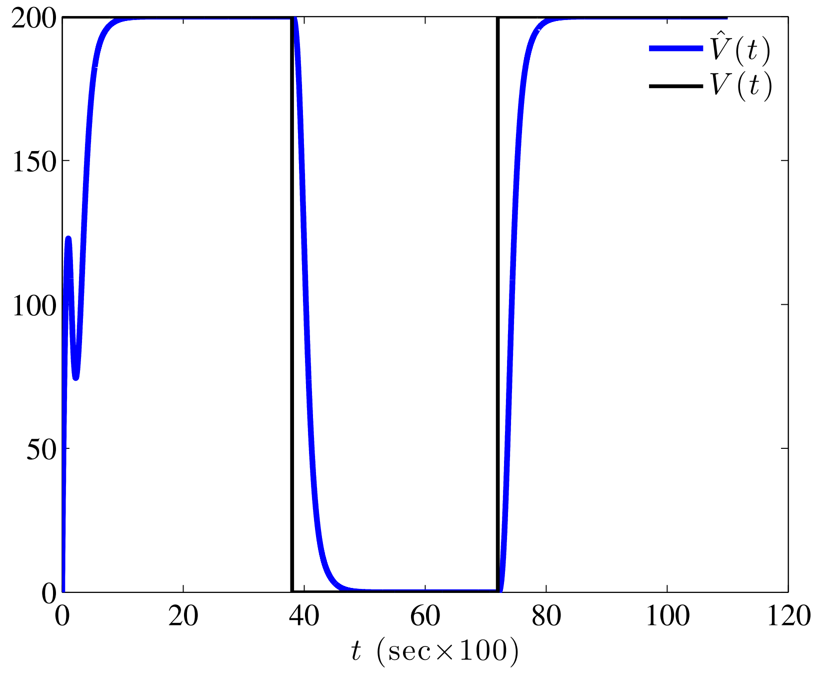


Figure 5.21: The estimation $\hat{V}$ (blue line) of the pump input V (black line) in Fig. 5.17 for Experiment I.

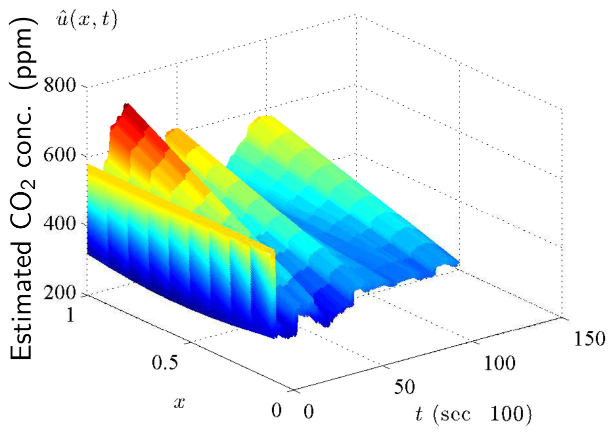


Figure 5.22: Left: The estimation of the CO$_2$ concentration $\hat{u}(x,t)$ in the room for Experiment II. Right: The estimation error $\tilde{u}(x,t) = u(x,t) - \hat{u}(x,t)$ of the CO$_2$ concentration in the room for Experiment II (shown after 3×100 sec for a better visualization).

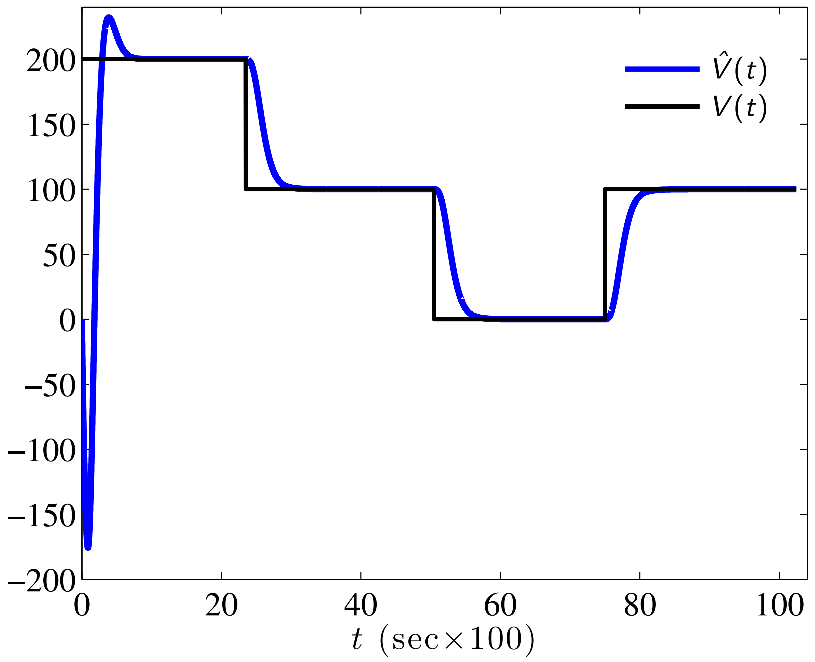


Figure 5.23: The estimation $\hat{V}$ (blue line) of the input V (black line) in Fig. 5.19 produced from the occupants for Experiment II.

[110], [111]. Using the static relationship (5.9) as a parametric model and defining the "prediction error"

$$\hat{e}(x,t) = u(x,t) - \hat{b}v(x,t) - \hat{b}_X p(x,t) - \eta(x,t), \tag{5.16}$$

the identifiers for b and b_X are given by the following gradient update laws with normalization

$$\begin{aligned} \dot{\hat{b}}(t) &= -\gamma_1 \hat{b}(t) \mathrm{Proj}_{\bar{b}} \left\{ \frac{\int_0^1 \hat{e}(x,t)v(x,t)dx}{1 + \int_0^1 v(x,t)^2 dx + \int_0^1 p(x,t)^2 dx} \right\} & (5.17) \\ \dot{\hat{b}}_X(t) &= -\gamma_2 \hat{b}(t) \frac{\int_0^1 \hat{e}(x,t)p(x,t)dx}{1 + \int_0^1 v(x,t)^2 dx + \int_0^1 p(x,t)^2 dx}, & (5.18) \end{aligned}$$

where the projector operator is defined as

$$\mathrm{Proj}_{\bar{b}}\{\tau\} = \begin{cases} 0, & \text{if } \hat{b} = \bar{b} \text{ and } \tau > 0 \\ \tau, & \text{otherwise} \end{cases}. \tag{5.19}$$

and $\gamma_1, \gamma_2 > 0$, $\bar{b} < 0$. The goal of the projection operator is to ensure that $\hat{b} < \bar{b}$.

We design next an online identifier for a. Define the filters

$$\begin{aligned} \dot{\Omega}_0(t) &= \bar{A}\left(\Omega_0(t) - X(t)\right) + V(t) & (5.20) \\ \dot{\Omega}(t) &= \bar{A}\Omega(t) - X(t), & (5.21) \end{aligned}$$

where $\bar{A} < 0$. Defining the error

$$\epsilon(t) = X(t) - \Omega_0(t) - \Omega(t)a, \tag{5.22}$$

the identifier for a is

$$\begin{aligned} \dot{\hat{a}}(t) &= \gamma_3 \frac{\hat{\epsilon}(t)\Omega(t)}{1 + \Omega(t)^2} & (5.23) \\ \hat{\epsilon}(t) &= X(t) - \Omega_0(t) - \Omega(t)\hat{a}(t), & (5.24) \end{aligned}$$

where $\gamma_3 > 0$. From [108], it can be proven that the estimated parameter $\hat{a}(t)$ converges to the true value of the parameter a. Furthermore, if the parameter b is known, then it can also be shown that the estimate $\hat{b}_X(t)$ also converges to the corresponding true value b_X.

Simulations

We initialize the filters as $v(x,0) = p(x,0) = \eta(x,0) = 50$, for all $x \in [0,1]$, and the update laws as $\hat{b} = -3$, $\hat{b}_X = 0$, and $\hat{a} = 0.5$. We set the upper bound of the estimation of b that is used in the projector operator as $\bar{b} = -0.1$. The gains of the update laws are chosen as $\gamma_1 = 3$, $\gamma_2 = 1.5$ and $\gamma_3 = 0.3$. In Fig. 5.24 we show the estimations of the parameters b, b_X, and $\hat{a}$ for Experiment I that converge close to their true values.

Using the same initial conditions for the filters and the update laws, as well as the same gains for the update laws, we show in Fig. 5.25 the estimations of the parameters b, b_X, and $\hat{a}$ for Experiment II that converge close to their true values as in the case of Experiment I.

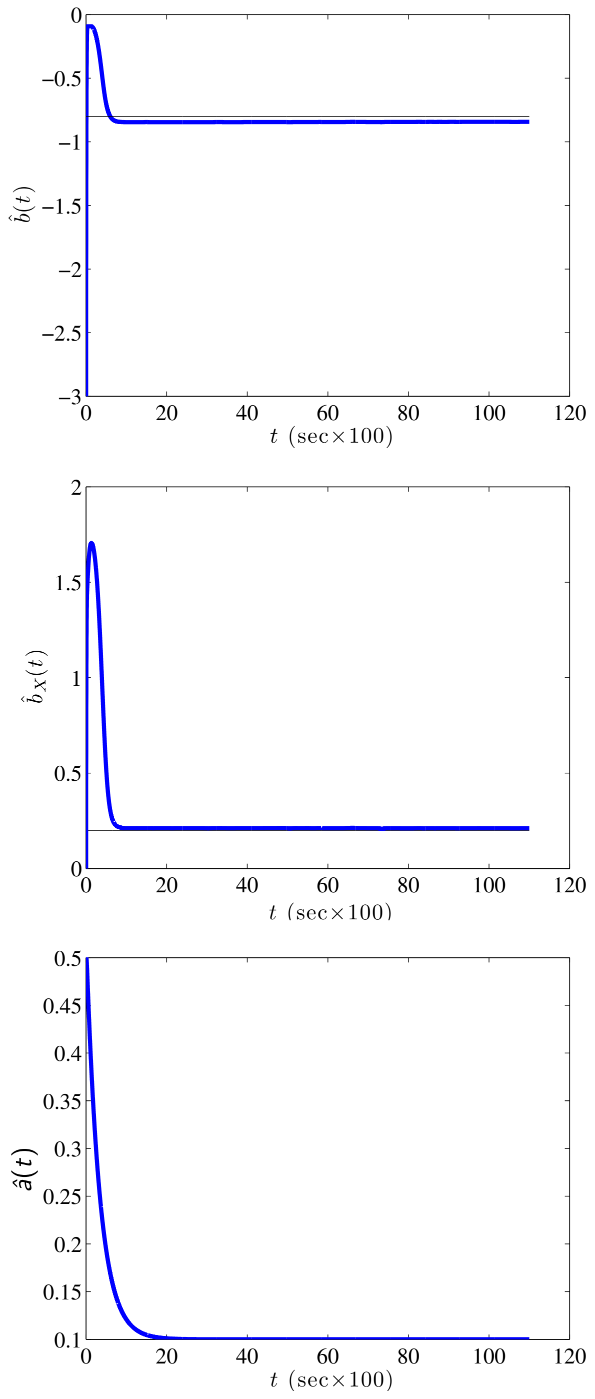


Figure 5.24: Blue line: The estimations $\hat{b}$, $\hat{b}_X$ and $\hat{a}$ of the parameters b, b_X, and a given by (5.17), (5.18), (5.23), for Experiment I. Black line: The true values of the parameters b, b_X, and a from Table 5.4.

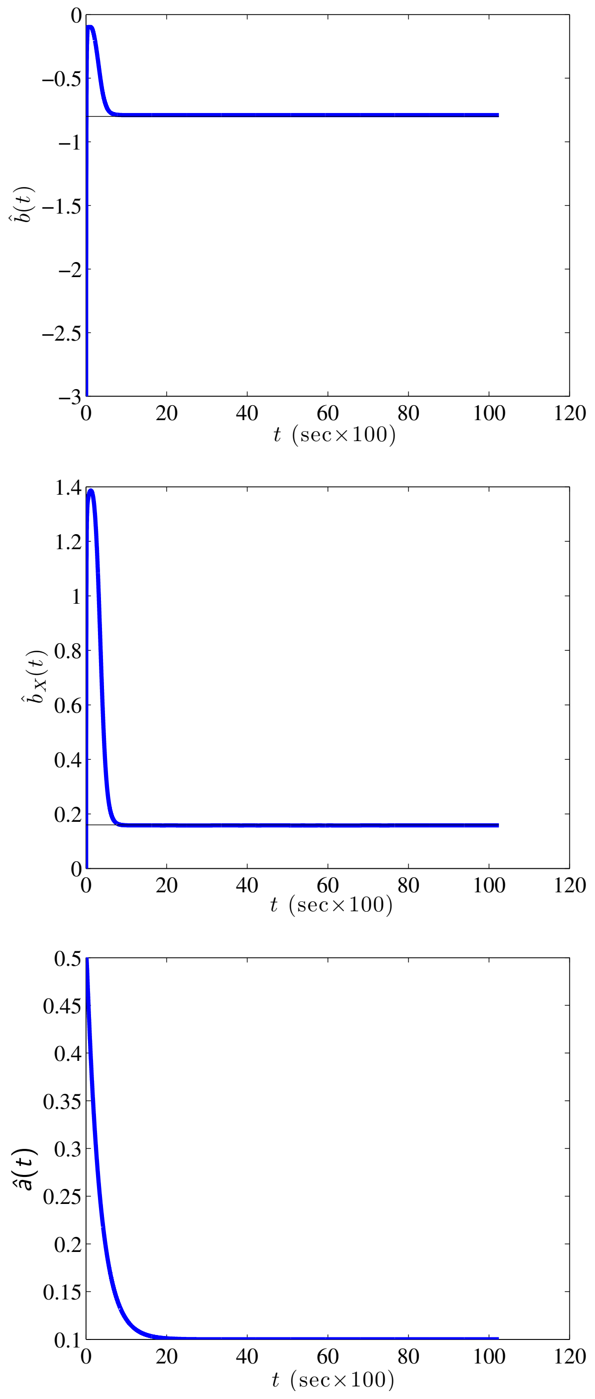


Figure 5.25: Blue line: The estimations $\hat{b}$, $\hat{b}_X$ and $\hat{a}$ of the parameters b, b_X, and a given by (5.17), (5.18), (5.23), for Experiment II. Black line: The true values of the parameters b, b_X, and a from Table 5.5.

Conclusions of the CO_2 study

Presently, we develop a PDE-ODE model that describes the dynamics of the CO_2 concentration in a conference room, and validate that the model is suitable using data from two ground truth experiments. We design and validate an observer for the estimation of the unknown CO_2 input that is generated by occupants, which acts as a intuitive proxy for the number of occupants breathing in the local air space. The estimator used in this study also has mathematical guarantees of convergence. To estimate unknown parameters of the model, we design online parameter identifiers which can be proven to converge to the true values, and we show in our experimental work that the parameters are indeed identified.

A topic for future research is to combine the observer design with the update laws for the estimation of the parameters of the model. In other words, to design an adaptive observer [109]. Yet, in contrast to the finite-dimensional case, in the case of PDE systems this is far from trivial due to the lack of systematic procedures for the construction of state-transformations that can transform the original system to a system having an observer canonical form [109], [111]. For this reason designing adaptive observers for PDE systems is possible only in special cases [111]. As an alternative one could resort to finite-dimensional approximations as it is done, for example, in [112].

To address the end application of occupancy estimation, future work will address the problem of estimation of the actual occupancy level using measurements of CO_2. This is a highly non trivial problem because occupants' CO_2 generation rates can vary widely between different persons depending on current activity, diet, and body size [105]. One possibility to address this problem is to build a database of parameters which characterize the amount of CO_2 generated by each individual, however, this would require a means of consistently identifying occupants. A more robust direction that we are pursuing is to use sensor fusion techniques to combine other sources of information, such as correlations to coarse PM (see Section 5.2), into the estimation.

5.4 Conclusions

The primary purpose of office and residential buildings is always to provide comfortable residence for their occupants to work and live. Heating, ventilation, air conditioning and lighting services are means of preparing the indoor environment for occupants, however provisioning and operating these services incurs significant cost and energy demands. To mitigate these costs, methods which estimate indoor occupancy can simply turn off lights and HVAC services when occupants are not present to consume them. Despite the simple control strategy, this can reduce consumption by up to 15% [54] when systems are heavily duty-cycled. However, perhaps more exciting is that once we can accurately estimate occupancy indoors, we can begin to improve models of occupancy in buildings. These models are used to predict future occupancy, so that building services can plan ahead for energy-efficiency, and building designers can appropriately provision services to suit future occupant demands.

The studies conducted in this chapter are the first steps in demonstrating that readings from indoor air quality (IAQ) sensors, in this case CO_2 and Particulate Matter (PM), can be used to estimate local occupancy. The techniques we develop do not require occupant-carried tags or an indoor positioning infrastructure to function. This is an important result since the types of sensors used are already being deployed for IAQ purposes, meaning we are adding value to an existing value proposition. Our hope would be that this leads in a direction that encourages building owners to deploy more IAQ sensors, especially given that Sick Building Syndrome is becoming more of a concern.

Although we specifically illuminate the advantages of directly estimating occupancy, we also note that the occupancy estimation problem is fundamentally coupled to the occupant tracking problem. In fact, we will demonstrate in the next chapter how occupancy information can be integrated into a positioning infrastructure. This is a new development in the field of positioning, whereas traditionally, occupant tracking has been used as an input to solving occupancy estimation only.

