Localization and Sensing Applications in the Pushpin Computing Network

by

Michael Joseph Broxton

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

Bachelor of Science in Computer Science and Electrical Engineering

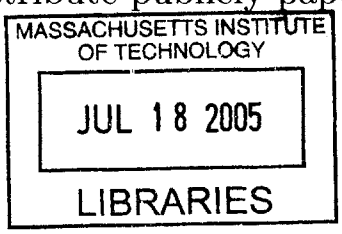
and

Master of Engineering in Computer Science and Electrical Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2005



Author .....
Department of Electrical Engineering and Computer Science
January 28, 2005

Certified by .....
Joseph Paradiso
Associate Professor, MIT Media Lab
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

Localization and Sensing Applications in the Pushpin Computing Network

by

Michael Joseph Broxton

Submitted to the Department of Electrical Engineering and Computer Science
on January 28, 2005, in partial fulfillment of the
requirements for the degrees of
Bachelor of Science in Computer Science and Electrical Engineering
and
Master of Engineering in Computer Science and Electrical Engineering

Abstract

The utility and purpose of a node in a wireless sensor network is intimately tied to the physical space in which it is distributed. As such, it is advantageous under most circumstances for a sensor node to know its position. In this work, we present two systems for localizing a network of roughly 60 sensor nodes distributed over an area of 1-m^2. One is based on a linear lateration technique, while the second approach utilizes non-linear optimization techniques, namely spectral graph drawing and mesh relaxation. In both cases, localization is accomplished by generating distance constraints based on ultrasound time-of-flight measurements to distinct, global sensor stimuli. These distance constraints alone are sufficient to achieve localization; no *a priori* knowledge of sensor node coordinates or the coordinates of the global sensor events are required. Using this technique, we have achieved a localization error of 2.30-cm and an error standard deviation of 2.36-cm.

Thesis Supervisor: Joseph Paradiso
Title: Associate Professor, MIT Media Lab

Acknowledgments

I would like to thank Josh Lifton and Joe Paradiso for their guidance and mentorship over the past three years. I also wish to thank the Things That Think Consortium and other sponsors of the MIT Media Lab for their generous support. Portions of this work are supported by National Science Foundation grant #ECS-0225492.

Contents

Chapter 1

Introduction

A forest ranger crests a hill and looks out over the landscape, satisfied with the view. He takes a small device – a sensor node not much larger than a ping-pong ball – out of the pouch on his belt and places it on the ground. He has placed hundreds of these tiny devices throughout the park today as part of a program to measure rainfall and monitor the health of the forest. In an emergency, these nodes could even be used to pinpoint the flash point of a forest fire or locate a lost hiker who has activated a small emergency beacon. Halfway across the world, a soldier crests a different hill, taking out a similar device and tossing it haphazardly from his hand. He is in a hurry to set up these sensors to track future enemy activities in the area, so he jogs another 200 meters where he drops another. Back home, a farmer has just finished a long day of deploying hundreds of tiny sensor nodes from the back of a tractor to monitor pH and salinity in the soil on his many acres of farmland. As he sits, he marvels how these tiny devices have doubled his yield over the past five years by helping him to optimize his growing cycles and crop rotation.

Environmental monitoring, target tracking, and resource management are likely to be revolutionized by the availability of numerous, low-cost, networked sensors. Sensor networks will allow us to see the world through a sharper lens. They promise to provide information at an unprecedented resolution and scale and to bring that information to us over an ad hoc communication network. In a very real sense, they will transform the surfaces on which they are deployed into a sensing apparatus – an

electronic skin.

However, it is implied in the scenarios described above that sensor nodes know their position. Without this information, sensor measurements are of dubious usefulness. However, because many nodes must be deployed over a short amount of time, the ranger, soldier and farmer do not have time to record these positions themselves. There are simply too many sensor nodes to make this approach practical. Instead, the sensor nodes must localize themselves. This could be done by using some fixed infrastructure such as GPS, however the cost and complexity of adding GPS to every node might diminish their utility. Furthermore, a fixed localization infrastructure might not be available. For example, there is no GPS on Mars.

Instead, sensor nodes must localize by capitalizing on their strengths: sensing and collaboration. The external environment is full of clues that can aid in localization. In particular, certain global stimuli that are detected by several nodes in the network create points of correspondence that serves to constrain the possible positions of the sensor nodes. By measuring their distances to global correspondence points and sharing these measurements with each other, sensor nodes can develop an accurate picture of their layout.

In this Thesis, we introduce our technique for localizing sensor nodes using range measurements to global stimuli. First, however, we will provide a brief history of sensor networks and discuss the state of the art in sensor network localization.

1.1 Historical Perspective

From their inception, sensor nodes have been envisioned as very small, numerous, and easy to distribute. This vision was best espoused by researchers at Berekeley who coined the now famous term "smart dust" to describe