Chapter 6

Filtering Algorithms for Occupant Tracking

6.1 Introduction

Within the office space mobile environment, a valuable direction of research is to directly estimate the position of the occupants over time, i.e. occupant tracking. We have already discussed the benefits of occupancy estimation, which can calculated from accurate occupant tracking of all occupants. However, identifying and tracking individual occupants has benefits beyond occupancy estimation purposes. For instance, individual occupants could have different tolerances for indoor climates and lighting and building services could be adjusted accordingly. Another popular use of occupant tracking is to provide location-based services, such as indoor navigation, advertising and assisting occupants in finding each other indoors. There is also research interest in "social games" where occupants are incentivized to save energy in their daily routine and tracking the position of the participants is essential to enforcing the rules of the game. In this chapter, we construct a solution to the occupant tracking problem using particle filters. Via a field operational test, we validate that the particle filter is a viable method for occupant tracking, and can be the basis of many possible extensions. These extensions include connections to our work in occupancy estimation and on the development of the environmental sensing platform. Addressing the occupant tracking and occupancy estimation problems simultaneously, a unified particle filter is proposed which shares information between the two problems and incorporates information gathered from a variety of sensors.

Indoor Positioning Systems

The application area for Indoor Positioning Systems (IPS) is rapidly expanding as researchers discover location-aware services such as on-demand lighting or ventilation control. However, due to the unstructured nature of indoor environments, designing an IPS has many different challenges and requirements in comparison to outdoor and large-scale positioning systems [113]. The requirements of an IPS vary in cost, power usage, accuracy, reliability, and computational efficiency, depending on the application. This has led to many IPS concepts which fill particular niches. For example, a robot needs highly accurate positioning to avoid obstacles, whereas a location-aware service for occupants may only need to know which room an occupant is in. For the former case, a highly accurate ultrasound system [114] might be used. However, in the latter case, power consumption is a greater concern than positioning accuracy, where room-level accuracy might be sufficient.

Advances in low-power radios and improved wireless network protocols have enabled Radio Frequency Identification (RFID) as a practical solution to the occupant tracking problem, with many approaches of how to use radio measurements for positioning [84]. A popular avenue of research is to use existing WiFi infrastructure by measuring received signal strength (RSS) to the building's access points [85, 86]. However, some buildings do not have the access point density needed for the IPS to function, and more must be installed. Also, WiFi devices inherently have higher power requirements than other technologies, but are considered practical since many occupants already carry and use these types of devices, such as smartphones. Alternatively, IEEE 802.15.4 is a low-cost and low-power wireless sensor network technology which can be deployed for positioning and other low data-rate applications [115, 116]. A device based on this technology can be installed in an employee ID badge, or in our case, a small fob that the occupant carries.

Statistical techniques for radio-based indoor positioning

In this chapter, we describe a method to filter RSS measurements using a sequential Monte-Carlo Bayesian estimation technique [99, 5], also known as a *particle filter*. We implement the most basic form of the algorithm, called Sampling Importance Resampling (SIR). Particle filtering methods are especially applicable to unstructured environments due to their ability to elegantly handle non-continuous systems and integrate heterogeneous measurements. They have been used successfully in robotics [117], and in IPS installations with a combination of sensors such as RFID, inertial, infrared, and laser scanning [118, 119, 120]. A similar particle filtering IPS implementation [121] to ours uses WiFi instead of RFID technology and explores the same "histogram method" to model the RSS measurements. Their state is modelled as the occupant being at one of 510 predefined calibration cells. The occupant must travel along and adjacency graph between these cells. Our implementation also relies on an adjacency graph, but one which is parametrized and automatically computed from the floorplan layout. We also directly model the state as continuous 2D coordinates, instead of discrete locations.

Our IPS strictly uses RSS measurements, although we envision adding other types of sensors to improve the tracking performance. This article is an account of how our IPS is built up from the principles of particle filtering, including practical implementation details one might need if installing an RSS-based IPS system of their own. In particular, we provide a unique method of incorporating the RSS sensor measurements in a way that reduces dependence on a physical transmission model by creating empirical estimates of sensor error distributions. This leads our IPS implementation to achieve within a 5 m accuracy for 90% of estimates. We also provide a discussion of the computational requirements of our multi-threaded program, which achieves a real-time to execution time ratio of about 18000, leaving significant room for more complex models and tracking of several hundreds of occupants.

6.2 Indoor positioning using SIR

SIR estimation framework

In this section, we instantiate the SIR algorithm [5, 117] for our system. For this article, we denote x_t as a three-dimensional random variable corresponding to time t. We use a discrete-time algorithm with a time interval of δt, in seconds. The first two dimensions, $x_{t,1:2}$, are the coordinates, in meters, of the occupant-carried RFID tag being positioned, and the last dimension, $x_{t,3}$, represents an attenuation factor, in dBm, of the tag's local environment. This third term attempts to correct for calibration errors or differences in radio propagation between different tags. To simplify notation when discussing the state in a general context, we use $x_{1:2}$ to represent the coordinates of the tag and x_3 to represent the attenuation factor. We define y_t as the RSS observations measured at time t by the RFID system. This observation vector has N_{sensors} dimensions and measured in units of dBm.

The goal of the algorithm is to estimate the *belief distribution*

$$\Pr\left(x_t | y_{1\ldots t}\right), \tag{6.1}$$

i.e. the probability at time t of state x_t being the actual state of the system, given all of the past observations from $t = 1$. If we estimate the belief perfectly, then the maximum likelihood estimate,

$$\hat{x}_t = \underset{x}{\operatorname{argmax}} \Pr\left(x | y_{1\ldots t}\right),$$

is the ideal estimation of the tag's position given all of the previous data.

If we assume a hidden Markov Model, then the current observations only depend on the current state, i.e.

$$\Pr\left(y_t | x_{1\ldots t}, y_{1\ldots t}\right) = \Pr\left(y_t | x_t\right),$$

and the next state depends only on the last state, i.e.

$$\Pr\left(x_{t+1} | x_{1\ldots t}\right) = \Pr\left(x_{t+1} | x_t\right).$$

Applying Bayes rule and the above assumptions gives

$$\begin{aligned} &\Pr\left(x_t \middle| y_{1\ldots(t-1)}\right) = \\ &\qquad \int \Pr\left(x_t | x_{t-1}\right) \Pr\left(x_{t-1} \middle| y_{1\ldots(t-1)}\right) dx_{t-1}, \end{aligned} \tag{6.2}$$

$$\begin{aligned}\Pr\left(x_t|y_{1\ldots t}\right) &= \frac{\Pr\left(y_t|x_t\right)\Pr\left(x_t\middle|y_{1\ldots(t-1)}\right)}{\Pr\left(y_t\middle|y_{1\ldots(t-1)}\right)} \\ &= \eta\Pr\left(y_t|x_t\right)\Pr\left(x_t\middle|y_{1\ldots(t-1)}\right),\end{aligned} \tag{6.3}$$

where η is a normalization constant and essentially accounted for by (6.4).

Thus, (6.2) and (6.3), are a recursive solution to (6.1), and are termed the *prediction* and *update* steps, respectively.

For many problems, including the indoor positioning task, analytically evaluating (6.2) and (6.3) is infeasible, therefore we turn to Monte-Carlo methods where samples are used to estimate the distribution.

We track the belief by a set of n particles, where the i-th particle consists of a *state estimate*, $x^{(i)}$, and *importance factor*, $w^{(i)}$. Although not strictly necessary, when the importance factors are updated, they are renormalized so that

$$\sum_{i=1}^{n} w^{(i)} = 1. \tag{6.4}$$

The prediction and update steps are adapted to propagate samples and are as follows:

Prediction

For each particle, sample

$$\bar{x}_t^{(i)} \sim \Pr\left(x_t^{(i)}\middle|x_{t-1}^{(i)}\right),$$

so that the particles are now distributed as (6.2). The *state transition distribution*,

$$\Pr\left(x_t^{(i)}\middle|x_{t-1}^{(i)}\right),$$

represents the system dynamics and is described in Section 6.2.

Update

For each particle, evaluate the non-normalized importance weight

$$\bar{w}_t^{(i)} = \Pr\left(y_t^{(i)}\middle|x_t^{(i)}\right),$$

where the *observation distribution*, $\Pr\left(y_t^{(i)}\middle|x_t^{(i)}\right)$, represents the sensor noise and is described by Section 6.2. Then, the importance weights are normalized by

$$w_t^{(i)} = \left(\sum_{j=1}^{n} \bar{w}_t^{(j)}\right)^{-1} \bar{w}_t^{(i)}.$$

We then take n samples, $\left\{x_t^{(i)} : i = 1, \ldots, n\right\}$, from the discrete distribution defined over $\left\{\bar{x}_t^{(i)} : i = 1, \ldots, n\right\}$, where $\Pr\left(x = \bar{x}_t^{(i)}\right) = w_t^{(i)}$.

After both steps are run, the particles are will be approximately distributed according to the belief (6.1). Additionally, the pairs of $\bar{x}_t^{(i)}$ and $w_t^{(i)}$ approximate the probability density function (PDF) of the belief, i.e.

$$w_t^{(i)} \approx \Pr\left(\bar{x}_t^{(i)} \middle| y_{1\ldots t}\right)$$

Two more implementation details remain. Since this is a recursive algorithm, an initial state must be chosen for each of the particles $\left\{x_1^{(0)}, \ldots, x_1^{(n)}\right\}$, from a given prior distribution $f_1(x)$. The initial states are sampled from uniform distributions since little is known about the state of the system before any measurements arrive. For the coordinate states, $x_{1:2}$, the uniform distribution is over the entire floor area of the office space, whereas for the attenuation factor, x_3, we sample from $\mathcal{U}\left(-a_{\max}, a_{\max}\right)$, where $a_{\max}$ is a model parameter.

The second implementation detail is the method used to select the actual estimate, $\hat{x}_t$, of the state from the set of particles. We chose the estimate via one of the following two methods and evaluated both for their accuracy:

maximum likelihood particle (MLP)

$$\begin{aligned}\hat{x}_t &= \bar{x}_t^{(i)}, \\ i &= \operatorname*{argmax}_j w_t^{(j)}.\end{aligned}$$

nearest weighted mean particle (NWMP)

$$\begin{aligned}\hat{x}_t &= \bar{x}_t^{(i)}, \\ i &= \operatorname*{argmin}_j \|\bar{x}_{t,1:2}^{(j)} - m_t\|_2, \\ m_t &= \frac{1}{n}\sum_{k=1}^{n} w_t^{(k)} \bar{x}_{t,1:2}^{(k)}.\end{aligned}$$

State Transition Distribution

During the update step, there is the need to sample from $\Pr\left(x_t^{(i)} \middle| x_{t-1}^{(i)}\right)$. The state is comprised of the x-y coordinate pair of the tag, $x_{t,1:2}$, and the attenuation factor, $x_{t,3}$, which are sampled from two separate models.

Tag Coordinates

The state transition distribution essentially adds knowledge about the system dynamics into the estimation. For our problem, this means describing where a tracked occupant could move to (i.e. their future position $x^{(i)}_{t+1,1:2}$) between time t and time $t+1$, given their current position is $x^{(i)}_{t,1:2}$.

Common choices for the transition distribution for robot and occupant tracking problems incorporate the velocity or heading in the state and use kinematic models with additive gaussian white noise [120, 117]. Other, more complex models try to add intuition about human motion [122], or learn the model using machine learning techniques [123].

Our initial approach is to use a circular uniform distribution centered on $x^{(i)}_{t,1:2}$ with radius of $v \cdot \delta t$, where v is an average occupant's walking speed in m/s. This is easy to sample from, but does not incorporate any information about the local environment, such as the fact that occupants will not move into or through obstacles, such as walls and tables.

Therefore, we devise a method which includes reachability information in the transition distribution. We start with an *obstacle map* of the office space, shown in Figure 6.1, defining the domain of the space and the obstacles, such as walls or desks, which obstruct movement. The domain is discretized into square cells of width C in meters.

Before the IPS program can start, we must precompute a *look-up table (LUT)*. For each cell in the domain, we compute the set of cells reachable within a grid distance of $\lfloor v\delta t C^{-1} \rfloor$. We use depth-limited dynamic programming [124] and a 4-adjacency graph model of the cells to compute these sets.

Although the precomputation step is computationally intensive, it only needs to performed once if v, C, and δt are constant. The benefit is that we can construct a distribution which is easy to sample from and physically intuitive. Our sampling method is illustrated by Figure 6.2 for $\lfloor v\delta t C^{-1} \rfloor = 4$. The steps are as follows:

1. Find the index of the cell, c, corresponding to the coordinates given by $x^{(i)}_{t-1,2:3}$ (blue circle), shown by the red-bordered and green shaded cell,
2. Using the precomputed LUT, find the reachable set, $\mathcal{R}$, of cells for cell c, shown by the green and orange shaded cells,
3. Pick a cell r at random from $\mathcal{R}$, shown by the orange shaded cell,
4. Sample $x^{(i)}_{t,2:3}$ (yellow circle) from a uniform distribution over the points in r.

Other map-matching techniques rely on detecting obstacle crossings during the prediction step and either redrawing samples that are within an obstacle or terminating and reinitializing the particle [120]. Our method saves the computational load of detecting and handling obstacle crossings, at the cost of map precision, since it requires discretization. Our method also allows a more accurate representation for large δt, as it allows a particle to travel around an obstacle, provided it is reachable within δt time.

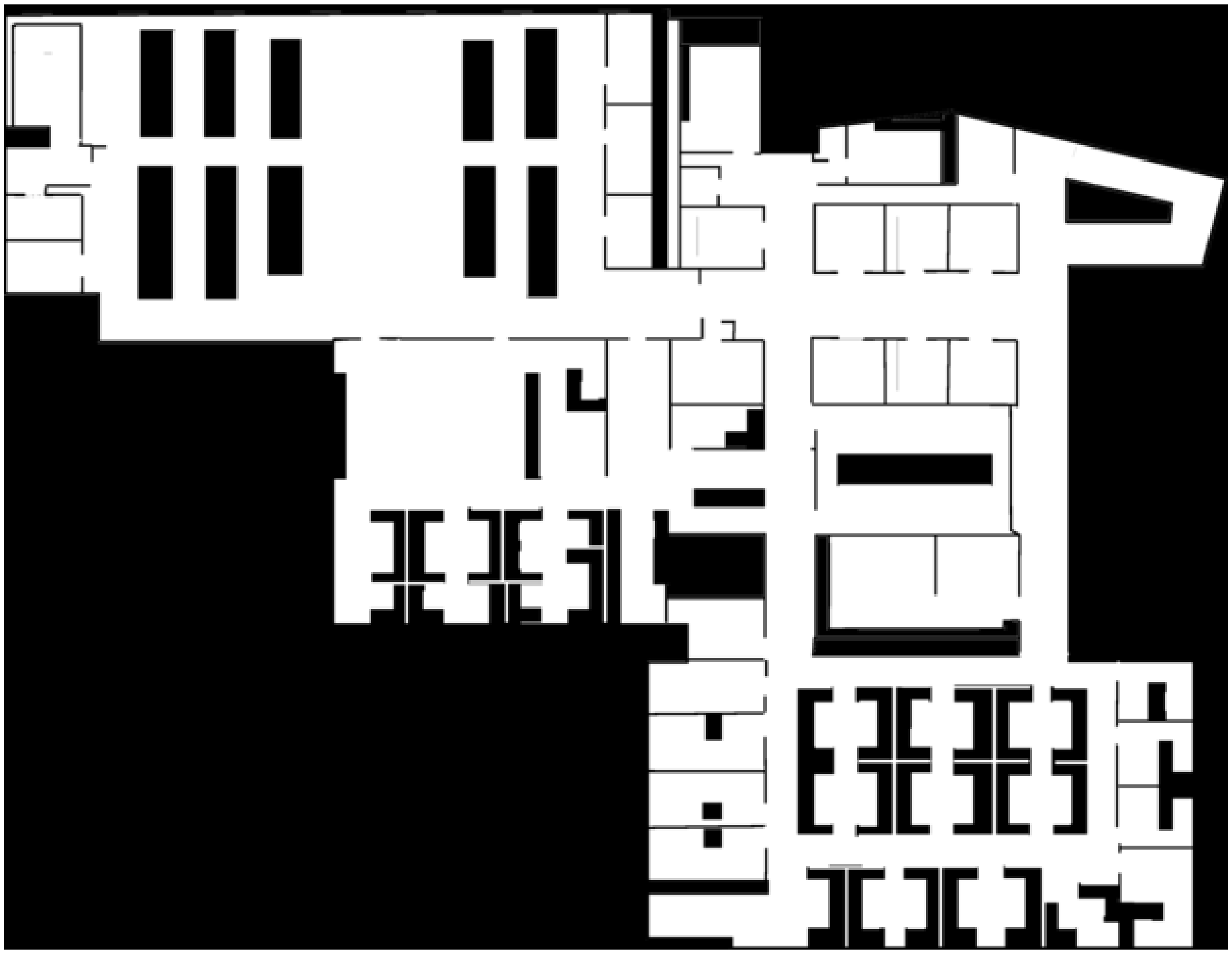


Figure 6.1: Obstacle map of the domain. Black areas represent obstacles that occupants cannot enter or pass through.

Attenuation Factor

In addition to the position components of the state, $x_{1:2}$, we also describe the state transition distribution of the attenuation factor x_3 by

$$\Pr\left(x_{t,3}^{(i)} \middle| x_{t-1,3}^{(i)}\right) = \mathcal{U}\left(x_{t-1,3}^{(i)} - \delta a, x_{t-1,3}^{(i)} + \delta a\right),$$

where δa is a model parameter.

Repositioning

One problem of particle filters is that the particle state estimates converge as more information becomes available and the variance of the belief is reduced. While this seems to be ideal, this means that the particle states cover less and less of the domain, and we lose

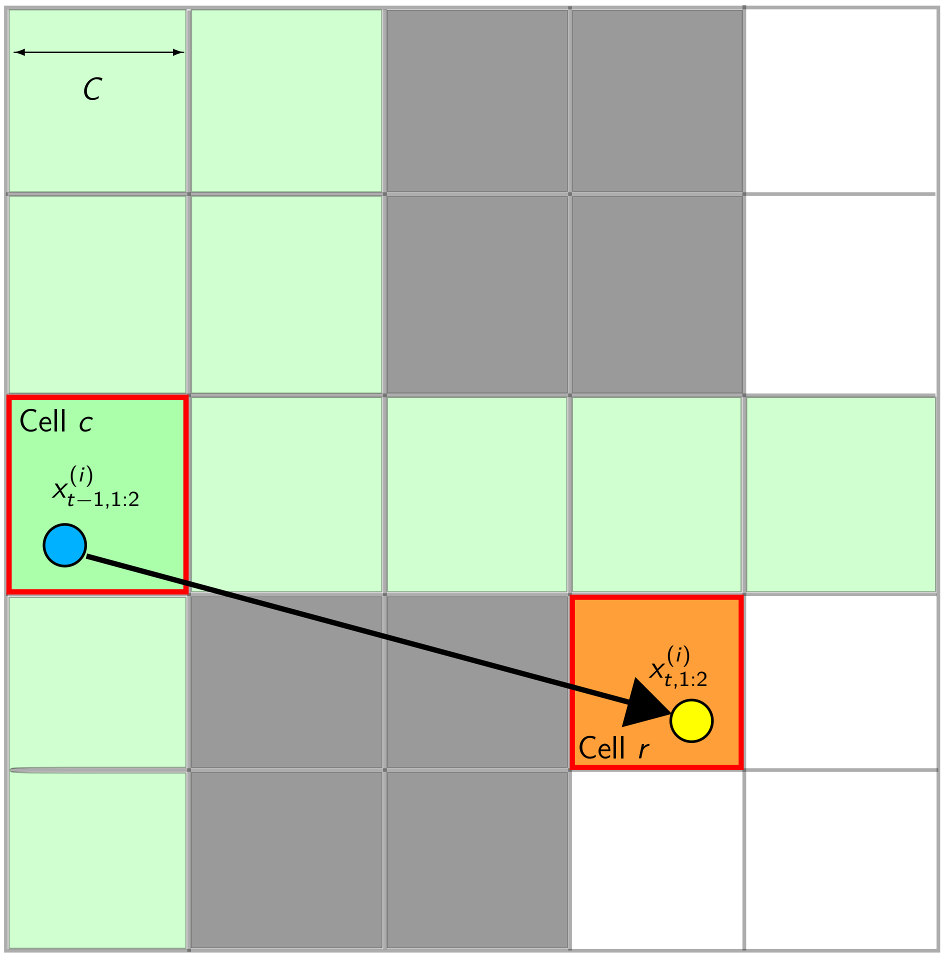


Figure 6.2: Illustration of the state transition sampling approach. Grey cells are obstacles, green and orange cells are reachable from cell c within a grid distance of 4. The orange cell r is picked at random from the reachable set. Blue and yellow circles represent the last and current state, respectively.

information about that part of the distribution. Thus, we augment the sampling algorithm with a *repositioning* step:

$$x_t^{(i)} \sim \begin{cases} f_1(x) & z < \Gamma \\ \Pr\left(x_t^{(i)} \middle| x_{t-1}^{(i)}\right) & \text{otherwise} \end{cases}$$
$$z \sim \mathcal{U}(0,1)$$

where $\Gamma \in [0,1]$ is the *repositioning ratio*, and an algorithm parameter. A $\Gamma > 0$ ensures that some fraction of particles are artificially moved to "explore" parts of the domain that might have been missed.

Observation Distributions

It is well known that modelling propagation of radio signals in an unstructured indoor environment is highly complex [125] due to varied reflection and attenuation properties of materials in the space, as well as multi-path affects, which are especially present with narrow-band and high-frequency channels. Therefore, there are a variety of models used for radio positioning systems such as ours. Some directly derive the model from physical phenomena [86], and others are data-driven models where many measurements are taken at fixed points to create an RSS *fingerprint* [116, 87, 85].

Our method is a data-driven model, but motivated by the physical property that the RSS will, in general, be lower as the sensor is farther from the transmitter. A broad description of the method is that, for each sensor, r, we radially partition the domain into N_{bins} evenly-spaced bins. For each bin, b, there is a corresponding empirically-derived PDF, $f_b^r(\cdot)$, of the signal strength measurement if the sensor is inside the corresponding partition.

To derive the method, we start by assuming the sensor outputs are independent from each other, given the state. Thus, we can decompose the observation distribution by

$$\Pr(y|x) = \prod_{r=1}^{N_{\text{sensors}}} \Pr(y_r|x),$$

where y_r is the RSS reading from the rth receiver. We model the individual sensor distributions as

$$\Pr(y_r|x) = \begin{cases} f_b^r(y_r) + x_3 & b \le N_{\text{bins}} \\ 0 & \text{otherwise} \end{cases}$$
$$b = \left\lfloor \frac{N_{\text{bins}}}{d_{\max}} \|p_r - x_{1:2}\|_2 \right\rfloor + 1,$$

where p_r is the coordinates of the rth sensor and $d_{\max}$ is an algorithm parameter and represents the maximum distance between any sensor and a possible tag's position.

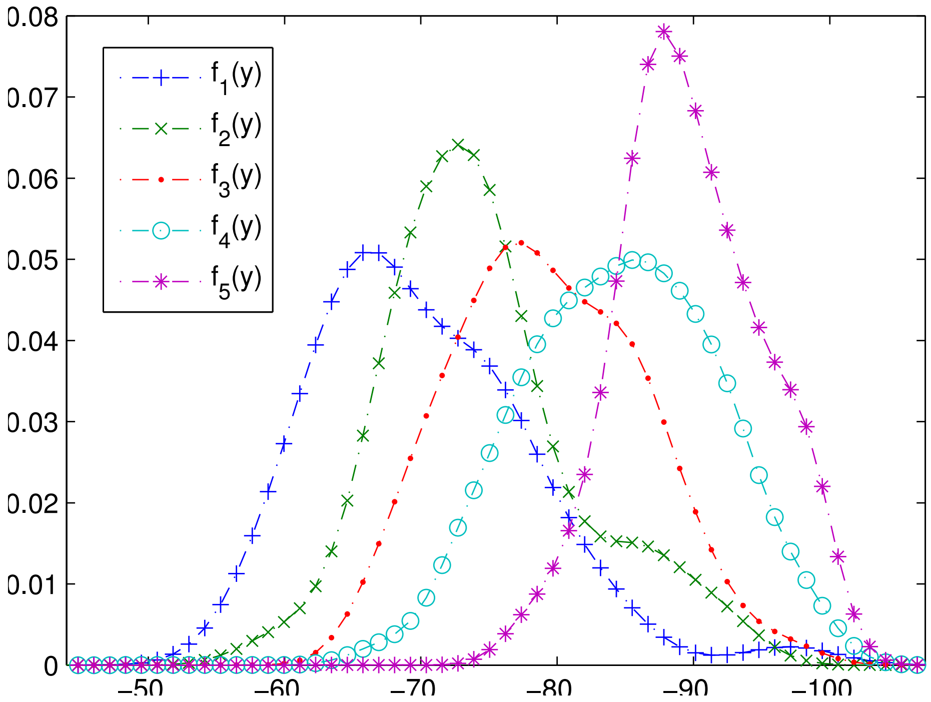


Figure 6.3: Example of learned probability distributions for a receiver for $N_{\text{bins}} = 5$ and $d_{\text{max}} = 21\,\text{m}$.

The distribution $f_b^r(y_r)$ is empirically derived from a ground truth training data set

$$\mathbf{GT} = \{(y_1, g_1), \ldots, (y_T, g_T)\},$$

where y_i is an RSS measurement, g_i is the coordinates where the transmitter is located when y_i is taken, and T is the total number of training data points.

The RSS measurements are partitioned into N_{bins} sets

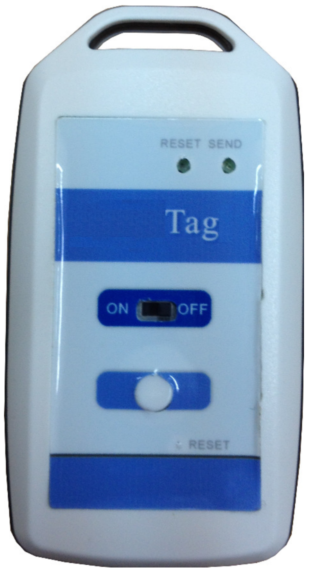
$$\{\mathbf{BIN}_i^r : i \in 1 \ldots N_{\text{bins}}\},$$

where

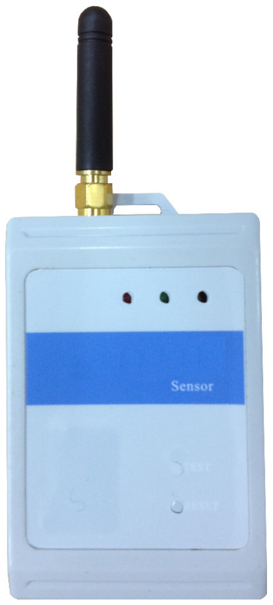
$$\mathbf{BIN}_i^r \triangleq \{y_j : (i-1) \cdot s \leq \|g_j - p_r\|_2 \leq i \cdot s\},$$

and $s = d_{\text{max}}/N_{\text{bins}}$.

Finally, we make a gaussian kernel estimate [126] using the `scipy.stats.gaussian_kde` function of the Scipy [127] distribution. The estimate is sampled at 100 points from $-125\,\text{dBm}$ to $-10\,\text{dBm}$ and used as the numerical estimate for $f_b^r(\cdot)$. Figure 6.3 shows the family of PDFs, $f_b^r(\cdot)$ for $b = \{1, \ldots, 5\}$, learned from a 40 min ground truth experiment, for a single sensor.



(a) Tag



(b) Sensor

Figure 6.4: Active RFID devices used by the IPS sensor network.

6.3 System architecture

Sensor placement

A grid of 24 sensors are installed in the false ceiling of a 16.5 m × 13.2 m (217.8 m^2) office area, above the ceiling tiles, to be hidden from view. The grid configuration is 6 by 4 sensors and the spacing between sensors is 3.3 m and 4.4 m, respectively. Power to the sensors is supplied by 5 V low-voltage wires run through the false ceiling, with 4 cable runs powering 6 sensors each. We measured the voltage drop at the end of each cable run to be less than 1 V, even when the system is powered on. Though this means there is only 4 V available to the last sensor, this is not a cause for concern since the sensors internally regulate the input power to 3.3 V.

Network setup

A diagram of the network configuration of our RFID-based IPS system is shown in Figure 6.5. Our sensor network is constructed of a low-cost network of ZigBee devices based on the TI CC2530 System-on-Chip [87]. There are three types of devices in the network:

- The *tags* (see Figure 6.4a) are small key-fob devices which are programmed with a unique 16-bit ID and broadcast this ID every second. They are designed to operate continuously for 1 month on a small lithium battery.

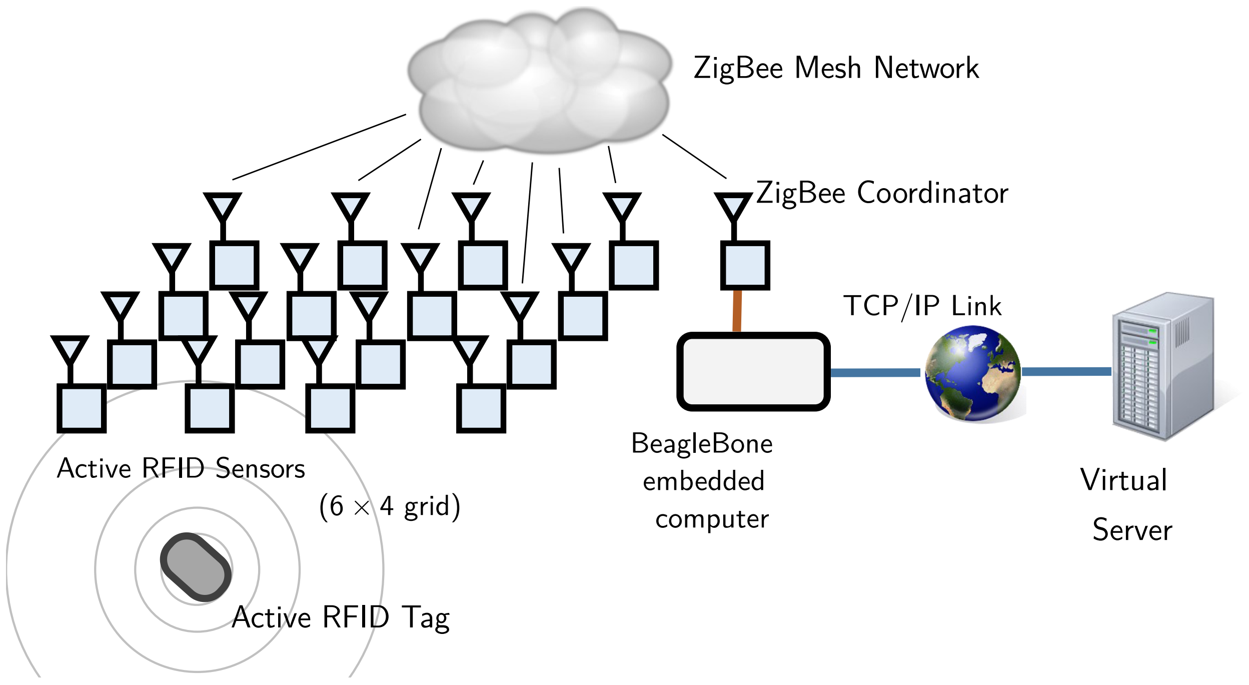


Figure 6.5: Network diagram showing data flow from beacons emitted by occupant-carried RFID tags to an Internet server which stores the data and runs the positioning program.

- The *sensor* (see Figure 6.4b) devices are low-cost (approx. 15 USD) and also programmed with a unique 16-bit ID. They form a ZigBee mesh network with the other sensors and coordinator. Upon receiving a beacon from any tag, they note that tag's ID and the RSS value of the beacon, and send it along the ZigBee network to the coordinator. The radios of the sensors must always be listening in order for the system to operate, thus these sensors cannot be battery-powered.
- There is one *coordinator* which initializes the ZigBee network and collects data packets from the sensor nodes. These packets are output in a custom format out of an external serial connection.

Not every beacon from the tags actually arrives at the coordinator due to network collisions or interference. For example, we experienced only a 8-20% successful transmission rate. We believe a higher transmission rate would greatly improve the performance of the IPS method.

A BeagleBone [128] embedded computer is connected to the coordinator and parses the custom format. The BeagleBone connects over the Internet via a TCP/IP socket to a Virtual Private Server (VPS) instance which handles sensor data storage and processing. Every second the software on-board the BeagleBone reports the RSS measurements received from the sensor network in the last second to the VPS.

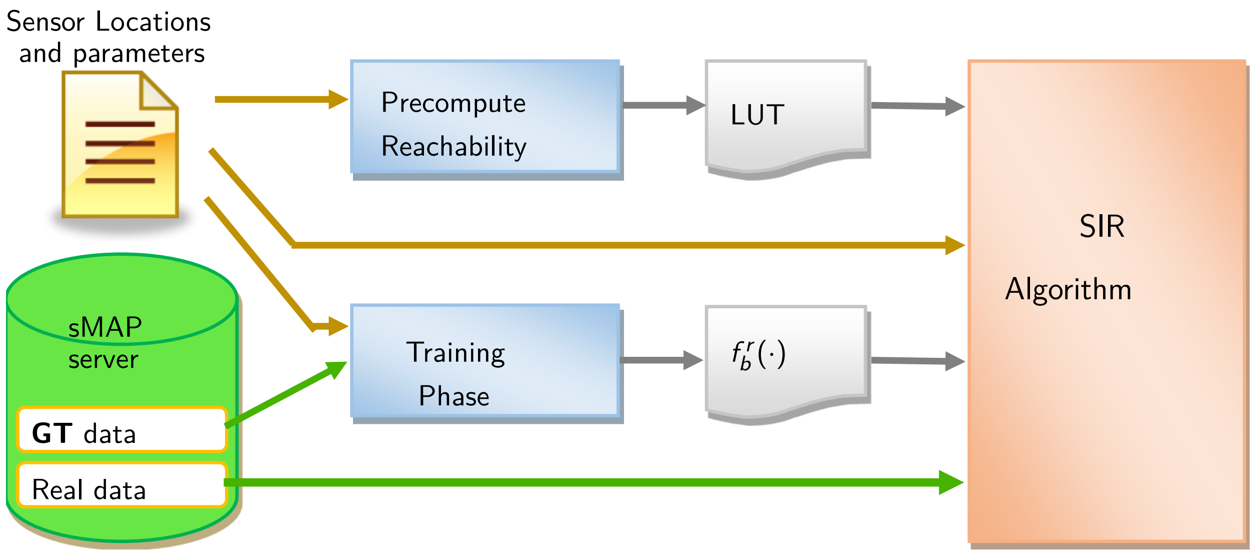


Figure 6.6: Process of the IPS software flow. Blue processes are precomputation steps, and grey files are intermediate outputs, but can be reused as long as parameters are held constant.

The VPS runs an instance of the sMAP [60] server, which efficiently stores the sensor data. The VPS also runs custom software to aid in translating information to and from the sMAP format.

Software setup

Precomputation

The software process for retrieving and generating the necessary inputs to the SIR algorithm is shown by Figure 6.6. The precomputation steps are to compute the reachability LUT for the state transition distribution described in Section 6.2, and also to compute the numerical probability distributions from the ground truth data, referred to by Section 6.2.

We relied heavily on the SciPy [127] distribution to implement loading and manipulating ground truth data for generating $f_b^r(\cdot)$. Fortunately, despite using a non-compiled language such as Python, an input of 14760 data points is processed in less than 1.7 s.

Although computing the reachability LUT is computationally arduous, a small cell size, C, is desirable to reduce discretization error. Therefore, this part of the algorithm is incorporated into our C++ framework and compiled with a high optimization level. Computing the LUT for 49152 grid cells at $C = 25$ cm, takes less than 30 s, and for the default parameter value of $C = 75$ cm, computing the map takes less than 500 ms.

SIR Algorithm Program

We implement the SIR algorithm as a multithreaded application in C++ and emphasize speed of execution. Our development machine has 4 cores, so we use a pool of 5 worker threads and one coordinator thread, in an attempt to saturate the CPU with workload. We are able to use multiprocessing techniques whenever each particle needs to be independently processed without interaction from others. In the SIR algorithm, this is during the prediction step and while evaluating $\Pr\left(y_t^{(i)} \middle| x_t^{(i)}\right)$ during the update step. For these tasks, we divide the set of particles into 5 equal subsets and each worker thread is responsible for predicting and updating their respective subset. There is also the need to sort an n-sized array during the update step, when the particles are resampled. At this time, the worker threads are used to sort 5 subarrays and the coordinator thread merges the subarrays into one sorted array.

The C++ program also uses Simple Directmedia Layer (SDL) [129] to animate the progress of the particle filter. The visualizer shows the obstacle map, the particle state coordinates, sensor positions, IPS output estimate, and the actual position of the tag, if given. These visualizations assist in debugging the implementation and tuning parameters. An example of 6 steps of the SIR algorithm in progress is shown by Figure 6.8.

6.4 Field operational test

To evaluate the method, we ran three experiments and collected three corresponding sets of RSS measurements. All of the described experiments consisted of walking once along the trajectory shown in Figure 6.7.

Ground Truth (GT) was to collect the learning data needed for the calibration described in Section 6.2. Average speed was 5 cm/s, enforced by pausing for 10 s every 50 cm. Three tags were carried for this test, to increase the number of data points collected. The experiment lasted approximately 40 minutes and collected 14760 (8.5% of sent) RSS measurements.

Tuning (TUN) was to tune the model parameters to a quicker moving particle than the **GT** data set. The average speed was 50 cm/s and 1106 (19% of sent) RSS measurements were taken.

Verification (VER) was reserved to evaluate the performance of the method and we do not use the results to tune or adjust the algorithm. The procedure was the same as the **TUN** experiment and 721 (12.5% of sent) RSS measurements were taken.

The performance of the IPS is measured by the **RMS** and **MAX** evaluation functions:

$$\mathbf{RMS}(\hat{x}, x^\star) = \sqrt{\frac{1}{T}\sum_{t=1}^{T} \|\hat{x}_t - x_t^\star\|_2^2}$$

$$\mathbf{MAX}(\hat{x}, x^\star) = \max_{t=1,\dots,T} \|\hat{x}_t - x_t^\star\|_2$$

Parameter	**Description**	**Default**	**Effect of Adjusting**
δt	Time interval	5 s	Best performance between 3 s and 9 s.
n	Number of particles	100	Performance degrades below $n = 60$, otherwise marginal improvement with more particles.
Selection Method	Method to select $\hat{x}_t$ from the set of particles.	NWMP	Using MLP results in slightly reduced performance.
Γ	Repositioning ratio	0%	Slightly worse performance from $0 < \%\Gamma \leq 50\%$, significantly worse when $\Gamma > 50\%$.
a_{max}	Maximum initial attenuation	1.5 dBm	No effect on performance.
δa	Maximum change of attenuation	0.75 dBm	Slight improvement inside 0.25 dBm and 0.75 dBm.
C	Reachability LUT cell width	0.75 m	Significantly worse performance when $C > 0.75$ m.
v	Occupant move speed	0.5 m/s	Significant decrease in performance as v moves away from 0.5 m/s.
N_{bins}	Number of partitions for the calibration in Section 6.2	5	Significantly worse performance for $N_{\text{bins}} \notin \{3, 5\}$.

Table 6.1: List of parameters tuned and their effects on the accuracy of the IPS.

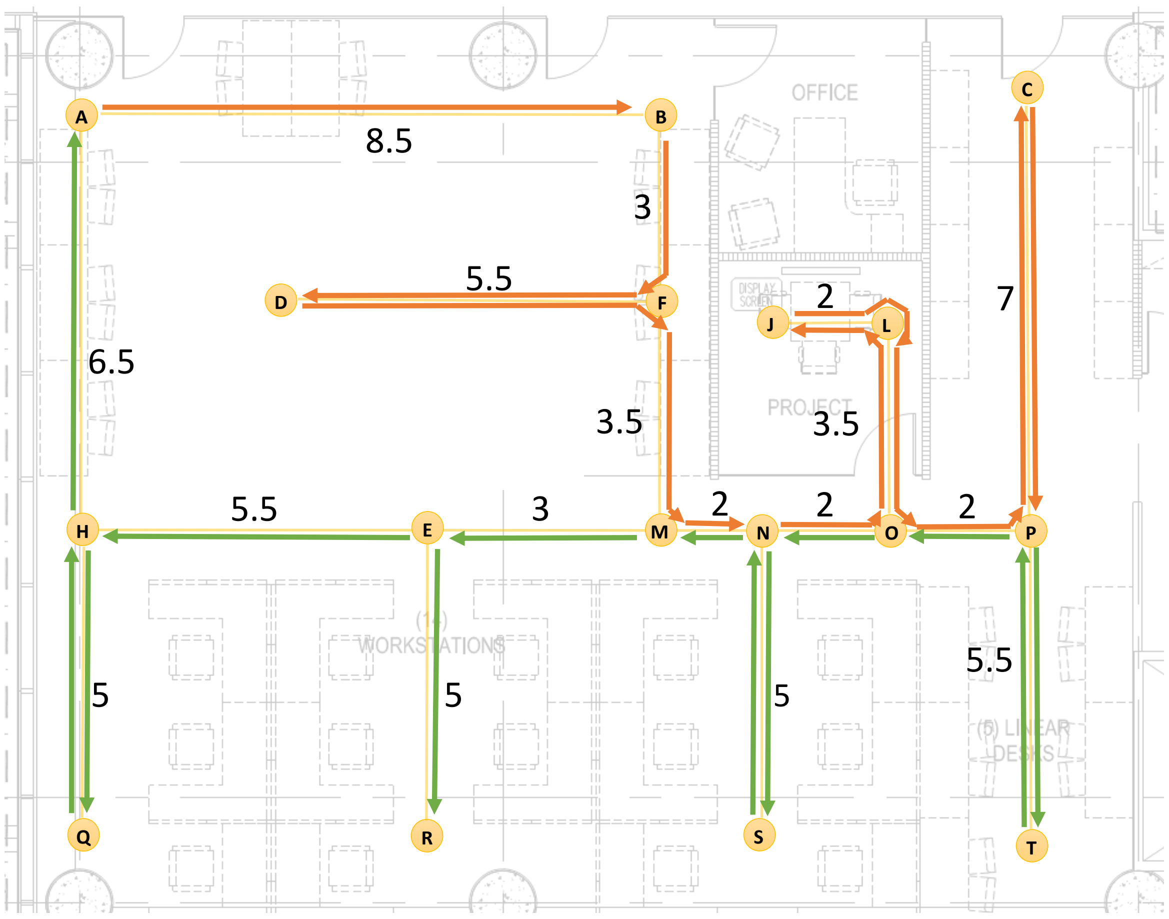


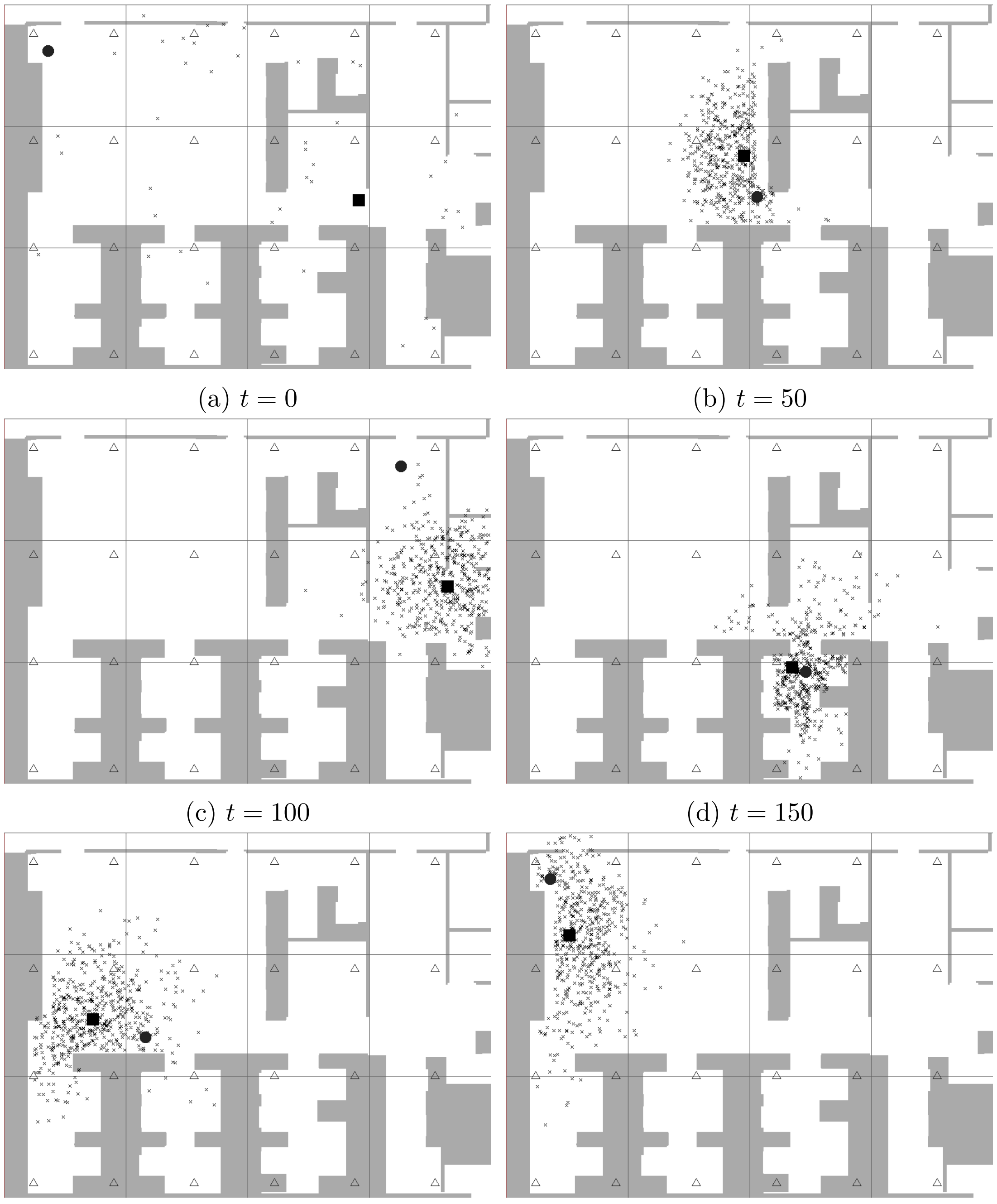
Figure 6.7: Ground truth trajectory diagram plotted over the floor plan of the office space. Trajectory starts at anchor point A and all distances are in meters.

where $x^\star$ is the known actual positions of the tag and T is the number of estimates generated.

Parameter tuning

Our IPS implementation has 8 main parameters shown by Table 6.1. For the "Effect" column, other parameters are held at their default values while the tested parameter is adjusted. The automated testing is performed with a Python program which runs the IPS program 10 times for each parameter choice and calculates the **RMS** and **MAX** metrics of the tests.

The defaults for the parameters are determined by manual inspection. First we pick realistic starting values (e.g. $v = 0.5\,\mathrm{m/s}$ is the known speed of the moving occupant). Then the Python program is run to sweep each parameter over a range of hand-picked values, usually a linearly-spaced numerical range. It is infeasible to iterate combinatorially over every configuration of parameters. Therefore, when testing, each parameter is swept



(a) $t = 0$

(b) $t = 50$

(c) $t = 100$

(d) $t = 150$

(e) $t = 200$

(f) $t = 250$

Figure 6.8: Intermediate steps of the SIR program visualized. Particle state estimates are small crosses, the NWMP estimate is a square, the actual tag's position is a circle, and triangles give the sensor positions. Grid lines are spaced every 5 m. Parameters for the simulation were set to default values given by Table 6.1, except $n = 600$, for illustrative purposes.

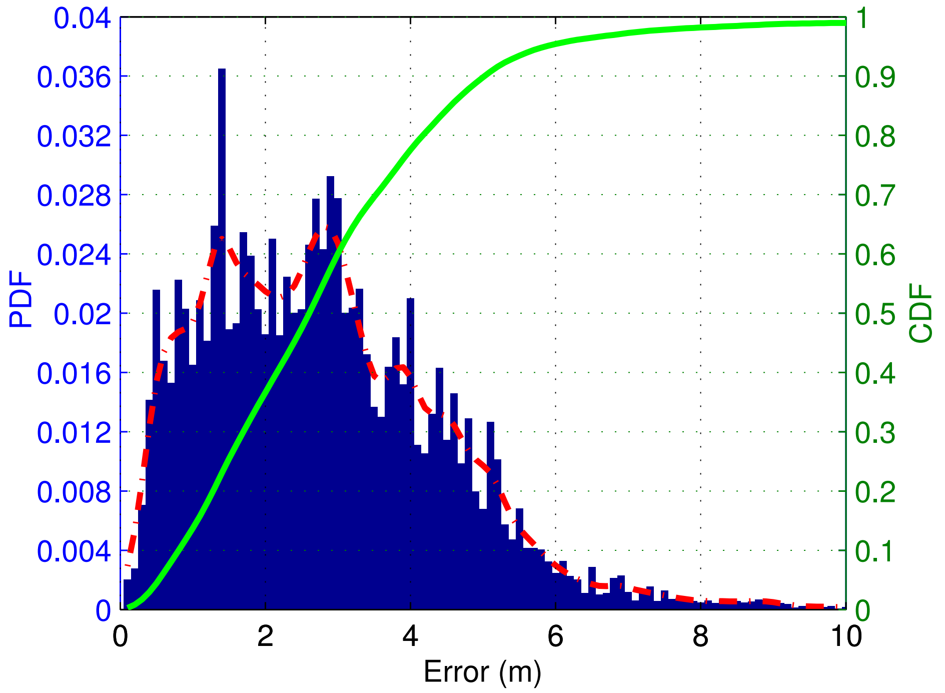


Figure 6.9: Histogram of error magnitudes collected over 1000 tests using the **VER** data set. Dotted line shows PDF estimate and solid line shows cumulative distribution function estimate.

independently, while the others are held constant at their defaults. Then, we set the default for the tested parameter to the value corresponding to the lowest value and performed the sweep on another parameter. The method required significant supervision to find the default values in Table 6.1, but automating the process is straightforward if needed for an integrated IPS solution.

Note that, in this analysis, a couple of the features of our method seem to be irrelevant, such as a_{max}, or even degrading to the performance, such as repositioning, which is used when $\Gamma > 0$. For the case of repositioning, the benefit is that it helps to recover the filter if the particles get "stuck" in some part of the state space, which happens rarely. However, this comes at a cost of aggregate performance.

Accuracy

Using the default parameter values and running the IPS program 1000 times on the **VER** data set, we achieve an accuracy of 90% estimates within 5 m. The mean error is 2.9 m and RMS error is 3.6 m. A histogram of the 144 thousand error samples is plotted in Figure 6.9. From the histogram we can conclude that, while the mean error is within room-level accuracy, there is still a significant probability of large errors, e.g. 1% probability of error being greater

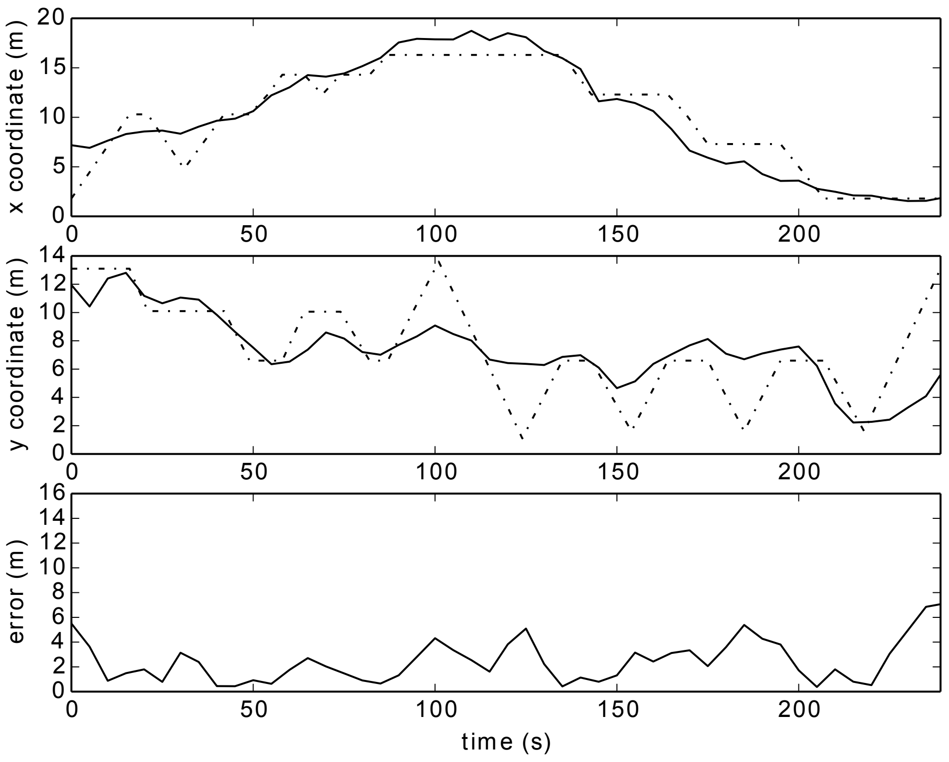


Figure 6.10: Example trajectory using default parameter values and **VER** data set. Dotted line is actual trajectory of occupant.

than 9 m. Thus, the direction of future work will focus on improving the reliability of the IPS by adding sensors, such as occupancy sensors, which can constrain particle estimates closer to reality.

An example of the errors seen during two tests is shown by Figure 6.10, where the timeseries of coordinates and the error term is plotted. In Figure 6.11 we plot the true and estimated locations on a map of the space.

Examining these results, our IPS is able to achieve substantially better tracking along the x dimension than the y dimension. This could be a result of there being 6 sensors along the x dimension and spaced 3.3 m apart, rather than 4 sensors spaced 4.4 m apart in the y dimension.

Another contributing factor is that the particle estimates seem to lag behind the true value, especially during fast moves, such as those along the y dimension. The difference is made more clear in Figure 6.12, where we plot the estimation results using the **GT** dataset, where the occupant was moving much slower and the estimate follows the true position much more closely.

Our motion model does not incorporate velocity or direction of movement, thus, the cloud of particles tends to expand radially outwards, instead of following the true position. An improvement to the algorithm would track the occupant's intended direction, in addition

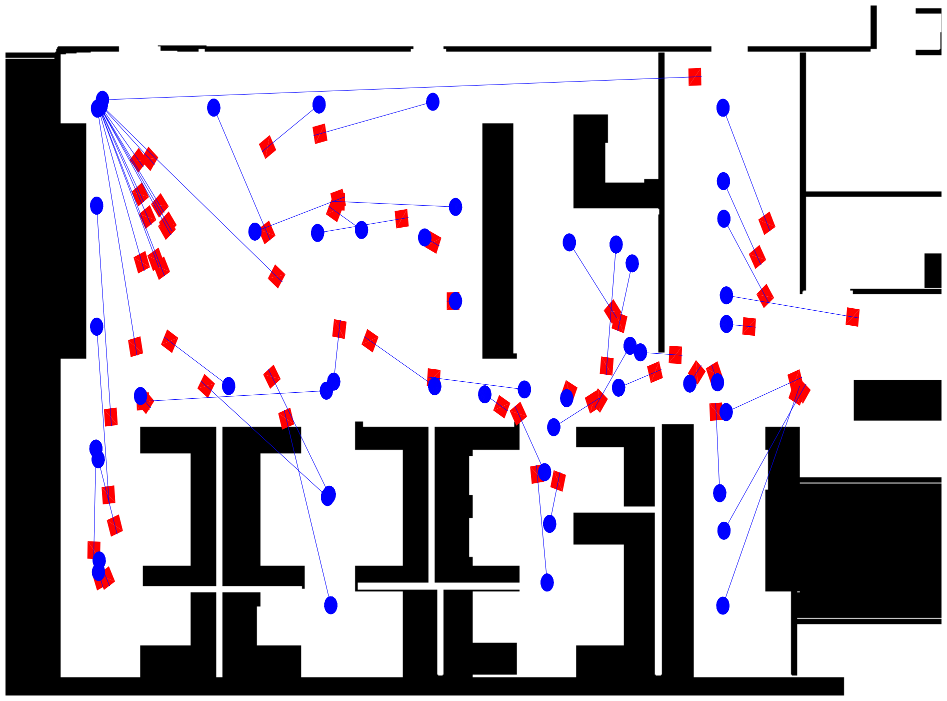


Figure 6.11: Plot of actual coordinates (blue circles) and estimated coordinates (red squares) during **VER** experiment.

to their position, and favor estimates in that direction. This would likely correct for the lag, but would require a finer time resolution, to avoid overshooting if the occupant suddenly changes direction.

Computational Resources

Our eventual goal is to provide real-time IPS services simultaneously for many occupants, therefore we evaluate the computational load of the algorithm. We use the **GT** data set, which contains the most data points of the three, in order to reduce the impact of the time it takes the program to load data into cache for each test. For each test, an average over 24 program executions is taken. Our testing CPU is an Intel Core i7-3720QM at 2.6GHz, containing 4 cores and 8 logical processors. The system contains 24GB total of physical RAM and runs Windows 7 Professional 64-bit.

For the default parameter values in Table 6.1, the average Memory Usage is 7.9 MB, and the average Execution Time is 129 ms. This corresponds to about 3.7 thousand time steps processed per second, or 5 real-time hours processed per execution second.

We are also interested in how three of the parameters, δt, C, and n, effect the resource needs. Each parameter is swept across values which highlight their effect.

In Figure 6.13, we show that both the memory footprint and execution time increases linearly, $\mathcal{O}(n)$, with n, the number of particles used. This follows intuitively since the

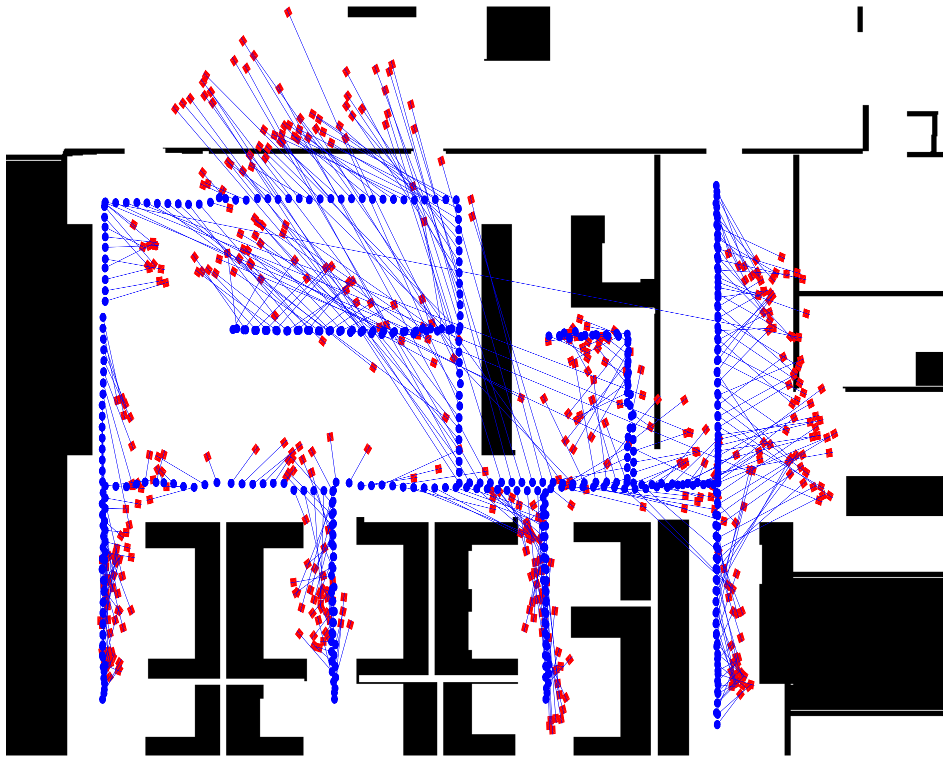


Figure 6.12: Plot of actual coordinates (blue circles) and estimated coordinates (red squares) during **GT** experiment.

prediction step and evaluation of $\Pr(y|x)$ need to be performed once per particle. The discrete resampling phase of the update step requires sorting the particles, so we expect to see linearithmic, $\mathcal{O}(n \log n)$, increase of execution time for very large n.

In Figure 6.14, reducing the cell width dramatically increases the computational requirements of the algorithm. This is because the number of cells in the reachability LUT is proportional to C^{-2}. Additionally, the the number of cells reachable from any other cells is proportional to $C^{-2}v$. Therefore, we expect to see a relation of $\mathcal{O}(C^{-4})$ between cell size and memory usage. The execution time follows this trend as well, since loading a large LUT is time-consuming in relation to executing the rest of the program. We empirically find the relationship plotted in Figure 6.14 to be $\mathcal{O}(C^{-3.7})$.

Figure 6.15 gives the relation between the time interval and resources needed. The amount of time steps for a given real-time interval increases proportionally to $(\delta t)^{-1}$, therefore, we expect to see an $\mathcal{O}((\delta t)^{-1})$ effect on the computation time. Empirically, we find that there is an $\mathcal{O}((\delta t)^{-0.85})$ relationship. The memory usage also follows this trend because we interpolate the sensor measurements into a timeseries with samples spaced at δt as a preprocessing step. If interpolation is done on-line, the memory needs will be reduced

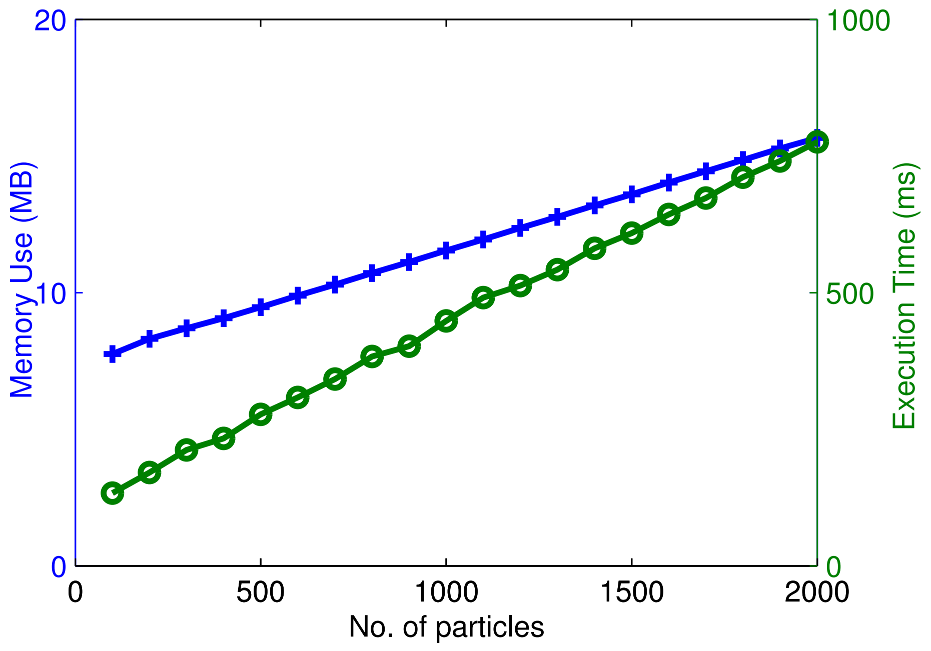


Figure 6.13: Plot of the resources used as a function of the number of particle estimates, n. Memory usage given by + markers and execution time given by ∘ markers.

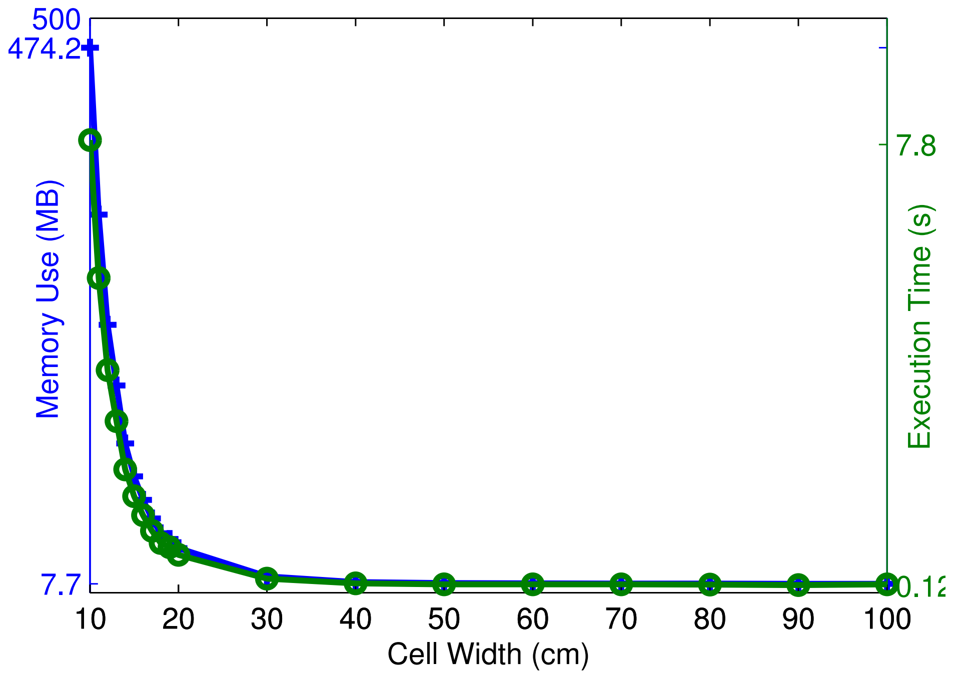


Figure 6.14: Plot of the resources used as a function of the reachability LUT cell width, C. Memory usage given by + markers and execution time given by ∘ markers.

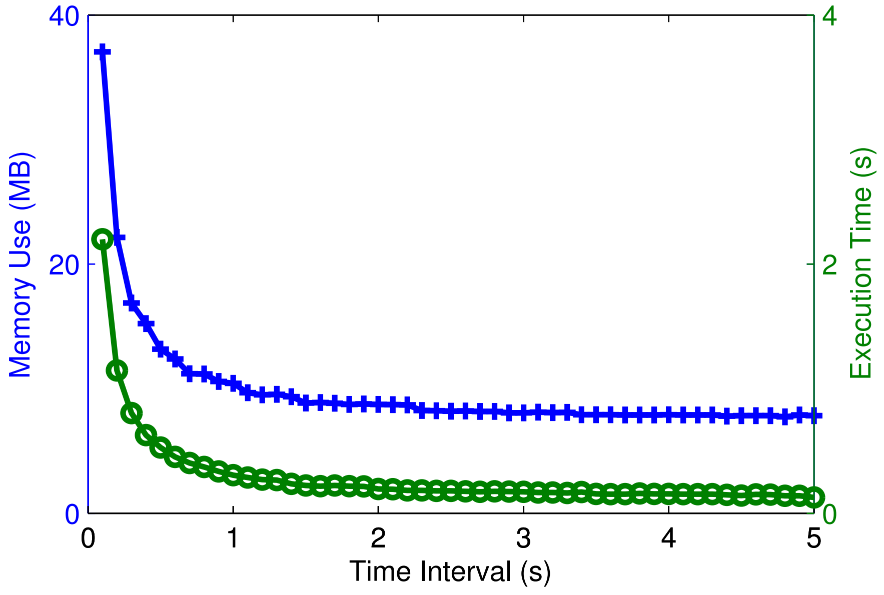


Figure 6.15: Plot of the resources used as a function of the time interval, δt. Memory usage given by $+$ markers and execution time given by $\circ$ markers.

significantly.

Finally, the benefit of multithreading is examined by changing the number of worker threads used by the program to assist in the prediction and update steps. The results displayed in Figure 6.16, show that there is a significant benefit to execution time if threads are used at all, and that 5 threads is the optimal value for this particular 4-core computer. Note that a number of threads *greater than* the number of cores is optimal since at times threads may be blocked waiting for memory accesses, and having an extra thread to occupy the CPU during this time can actually improve processor throughput. However, having too many threads, and the associated overhead, can quickly degrade throughput.

The results achieved in this section lead us to believe our algorithm is well-suited for broader deployment covering more area and occupants. There is also room for more complex motion and sensor models to be incorporated and continue to achieve real-time performance.

6.5 Future directions and extensions

State augmentations

Motivation for augmenting the state

Currently, the state of the particle filter tracks the 2-dimensional position of the occupant on the office floor and also tracks the attenuation factor. A future direction is to also augment the state to estimate additional information about the occupant. This will benefit

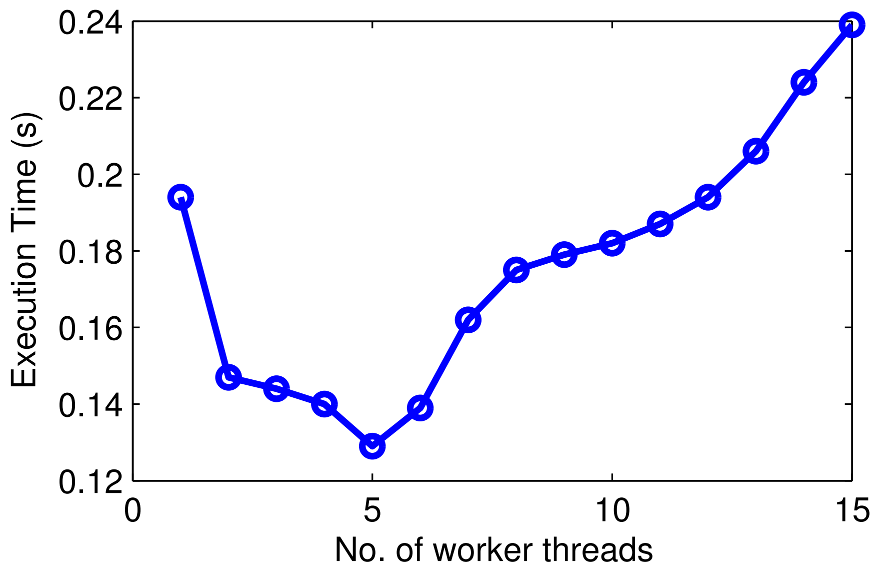


Figure 6.16: Plot of execution time as a function of the number of the number of worker thread.

the particle filtering technique if this information allows more confident estimates of future states. However, since we are adding more dimensions to the state, this also adds more avenues for the particle filter to produce the wrong estimate, especially if the extra degrees of freedom are not constrained by real sensor measurements. Therefore, it is important to study the affects of augmenting the state to ensure that the estimation is actually improved. Below, we describe two augmentations that are interesting for future study: Tracking the velocity of the occupant, and tracking the target location of the occupant.

Tracking velocity

A natural extension to the particle filter would be to also track the velocity of the occupant in addition to their position. This implementation tracks the velocity directly as

$$x_{t,4:5} = \dot{x}_{t,1:2},$$

which is computed by finite-differences from the last state. Equivalently, one could augment the state with the last state directly

$$x_{t,4:5} = x_{t-1,1:2},$$

or, if it is assumed that occupants have a nearly-constant walking speed, one could track only the direction the occupant is currently moving

$$x_{t,4} = \theta,$$

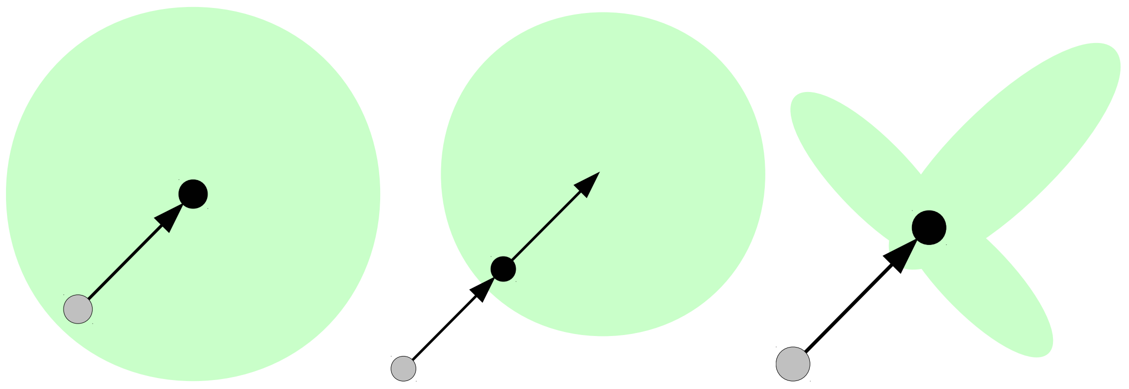

(a) Velocity Ignored (b) Translated uniform dist. (c) Designed distribution

Figure 6.17: Transition distribution strategies given occupant velocity. Previous position is shown by a grey circle, current position is shown by a black circle, and the depiction of the transition distribution (i.e. where an occupant can move to) is shown in green.

where θ is the angle from the point $x_{t-1,1:2}$ to $x_{t,1:2}$.

Once velocity information (or equivalently, the direction or last position of the occupant) is tracked, we can incorporate this information into the state transition distribution for position, $\Pr(x_{t,1:2}|x_{t-1,1:2})$. Figure 6.17 illustrates possibilities of how the state transition could be constructed, assuming no nearby obstacles. The current implementation is depicted by Figure 6.17a, where no velocity information is incorporated and the distribution is a uniform distribution around the current position.

A simple way of integrating velocity would be to simply translate this uniform distribution by the same movement that the occupant made in the last time step, as is shown by Figure 6.17b. This essentially predicts that the occupant will continue moving in the same direction, such as if they are walking down a corridor. There still remains some uncertainty about the occupant's movement which is modelled by continuing to use a uniform distribution. However, this model erroneously gives the possibility of the occupant moving farther than $v \cdot \delta t$ in one time step.

A more intelligent construction of the state transition distribution is illustrated by Figure 6.17c, where the possible next locations are a subset of those in Figure 6.17a (and therefore it maintains that the occupant cannot move farther than $v \cdot \delta t$). The next locations are specified as those where the occupant continues walking directly along the current heading or where the occupant makes a 90-degree turn before moving forward. Since the passageways in offices are often laid out rectilinearly, this places a higher emphasis on occupant paths which include 90-degree turns. As seen by the figure, there is still some uncertainty represented by the wideness of the green areas.

Tracking desired areas

We can also utilize the logical division of a building into spaces (such as rooms, desks, and cubicles) to add intuitive information into the tracking algorithm. A useful addition to the state would be a prediction of the desired location that an occupant would want to go. For example, if we determine that an occupant is thirsty, then there is a higher probability that the occupant will follow a path leading to the water cooler. Although we do not yet have sensors to directly sense a occupant's intention, information such as historical behavior, time of day, and current direction provide clues.

A corollary of this augmentation is to gather and track knowledge about where an occupant will *not* travel to, such as a janitor's closet, or the opposite-gendered restroom. Depending on how many areas are removed from consideration, this can significantly reduce the size of the state space and consequently, possible errors in the estimation. In addition to removing these specific areas from consideration, we can also remove any paths which lead only to non-considered areas, since the occupant will not walk along these paths as well.

Information from fixed infrastructure

CO_2 concentration

In Section 5.3, we describe a study carried out to model the effect of occupants on the CO_2 concentration within a room and built an estimator and identifier for the model. Given CO_2 measurements at the supply and return vents in an enclosed room (such as a conference room), we demonstrate a technique to estimate the rate of CO_2 being generated by occupants. Since this value is heavily dependent on occupants, it would be appropriate to augment the observation distribution accordingly. For instance, if we detect that very little CO_2 is being generated in a room, this indicates very low probability that any tracked occupants are inside of this room. A further improvement is to augment the state of positioning system to include the metabolic rate of the tracked occupant, thus, the actual value of CO_2 production may be able to identify individual occupants inside a CO_2-monitored room. The metabolic rate of the occupant is determined when the occupant is known to be the only occupant inside a CO_2-monitored room (e.g. their own office with no visitors).

Coarse Particulate Matter

Like CO_2, coarse particulate matter (PM) (i.e. airborne particles with diameter $\geq 2\,\mu m$) is a proxy to local occupancy (See Section 5.2), thus, we will be able to use similar methods as discussed for CO_2. The main difference is that human occupants are the primary producers of CO_2 in office buildings, whereas many factors can influence coarse PM concentrations. For instance, vacuuming frequency and floor type (e.g. carpet or tile) affects the amount of particles latent on the floor to be re-suspended. Therefore, given a detected spike in coarse PM from nearby occupant activity, there is still much uncertainty as to the number of occupants nearby, and very clean floors can result in especially weak and uncertain detection events. Fortunately, the particle filter is well adapted to accept uncertain sensor data, as long as the sensor noise is well modelled. In our implementation, coarse PM can play a role indicating whether an occupant is at their desk, i.e. if the sensor is mounted at foot-level near their desk. The results from Section 5.2 show that a coarse PM sensor is useful as a rudimentary threshold sensor, detecting when occupants walk past it. These detections could indicate higher probabilities of trajectories which estimate that the occupant walked past the sensor.

Light-based detection

Passive Infrared (PIR) sensing is a popular technology used by security systems, since they reliably detect human occupant movement within a specified cone of detection. The main limitation of PIR sensing for indoor positioning purposes is that the sensor cannot identify occupants, nor can it distinguish between one or multiple occupants. Therefore, PIR sensing alone cannot be easily used for positioning, but nonetheless is very valuable for narrowing down the feasible state space. That is, if the PIR does *not* detect an occupant, then the particle filter can remove any estimates falling within its cone of detection. Thus, a rich network of environmental sensor devices (See Chapter 4) carrying PIR devices allows the particle filter to focus the particle estimates on only the occupied areas of the building.

Infrared light is also used for reliable threshold detectors. Often seen in businesses to detect customers entering and exiting the storefront, these sensors utilize a beam of light from a laser or focused infrared diode and a corresponding light sensor. The beam of light is directed across the detection area, such as a doorway, possibly reflected off of a mirror, and finally reaches the sensor. An occupant is detected when they pass through the beam, blocking it from reaching the sensor. Like the PM sensor, detection events provided by this sensor can indicate a higher probability of trajectories which include crossing the beam of light.

Contact-based detection

Switches are simple, reliable, and low-power devices which can immediately detect state changes of objects like doors, windows, and drawers. Using switches connected to the expansion port of environmental sensor devices (See Chapter 4), we can continuously monitor the state of doors, which, when closed, represents an obstacle in the state-transition distribution. Switches can also be mounted on windows and drawers to detect state changes which provide, with high-confidence, a small area (i.e. within arms-length) of possible locations of an occupant.

Information from occupant-carried infrastructure

Wearable electronics, such as the Pebble [130] and eWatch [131], are becoming more ubiquitous and carrying more sensors, owing to advances in miniaturization and manufacturing.

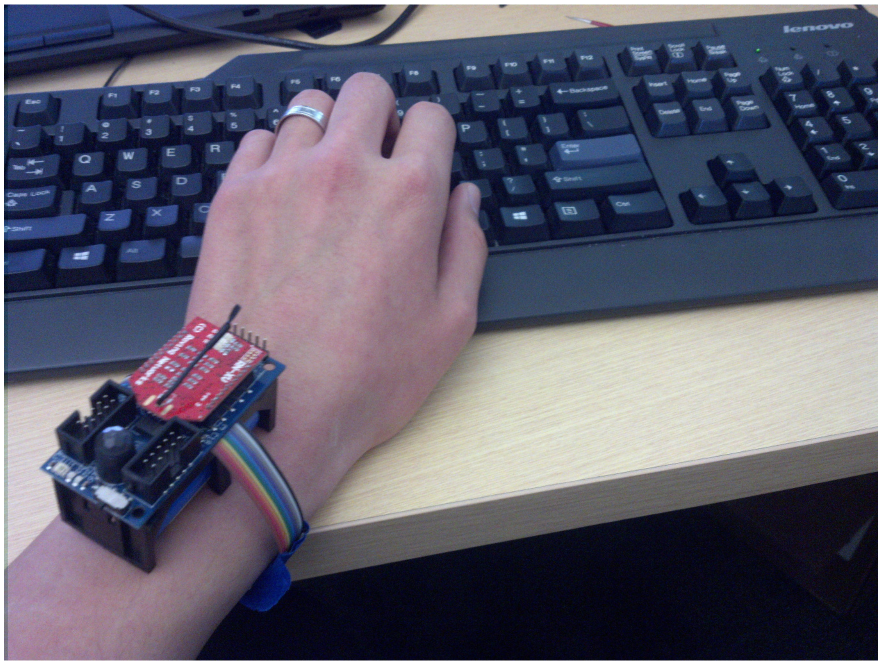
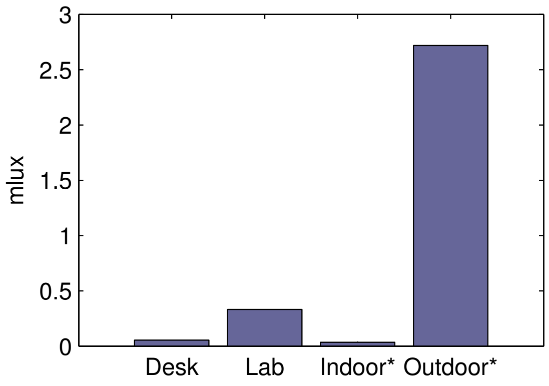


Figure 6.18: Occupant-carried environmental sensing watch

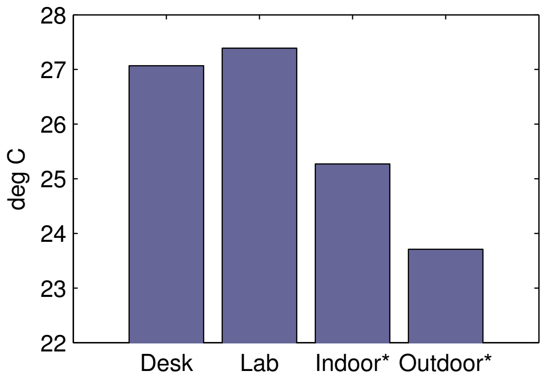
Currently, technology is at the point where these smart watches can carry a suite of environmental sensors and continuously monitor the wearer's local environment. In the near future, manufacturing techniques will enable the vision of *Smart dust* [132]: a fully-integrated cubic-millimeter size wireless sensor which could be non-intrusively integrated into jewelry, clothes, and implants. In Figure 6.18, we show how the environmental sensing platform (See Chapter 4) can be adapted into a smart watch to conduct wearable sensor studies.

The measurements collected from occupant-carried instruments is studied extensively. Classifiers are one way of associating *features*, i.e. heuristics calculated from the measurements, to activities that the occupant is performing [133]. For example, a high magnitude of acceleration could indicate that the wearer is running. Particle filters (a type of Bayesian classifier) demonstrate good performance at the activity recognition task [131, 134]. In addition, one study [131] demonstrates detection of coarse location from a pre-defined set (e.g. apartment, bus, lab, office, restaurant, street, and supermarket). In our implementation, we will attempt to achieve a finer location resolution, as well as use statistical distributions relating occupant positions to features, such that we do not need to pre-define distinct locations.

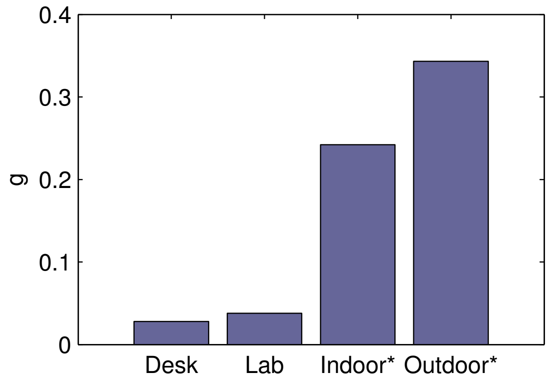
As a motivating experiment, we use the environmental sensing watch to measure 3-axis acceleration, light level, temperature, and humidity of the occupant's wrist while the occupant is in four different locations. We plot the features in Figure 6.19. The first two locations are at a desktop, and the occupant is typing (as in Figure 6.18). Both locations are near a window, however, the window near the "Desk" location is shaded, whereas the



(a) Light level



(b) Temperature



(c) Std. deviation of Acceleration

Figure 6.19: Features measured by the environmental sensing watch in different scenarios. *The indoor and outdoor tests were measured while occupant was walking normally.

window near the "Lab" location does not have shades. At the second two locations the occupant is walking. The indoor location is inside a windowless hallway, and the outdoor location is on an outdoor sidewalk.

Interpretation of light level

From Figure 6.19a, we can immediately recognize when the occupant is outside, as sunlight is much brighter than indoor light, even if there is shade or clouds. Additionally, indoor areas which receive more sunlight from windows, such as the lab, are distinguishable from areas such as the desk and indoor hallway which where the sunlight is occluded.

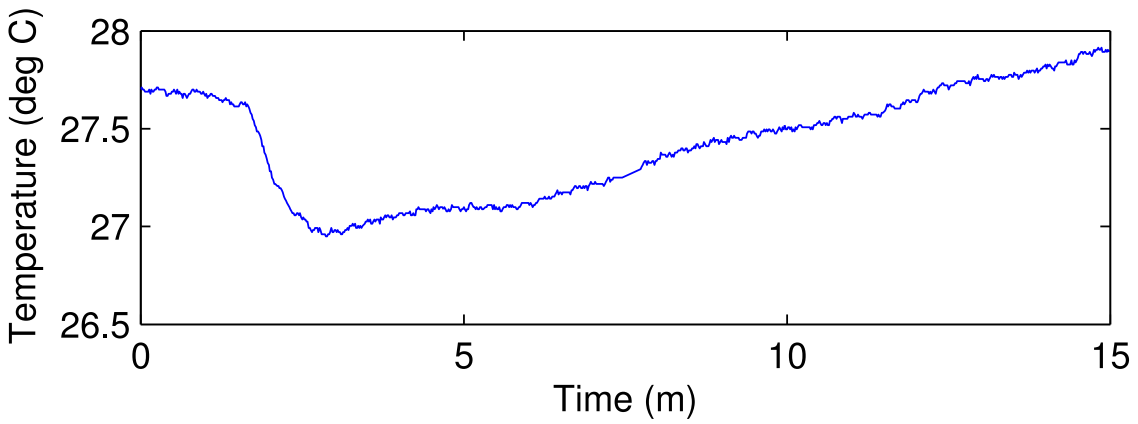


Figure 6.20: Cooling effect induced by walking. Between minute 1 and minute 3, the occupant is walking from the desk to the lab location. After minute 3, the occupant is seated at the lab location

Interpretation of temperature

The interpretation of temperature is more complicated since the occupant's metabolic heat generation influences the temperature reading of a wearable sensor. This is demonstrated by the results in Figure 6.19b, where the temperature reading is noticeably lower when the occupant is walking. We believe this is due to a cooling effect due to increased airflow over the watch while walking (the occupant walks with a normal gait, including swinging their arms). When the occupant is stationary, their body heat forms a warm boundary layer surrounding themselves which read by the temperature sensor. Evidence of this effect is seen by Figure 6.20, which plots a time series of temperature as the occupant walks from the desk to the lab. While walking, the temperature reading drops by approximately 1.5 °C, and then rises after the occupant sits and remains stationary at the lab.

Interpretation of acceleration

From the wristwatch experiment, we are clearly able to distinguish between walking and stationary states of the occupant based on acceleration measurements. Our chosen feature is the standard deviation of the magnitude of the 3-dimensional acceleration vector. That is, given N 3-axis measurements of acceleration: $a_x[i]$, $a_y[i]$, $a_z[i]$, and $i \in \{1, 2, \ldots, N\}$, we calculate

$$\begin{aligned}\sigma &= \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\bar{a}[i]-\mu\right)^2},\\ \mu &= \frac{1}{N}\sum_{i=1}^{N}\bar{a}[i],\\ \bar{a}[i] &= \sqrt{\left(a_x[i]\right)^2+\left(a_y[i]\right)^2+\left(a_z[i]\right)^2}.\end{aligned}$$

We show the values of σ for the four tested locations in Figure 6.19c. As expected, in the latter two locations, when the occupant is walking, the value of σ is significantly higher than the first two locations, when the occupant is stationary. This is likely due to the acceleration sensor measuring the swinging of the arms as the occupant walks. We believe that detecting whether the occupant is stationary or moving will improve the practical performance of the particle filter, especially in the office, where occupants frequently stop and start moving between places.

We have shown via a simple experiment that accelerometer measurements can provide information useful for positioning. In future studies with occupant-carried acceleration sensors, we will explore two additional manners in which the measurements can be used: by detecting orientation and by detecting footsteps.

Orientation is relatively straightforward to obtain from filtered 3-axis acceleration measurements simply by finding the relative direction of the gravity vector. We believe that orientation of occupant-carried devices can provide valuable information towards determining the state of the occupant, given the context of the sensor placement. For instance, if such a device is placed in a pocket, this can indicate whether an occupant is sitting or standing. For the positioning problem, detecting that an occupant is sitting indicates that the occupant is stationary, and probably located in a part of the office containing chairs. A sensor mounted in a wristwatch similarly measures whether an occupant is doing desk work or their hands are at their sides.

A popular and intuitive direction to integrate measurements from occupant-carried accelerometers is to detect footsteps which can provide information about immediate motions that the occupant is making [135, 136, 137]. For example, knowing the number of footsteps an occupant takes in a given time period constrains the distance they have moved in that period given an estimate of the occupant's stride length. It is also possible to distinguish between types of footsteps, such as walking, running and climbing stairs. This is powerful if a map of the building is available, for instance, if an occupant is known to be climbing stairs, then only trajectories which include a stairway will be considered. In some cases, if gyroscopes and accelerometers are mounted on both feet, an Extended Kalman Filter can be used to estimate the displacement of each foot between steps [138], further narrowing down possible locations of the occupant.

Instead of being mounted on the feet and estimating positional displacements, our method would only attempt to detect footsteps by looking for spikes in the signal when the occupant's foot strikes the ground. Therefore, the accelerometer can be mounted anywhere on the body of the occupant as long as there is a fairly rigid physical connection to the foot. Our hypothesis is that using accelerometer data from a smartphone carried in the occupant's pocket will be sufficient. If the occupant is carrying an environmental sensor (See Chapter 4), then the accelerometer readings from this sensor can be used as well. We would program either the phone or environmental sensor to report the number of footsteps f_t counted every second. To integrate into the particle filter, we would adjust the state transition distribution to account for an adaptive travel distance per time step, instead of statically limiting it to $v \cdot \delta t$ (where v is the occupant's walking speed and δt is the duration of one time step). Given the current number of footsteps per second f_t, we can instead constrain the occupant's travel distance to $f_t \cdot s \cdot \delta t$, where s is occupant's stride length. Since the parameter s is difficult to identify, The most information will be gained when $f_t = 0$, since this indicates that the occupant is stationary (travel distance is equal to 0).

Coupling to occupancy estimation

As we demonstrate in Section 6.4, our implementation has the benefit of processing data at 5 real-time hours per second and requiring only 7.9 MB. Therefore, it is feasible to build a composite particle filter which can track hundreds of occupants, in order to track every occupant in a building. At this point, we will then be able to calculate an estimate of occupancy for each space in the building by simply counting the number of tracked occupants in each room. This is a typical application of IPSs, given the advantages of knowing occupancy within a building (See Chapter 5).

This counting technique is a one-way transfer of information from the occupant tracking problem to the occupancy estimation problem, however, our next direction will be to explore the bi-directional sharing of information between the two problems. Our method is to use the particle filter framework to create a joint estimator that tracks the occupancy of rooms along with the position of occupants by augmenting the state. This allows us to address the relationship between the two problems directly in the same framework via the state transition distribution. For example, we can add a rule which explicitly specifies that a room's occupancy level is equal to the sum of the tracked occupants within it. This rule implicitly defines a bi-directional dependency between the occupant tracking and occupancy estimation results, since the particle filter will essentially estimate both simultaneously to reach a feasible solution. The flow of information from the occupancy estimation problem to the occupant tracking problem is especially interesting since we have identified new and informative models of indoor phenomena like CO_2 and particulate matter.

In building this new framework, we foresee several complications that will require special treatment. For instance, it will be difficult to model visitors who are unknown and not tracked by the system, yet still affect the true occupancy of the space. We will need to allow some uncertainty in the coupling between the occupancy estimation and occupant tracking problems. Another complication from building this composite estimator is that it will create a very large and high-dimensional state space. Particle filters are some of the best techniques for large state spaces, due to their scalability, however, there is still the possibility of the particle estimates getting "lost" or "trapped" in parts of the state space. With limited sensing, it is also more difficult to distinguish improbable states and narrow down an estimate when the state space is large. We hope to address these problems by providing enough information through a multitude of sensing technologies and intuitions about occupant behavior. This information serves to narrow down the feasible areas of the state space.

6.6 Conclusions

In this chapter, we have described an occupant-tracking system using particle filters that achieves an average of room-level accuracy when deployed in a real office environment. Moreover, by construction, our method does not position occupants inside inaccessible areas, which may lead to better estimates of the room that an occupant is inside. The particle filter method is also dynamic, meaning that the estimation will improve over time as more information is gathered. This results in slower-moving or stationary occupants resulting in very good predictions, such as in Figure 6.12.

If we assume all occupants are participating by carrying the RFID tag required by our IPS, then our method is likely to be accurate enough to estimate occupancy of the spaces in the office. However, we understand that this assumption is invalid in the case where there are visitors, or when participants forget to carry the tag. A popular answer to this problem is to use WiFi-based technology that can sense any WiFi-enabled smartphone carried by occupants since the occupants have other incentives to carry their phone with them. There are drawbacks to this as well, since occupants might not own a WiFi-enabled smartphone, or have the WiFi-disabled, or be carrying more than one WiFi-enabled device, such as a laptop. An additional drawback is that there is heterogeneity in the radio transmission characteristics of WiFi devices due to differences in the antennas, transceivers, and the way they are used. Therefore, relying on a characterization of signal strength can fail when these differences are not accounted for.

Therefore, as we describe in Section 6.5, our vision is that a hybrid infrastructure can be built which simultaneously estimates occupancy and tracks many or all occupants. The information from both problems would be elegantly integrated together by the particle filtering framework, which sees the building's occupancy and position of its occupants as components of a encompassing state variable. Although the state space will consequently be very large, particle filters are well suited to estimate within these spaces, due to the Monte-Carlo approach. Particle filters also allow us to leverage the peculiarities of each type of sensor to guide the particle filter estimates. Our studies into CO_2 and particulate matter concentrations in Chapter 5 are examples of experiments we have conducted to understand how these environmental variables are affected by occupants, and the results of these experiments can produce the parameters and models needed to integrate the readings into the particle filtering framework.

Chapter 7

Conclusions and Future Work

7.1 Contributions and status of the work

Applying estimation techniques towards real-world problems and using practical sensing infrastructures has particular challenges when facing mobile environments. However, these challenges can be overcome when mathematical and technological tools are properly chosen and applied towards the estimation problem. In this dissertation, we have described several mathematical estimation tools which were successfully adapted and applied towards a real-world scenario: Ensemble Kalman Filtering was used to estimate river currents given measurements from Lagrangian sensors. PDE estimators were used to estimate the amount of CO_2 generated by humans in a conference room. Particle filters were used to estimate the position of an occupant with radio signal strength readings. We have also described two sensing architectures which provide the measurement inputs necessary for the estimation tools: the Floating Sensor Network and the environmental sensing platform.

The contribution of the Floating Sensor Network is to enable more types of Lagrangian river studies via the development of two novel drifter designs: the passive Android drifter and the active Generation 3 drifter. Due to the low manufacturing cost of the units (around \$300), the Android drifter allows scientists to perform studies with a highly dense observation of the river state, and satisfy cost constraints. As well, the real-time remote reporting ability allows scientists to find and retrieve the units much easier than if the drifters only logged their measurements locally. The motorized Generation 3 drifter design is also easily mass-produced and has the unique ability to maneuver itself to avoid shorelines using an advanced Hamilton-Jacobi safety control technique that we developed. This drifter gives scientists the ability to conduct drifter studies in many parts of the river which cannot be otherwise sensed, due to factors such as wind, pushing passive drifters into the shorelines. Currently, we lend the Android drifter fleet to other environmental research groups within Stanford, the University of California Davis, and the US Geographical Survey.

We have also contributed an architecture for the sensing of mobile occupants in a building. A sensor device was designed which uses WiFi for robust data delivery and is also low-power enough to last for over 5 years on a battery. At the same time, the sensor is able to collect a rich set of environmental variables from its suite of on-board instruments, and is expandable to take measurements from nearly any other instrument, such as a CO_2 sensor. The device is also small enough to be carried by occupants or be deployed in size-constrained locations. The long battery life and compatibility with existing WiFi infrastructure result in a low-cost of deployment and maintenance. The environmental sensing platform is under active development and deployment. Currently, we have produced 100 of the "Version 3" environmental sensor devices for deployment at UC Berkeley's Center for Research in Energy Systems Transformation (CREST) laboratory and the Singapore Berkeley Building Efficiency and Sustainability in the Tropics (SinBerBEST) headquarters at the Campus for Research Excellence and Technological Enterprise (CREATE) tower in Singapore. There are eventual plans for inserting the measurements into a cluster-computing distributed database with nodes in both Singapore and Berkeley. After this, the measurements will be available alongside data from the buildings' lighting, HVAC, and electrical systems.

7.2 Future applications

We envision many avenues of extension to the work described in this dissertation. During our development of the active drifters, we explored the idea of a depth-profiling drifter, which used a buoyancy control system allowing it to dive underwater. This capability would not only allow 3-dimensional drifter studies, but also allow the drifters to leverage underwater currents to carry them to a desired destination. We have have also prototyped Android-based active drifters which have the capabilities of the active Generation 3 drifter, but are significantly more economical due to using off-the-shelf Android smartphones.

Our experiences in developing the Hamilton-Jacobi safety control technique can be applied to other mobile environments which have mobility constraints. We imagine that meteorological sensing could be augmented with underactuated gliders, kites, and balloons which have limited control authority, but can navigate enough to ride global wind currents to their destination. Minimum time-to-reach functions, calculated as an intermediate step of the safety control method, are also very powerful and can be used for other control algorithms. For instance, we have explored finding Zermelo-Voronoi partitions [139] for distributed control of drifter fleets. These partitions can be generated by finding the intersections of minimum time-to-reach functions emanating from each drifter in the fleet.

There is also significant interest by other researchers into the progression of the environmental platform due to its suite of sensors supporting indoor climate and energy efficient control studies. One project is to create a "building in a suitcase" which includes several of the sensor devices, an access point, and a gateway PC with a mobile Internet connection. With this, scientists or building contractors could rapidly deploy a functional environmental sensor network to survey a building site.

Finally, we envision that the particle filtering framework will inspire future research in this direction, as discussed in Section 6.5. An immediate goal is to extend the particle filter to perform occupancy estimation as well as occupant tracking. In the further future, we are especially excited about the flexibility in the types of measurements that the particle filter can accept. This is because future smart buildings will undoubtedly be outfitted with a plethora of sensors, such as temperature, light, occupancy (passive infrared and ultrasound), and plugload meters (i.e. devices which measure power consumption of appliances). Our tracking framework could theoretically be installed on such a smart building and provide occupant tracking and occupancy estimation services without requiring any additional dedicated infrastructure. Coupled with occupancy-aware control algorithms, this is clear advantage for future smart buildings to save energy without incurring prohibitive installation costs.

More generally, this dissertation exhibits solutions to real-world estimation challenges. In many cases, these solutions are not specific to the setting for which they were designed. For instance, the environmental sensing architecture was extended from, and shared significant portions of source code with, the Floating Sensor Network. Essentially we are building a library of algorithms and tools (both hardware and software) for future studies in mobile environments. Our goal is to give other researchers a head-start into their own experiments, either by adapting the techniques developed by us, or finding inspiration in the design choices we have made while designing their own platforms.

Bibliography

[1] P. Samar and S. B. Wicker. “On the behavior of communication links of a node in a multi-hop mobile environment”. In: *5th ACM international symposium on mobile ad hoc networking and computing*. ACM. 2004, pp. 145–156.

[2] P. Bergamo and G. Mazzini. “Localization in sensor networks with fading and mobility”. In: *13th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications*. Vol. 2. IEEE. 2002, pp. 750–754.

[3] G. Evensen. “Sequential data assimilation with a nonlinear quasi-geostrophic model using Monte Carlo methods to forecast error statistics”. In: *Journal of Geophysical Research* 99.C5 (1994), pp. 10143–10162. DOI: `10.1029/94JC00572`.

[4] M. Krstic and A. Smyshlyaev. *Boundary control of PDEs: A course on backstepping designs*. Vol. 16. SIAM, 2008.

[5] N. J. Gordon, D. J. Salmond, and A. F. Smith. “Novel approach to nonlinear/non-Gaussian Bayesian state estimation”. In: *IEE Proceedings F (Radar and Signal Processing)*. Vol. 140. 2. IET. 1993, pp. 107–113.

[6] I. H. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2005.

[7] *DWR Drops State Water Project Allocation to Zero, Seeks to Preserve Remaining Supplies*. Jan. 31, 2014. URL: `http://www.water.ca.gov/news/newsreleases/2014/013114pressrelease.pdf`.

[8] *Floating sensor network*. 2010. URL: `http://float.berkeley.edu/`.

[9] T. Conomos, R. Smith, and J. Gartner. “Environmental setting of San Francisco Bay”. In: *Hydrobiologia* 129.1 (1985), pp. 1–12.

[10] J. C. Swallow. “A neutral-buoyancy float for measuring deep currents”. In: *Deep Sea Research* 3.1 (1955), pp. 74–81. DOI: `doi:10.1016/0146-6313(55)90037-X`.

[11] D. D. Clark. “Overview of the Argos system”. In: *OCEANS ’89. Proceedings*. Vol. 3. Sept. 1989, pp. 934–939. DOI: `10.1109/OCEANS.1989.586711`.

[12] R. E. Davis. “Drifter observations of coastal surface currents during CODE: The method and descriptive view”. In: *Journal of Geophysical Research* 90.C3 (1985), pp. 4741–4755. DOI: `10.1029/JC090iC03p04741`.

[13] D. S. Bitterman and D. V. Hansen. "The design of a low cost tropical drifter buoy". In: *Marine Data Systems International Symposium (MDS).* 1986, pp. 575–581.

[14] P. P. Niiler, R. E. Davis, and H. J. White. "Water-following characteristics of a mixed layer drifter". In: *Deep Sea Research Part A. Oceanographic Research Papers* 34.11 (1987), pp. 1867–1881. DOI: `10.1016/0198-0149(87)90060-4`.

[15] A. Tinka, M. Rafiee, and A. M. Bayen. "Floating sensor networks for river studies". In: *IEEE Systems Journal* (2012). to appear; accepted Nov. 2011.

[16] J. C. Perez et al. "Development of a cheap, GPS-based, radio-tracked, surface drifter for closed shallow-water bays". In: *IEEE/OES 7th Working Conference on Current Measurement Technology.* Mar. 2003, pp. 66–69. DOI: `10.1109/CCM.2003.1194285`.

[17] J. Austin and S. Atkinson. "The design and testing of small, low-cost GPS-tracked surface drifters". In: *Estuaries* 27.6 (Dec. 2004), pp. 1026–1029. DOI: `10.1007/BF02803428`.

[18] J. Beard et al. "Mobile phone based drifting lagrangian flow sensors". In: *IEEE 3rd International Conference on Networked Embedded Systems for Every Application (NESEA).* IEEE. 2012, pp. 1–7.

[19] *Android Developers.* Google, Inc. 2014. URL: `http://developer.android.com/`.

[20] K. Weekly et al. "Autonomous river navigation using the Hamilton-Jacobi framework for underactuated vehicles". In: *IEEE Transactions on Robotics* (2014). in review.

[21] C. Oroza et al. "Design of a network of robotic Lagrangian sensors for shallow water environments with case studies for multiple applications". In: *Journal of Mechanical Engineering Science* (2012). in review.

[22] Google. *Protocol Buffers: Google's Data Interchange Format.* URL: `https://developers.google.com/protocol-buffers/`.

[23] K. Ang, G. Chong, and Y. Li. "PID control system analysis, design, and technology". In: *IEEE Transactions on Control Systems Technology* 13.4 (2005), pp. 559–576.

[24] P. J. Van Leeuwen and G. Evensen. "Data assimilation and inverse methods in terms of a probabilistic formulation". In: *Monthly Weather Review* 124.12 (1996), pp. 2898–2913.

[25] P. L. Houtekamer and H. L. Mitchell. "Data assimilation using an ensemble Kalman filter technique". In: *Monthly Weather Review* 126.3 (1998), pp. 796–811. DOI: `10.1175/1520-0493(1998)126<0796:DAUAEK>2.0.CO;2`.

[26] P. L. Houtekamer and H. L. Mitchell. "A sequential ensemble Kalman filter for atmospheric data assimilation". In: *Monthly Weather Review* 129.1 (2001), pp. 123–137.

[27] T. M. Hamill and C. Snyder. "A hybrid Ensemble Kalman Filter–3D variational analysis scheme." In: *Monthly Weather Review* 128.8 (2000).

[28] C. L. Keppenne and M. M. Rienecker. “Initial testing of a massively parallel ensemble Kalman filter with the Poseidon isopycnal ocean general circulation model.” In: *Monthly Weather Review* 130.12 (2002).

[29] J. P. Kaipio and E. Somersalo. *Statistical and computational inverse problems*. Vol. 160. Springer, 2005.

[30] J. M. Huttunen and J. P. Kaipio. “Approximation error analysis in nonlinear state estimation with an application to state-space identification”. In: *Inverse Problems* 23.5 (2007), p. 2141. DOI: 10.1088/0266-5611/23/5/019.

[31] J. M. Huttunen and J. P. Kaipio. “Approximation errors in nonstationary inverse problems”. In: *Inverse Problems and Imaging* 1.1 (2007), p. 77. DOI: 10.3934/ipi.2007.1.77.

[32] A. Nissinen, V. P. Kolehmainen, and J. P. Kaipio. “Compensation of modelling errors due to unknown domain boundary in electrical impedance tomography”. In: *IEEE Transactions on Medical Imaging* 30.2 (2011), pp. 231–242. DOI: 10.1109/TMI.2010.2073716.

[33] Q. Wu, X. Litrico, and A. M. Bayen. “Data reconciliation of an open channel flow network using modal decompositions”. In: *Advances in Water Resources* 32.2 (2009), pp. 193–204. DOI: 10.1016/j.advwatres.2008.10.009.

[34] Q. Wu et al. “Variational Lagrangian data assimilation in open channel networks”. In: *Water Resources Research* (2014). in review.

[35] J. Latombe. *Robot motion planning*. Springer Verlag, 1990.

[36] S. LaValle. *Planning algorithms*. Cambridge Univ Pr, 2006.

[37] S. Sundar and Z. Shiller. “Optimal obstacle avoidance based on the Hamilton-Jacobi-Bellman equation”. In: *IEEE transactions on robotics and automation* 13.2 (1997), pp. 305–310.

[38] A. Tinka et al. “Viability-based computation of spatially constrained minimum time trajectories for an autonomous underwater vehicle: implementation and experiments”. In: *American Control Conference*. June 2009.

[39] P. Cardaliaguet, M. Quincampoix, and P. Saint-Pierre. “Set-valued numerical analysis for optimal control and differential games”. In: *Stochastic and Differential Games: Theory and Numerical Methods* 4 (1999), pp. 177–247.

[40] A. Bayen et al. “A differential game formulation of alert levels in ETMS data for high altitude traffic”. In: *Proceedings of the AIAA Guidance, Navigation, and Control Conference*. Aug. 2003.

[41] I. Mitchell, A. Bayen, and C. Tomlin. “A time-dependent Hamilton-Jacobi formulation of reachable sets for continuous dynamic games”. In: *IEEE Transactions on Automatic Control* 50.7 (2005), pp. 947–957.

[42] M. Bardi and I. Capuzzo-Dolcetta. *Optimal control and viscosity solutions of Hamilton-Jacobi-Bellman equations*. Birkhauser Boston, 2008.

[43] J. Sethian. *Level set methods and fast marching methods: evolving interfaces in computational geometry, fluid mechanics, computer vision, and materials science*. 3. Cambridge University Press, 1999.

[44] S. Osher and R. Fedkiw. *Level set methods and dynamic implicit surfaces*. Vol. 153. Springer Verlag, 2003.

[45] I. Mitchell. "A toolbox of level set methods". In: *UBC Department of Computer Science Technical Report* 11 (2007).

[46] R. Bellman. "Dynamic Programming". In: *Princeton University Press* (1957).

[47] R. Isaacs. *Differential Games: A mathematical theory with applications to warfare and pursuit, control and optimization*. 1965.

[48] S. Osher. "A level set formulation for the solution of the Dirichlet problem for Hamilton–Jacobi equations". In: *SIAM Journal on Mathematical Analysis* 24 (1993), p. 1145.

[49] J. W. Thomas. *Numerical partial differential equations: finite difference methods*. Vol. 1. Springer, 1995.

[50] E. Ateljevich et al. "CFD modeling in the San Francisco Bay and Delta". In: *4th SIAM Conference on Mathematics for Industry*. 2009.

[51] J. Lygeros, C. Tomlin, and S. Sastry. "Controllers for reachability specifications for hybrid systems". In: *Automatica-Oxford* 35 (1999), pp. 349–370.

[52] C. Cassandras and S. Lafortune. *Introduction to discrete event systems*. Vol. 11. Kluwer academic publishers, 1999.

[53] L. Pérez-Lombard, J. Ortiz, and C. Pout. "A review on buildings energy consumption information". In: *Energy and buildings* 40.3 (2008), pp. 394–398.

[54] Y. Agarwal et al. "Occupancy-driven energy management for smart building automation". In: *2nd ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Building*. ACM. 2010, pp. 1–6.

[55] C. Chao and J. Hu. "Development of a dual-mode demand control ventilation strategy for indoor air quality control and energy saving". In: *Building and Environment* 39.4 (2004), pp. 385–397.

[56] V. L. Erickson, M. Á. Carreira-Perpiñán, and A. E. Cerpa. "OBSERVE: Occupancy-based system for efficient reduction of HVAC energy". In: *10th International Conference on Information Processing in Sensor Networks (IPSN)*. IEEE. 2011, pp. 258–269.

[57] ZigBee Alliance. *ZigBee specification*. 2006.

[58] T. Watteyne et al. "OpenWSN: a standards-based low-power wireless development environment". In: *Transactions on Emerging Telecommunications Technologies* 23.5 (2012), pp. 480–493.

[59] P. Levis et al. "TinyOS: An operating system for sensor networks". In: *Ambient intelligence*. Springer, 2005, pp. 115–148.

[60] S. Dawson-Haggerty et al. "sMAP: a simple measurement and actuation profile for physical information". In: *8th ACM Conference on Embedded Networked Sensor Systems*. ACM. 2010, pp. 197–210.

[61] *ReadingDB*. 2014. URL: `https://github.com/stevedh/readingdb`.

[62] INCITS, ANSI. "ISO/IEC 13239". In: *Information technology–Telecommunications and information exchange between systems–High-level Data Link Control (HDLC) procedures* (2002).

[63] *Arduino*. 2014. URL: `http://www.arduino.cc/` (visited on 02/16/2014).

[64] P. Dutta et al. "A building block approach to sensornet systems". In: *6th ACM conference on Embedded network sensor systems*. ACM. 2008, pp. 267–280.

[65] J. L. Hill and D. E. Culler. "Mica: A wireless platform for deeply embedded networks". In: *IEEE Micro* 22.6 (2002), pp. 12–24.

[66] J. Polastre, R. Szewczyk, and D. Culler. "Telos: enabling ultra-low power wireless research". In: *Information Processing in Sensor Networks, 2005. IPSN 2005. Fourth International Symposium on*. IEEE. 2005, pp. 364–369.

[67] H. Dubois-Ferrière et al. "TinyNode: a comprehensive platform for wireless sensor network applications". In: *5th international conference on Information processing in sensor networks*. ACM. 2006, pp. 358–365.

[68] M. Ceriotti et al. "Monitoring heritage buildings with wireless sensor networks: the Torre Aquila deployment". In: *International Conference on Information Processing in Sensor Networks*. IEEE Computer Society. 2009, pp. 277–288.

[69] Microchip Technology Inc. *RNXV Datasheet*. 2012. URL: `http://ww1.microchip.com/downloads/en/DeviceDoc/rn-171-xv-ds-v1.04r.pdf` (visited on 10/29/2012).

[70] L. Li et al. "The applications of WiFi-based wireless sensor network in internet of things and smart grid". In: *6th IEEE Conference on Industrial Electronics and Applications (ICIEA)*. IEEE. 2011, pp. 789–793.

[71] Digi International Inc. *XBee sensor*. 2014. URL: `http://www.digi.com/pdf/ds_xbeesensors.pdf` (visited on 02/13/2014).

[72] Powercast Corporation. *Lifetime power wireless sensor system*. 2014. URL: `http://www.powercastco.com/PDF/wireless-sensor-system.pdf` (visited on 02/13/2014).

[73] *OpenMote: Open hardware for the Internet of Things*. May 2012. URL: http://www.openmote.com/.

[74] L. Doherty, W. Lindsay, and J. Simon. "Channel-specific wireless sensor network path data". In: *Computer Communications and Networks, 2007. ICCCN 2007. Proceedings of 16th International Conference on*. IEEE. 2007, pp. 89–94.

[75] Measurement Specialties. *Humidity and temperature Sensor - HTU21D*. 2014. URL: http://www.meas-spec.com/product/humidity/HTU21D.aspx (visited on 02/17/2014).

[76] ams AG. *TSL2560 ambient light sensor with SMBus Interface*. 2014. URL: http://www.ams.com/eng/Products/Light-Sensors/Light-to-Digital-Sensors/TSL2560.

[77] STMicroelectronics. *LIS3DH MEMS digital output motion sensor ultra low-power high performance 3-axes "nano" accelerometer*. 2014. URL: http://www.st.com/web/en/catalog/sense_power/FM89/SC444/PF250725#.

[78] Panasonic Corporation. *AMN41121 | NaPiOn Series*. 2014. URL: http://www3.panasonic.biz/ac/e/search_num/index.jsp?c=detail&part_no=AMN41121.

[79] K. Weekly, N. Bekiaris-Liberis, and A. M. Bayen. "Modeling and estimation of the humans' effect on the CO_2 dynamics inside a conference room". Under review. 2014.

[80] CO2Meter, Inc. *K-30 10,000ppm CO_2 Sensor*. 2014. URL: http://www.co2meter.com/products/k-30-co2-sensor-module.

[81] Linear Technology. *Dust Networks SmartMesh Power and Performance Estimator*. 2014. URL: http://www.linear.com/docs/42452.

[82] Linear Technology. *SmartMesh IP Application Notes*. 2014. URL: http://www.linear.com/docs/43189.

[83] B. T. Rosenblum. *Collecting occupant presence data for use in energy management of commercial buildings*. 2012. URL: http://escholarship.org/uc/item/1pz2528w.

[84] H. Liu et al. "Survey of wireless indoor positioning techniques and systems". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews* 37.6 (2007), pp. 1067–1080.

[85] K. Kaemarungsi and P. Krishnamurthy. "Modeling of indoor positioning systems based on location fingerprinting". In: *23rd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*. Vol. 2. IEEE. 2004, pp. 1012–1022.

[86] P. Bahl and V. N. Padmanabhan. "RADAR: An in-building RF-based user location and tracking system". In: *19th Joint Conference of the IEEE Computer and Communications Societies*. Vol. 2. IEEE. 2000, pp. 775–784.

[87] H. Zou et al. “An RFID indoor positioning system by using weighted path loss and extreme learning machine”. In: *1st IEEE International Conference on Cyber-Physical Systems, Networks, and Applications (CPSNA)*. 2013, pp. 66–71. DOI: `10.1109/CPSNA.2013.6614248`.

[88] C. A. Redlich, J. Sparer, and M. R. Cullen. “Sick-building syndrome”. In: *The Lancet* 349.9057 (1997), pp. 1013–1016.

[89] P. Wargocki et al. “The effects of outdoor air supply rate in an office on perceived air quality, sick building syndrome (SBS) symptoms and productivity”. In: *Indoor Air* 10.4 (2000), pp. 222–236.

[90] D. Wyon. “The effects of indoor air quality on performance and productivity”. In: *Indoor air* 14.s7 (2004), pp. 92–101.

[91] O. Seppänen, W. Fisk, and M. Mendell. “Association of ventilation rates and CO_2 concentrations with health and other responses in commercial and institutional buildings”. In: *Indoor air* 9.4 (1999), pp. 226–252.

[92] A. Zanobetti and J. Schwartz. “The effect of fine and coarse particulate air pollution on mortality: a national analysis”. In: *Environmental Health Perspectives* 117.6 (2009), p. 898.

[93] J. Qian and A. R. Ferro. “Resuspension of dust particles in a chamber and associated environmental factors”. In: *Aerosol Science and Technology* 42.7 (2008), pp. 566–578.

[94] *Dust sensor module P/N: DSM501 Specifications*. 2013. URL: `http://www.samyoungsnc.com/products/3-1%20Specification%20DSM501.pdf`.

[95] *PPD-20V (Particle Sensor Unit)*. 2013. URL: `http://www.shinyei.co.jp/stc/optical/main_ppd20v_e.html`.

[96] *GT-526S Handheld Particle Counter*. 2013. URL: `http://www.metone.com/documents/GT-526S-Datasheet.pdf`.

[97] J. Qian et al. “Size-resolved emission rates of airborne bacteria and fungi in an occupied classroom”. In: *Indoor air* (2012).

[98] D. Fox et al. “Bayesian techniques for location estimation”. In: *Workshop on Location-Aware Computing*. 2003, pp. 16–18.

[99] S. Maskell and N. Gordon. “A tutorial on particle filters for on-line nonlinear/non-Gaussian Bayesian tracking”. In: *Target Tracking: Algorithms and Applications (Ref. No. 2001/174), IEEE*. IET. 2001, pp. 2–1.

[100] E. F. Nakamura, A. A. Loureiro, and A. C. Frery. “Information fusion for wireless sensor networks: methods, models, and classifications”. In: *ACM Computing Surveys (CSUR)* 39.3 (2007), p. 9.

[101] U. Satish et al. “Is CO_2 an indoor pollutant? Direct effects of low-to-moderate CO_2 concentrations on human decision-making performance”. In: *Environmental health perspectives* 120.12 (2012), p. 1671.

[102] M. J. Thorpe et al. "Cavity-enhanced optical frequency comb spectroscopy: application to human breath analysis". In: *Optics Express* 16.4 (2008), pp. 2387–2397.

[103] E. S. R. Laboratory. *Up-to-date weekly average CO_2 at Mauna Loa.* 2014. URL: `http://www.esrl.noaa.gov/gmd/ccgg/trends/weekly.html`.

[104] A. C. Megri and F. Haghighat. "Zonal modeling for simulating indoor environment of buildings: review, recent developments, and applications". In: *HVAC&R Research* 13.6 (2007), pp. 887–905.

[105] A. K. Persily. "Evaluating building IAQ and ventilation with indoor carbon dioxide". In: *ASHRAE Transactions* 103 (1997), pp. 193–204.

[106] L Mora, A. Gadgil, and E Wurtz. "Comparing zonal and CFD model predictions of isothermal indoor airflows to experimental data". In: *Indoor air* 13.2 (2003), pp. 77–85.

[107] A. Baughman, A. Gadgil, and W. Nazaroff. "Mixing of a point source pollutant by natural convection flow within a room". In: *Indoor air* 4.2 (1994), pp. 114–122.

[108] K. Weekly, N. Bekiaris-Liberis, and A. M. Bayen. "Modeling and estimation of the humans' effect on the CO_2 dynamics inside a conference room". In: (Mar. 20, 2014). arXiv: `1403.5085`.

[109] P. A. Ioannou and J. Sun. *Robust adaptive control.* Prentice-Hall, Inc., 1995.

[110] M. Krstic, I. Kanellakopoulos, P. V. Kokotovic, et al. *Nonlinear and adaptive control design.* Vol. 8. John Wiley & Sons New York, 1995.

[111] A. Smyshlyaev and M. Krstic. *Adaptive control of parabolic PDEs.* Princeton University Press, 2010.

[112] S. Moura, M Krstic, and N. Chaturvedi. "Adaptive PDE observer for battery SOC/SOH estimation via an electrochemical model". In: *ASME Journal of Dynamic Systems, Measurement, and Control* 136 (2014).

[113] A. H. Sayed, A. Tarighat, and N. Khajehnouri. "Network-based wireless location: challenges faced in developing techniques for accurate wireless location information". In: *IEEE Signal Processing Magazine* 22.4 (2005), pp. 24–40.

[114] N. B. Priyantha. "The cricket indoor location system". PhD thesis. Massachusetts Institute of Technology, 2005.

[115] S. Lanzisera, D. T. Lin, and K. S. Pister. "RF time of flight ranging for wireless sensor network localization". In: *International Workshop on Intelligent Solutions in Embedded Systems.* IEEE. 2006, pp. 1–12.

[116] L. M. Ni et al. "LANDMARC: indoor location sensing using active RFID". In: *Wireless networks* 10.6 (2004), pp. 701–710.

[117] S. Thrun et al. "Robust Monte Carlo localization for mobile robots". In: *Artificial intelligence* 128.1 (2001), pp. 99–141.

[118] D. Schulz, D. Fox, and J. Hightower. “People tracking with anonymous and id-sensors using rao-blackwellised particle filters”. In: *International Joint Conference on Artificial Intelligence (IJCAI)*. 2003, pp. 921–928.

[119] F. Evennou and F. Marx. “Advanced integration of WiFi and inertial navigation systems for indoor mobile positioning”. In: *EURASIP journal on applied signal processing* 2006 (2006), pp. 164–164.

[120] S. Beauregard, Widyawan, and M. Klepal. “Indoor PDR performance enhancement using minimal map information and particle filters”. In: *IEEE/ION Position, Location and Navigation Symposium (PLANS)*. IEEE. 2008, pp. 141–147.

[121] A. Haeberlen et al. “Practical robust localization over large-scale 802.11 wireless networks”. In: *10th annual international conference on mobile computing and networking*. ACM. 2004, pp. 70–84.

[122] A. Bruce and G. Gordon. “Better motion prediction for people-tracking”. In: *IEEE international conference on robotics and automation (ICRA)*. 2004.

[123] M. Bennewitz, W. Burgard, and S. Thrun. “Using EM to learn motion behaviors of persons with mobile robots”. In: *Conference on Intelligent Robots and Systems (IROS)*. Lausanne, Switzerland, 2002.

[124] E. Dijkstra. “A note on two problems in connexion with graphs”. In: *Numerische Mathematik* 1.1 (1959), pp. 269–271.

[125] H. Hashemi. “The indoor radio propagation channel”. In: *Proceedings of the IEEE* 81.7 (1993), pp. 943–968.

[126] D. Scott. *Multivariate Density Estimation*. Wiley, 1992.

[127] E. Jones, T. Oliphant, P. Peterson, et al. *SciPy: Open source scientific tools for Python*. `http://www.scipy.org/`. 2001–.

[128] *BeagleBone*. `http://beagleboard.org/Products/BeagleBone`. 2013.

[129] *Simple DirectMedia Layer*. `http://www.libsdl.org/`. 2013.

[130] Pebble Technology Corp. *Pebble Smartwatch*. 2014. URL: `https://getpebble.com/`.

[131] U. Maurer et al. “Location and activity recognition using eWatch: A wearable sensor platform”. In: *Ambient Intelligence in Everyday Life*. Springer, 2006, pp. 86–102.

[132] B. Warneke et al. “Smart dust: Communicating with a cubic-millimeter computer”. In: *Computer* 34.1 (2001), pp. 44–51.

[133] S. J. Preece et al. “Activity identification using body-mounted sensorsÂŮa review of classification techniques”. In: *Physiological measurement* 30.4 (2009), R1.

[134] N. Ravi et al. “Activity recognition from accelerometer data”. In: *AAAI*. Vol. 5. 2005, pp. 1541–1546.

[135] H. Wang et al. “WLAN-based pedestrian tracking using particle filters and low-cost MEMS sensors”. In: *4th Workshop on Positioning, Navigation and Communication, 2007. WPNC’07.* IEEE. 2007, pp. 1–7.

[136] D. Gusenbauer, C. Isert, and J Krosche. “Self-contained indoor positioning on off-the-shelf mobile devices”. In: *International conference on indoor positioning and indoor navigation (IPIN).* IEEE. 2010, pp. 1–9.

[137] J. Collin, O. Mezentsev, G. Lachapelle, et al. “Indoor positioning system using accelerometry and high accuracy heading sensors”. In: *ION GPS/GNSS 2003 Conference.* 2003, pp. 9–12.

[138] B. Krach and P. Robertson. “Integration of foot-mounted inertial sensors into a Bayesian location estimation framework”. In: *5th Workshop on Positioning, Navigation and Communication, 2008. WPNC 2008.* IEEE. 2008, pp. 55–61.

[139] E. Bakolas and P. Tsiotras. “The Zermelo–Voronoi diagram: A dynamic partition problem”. In: *Automatica* 46.12 (2010), pp. 2059–2067.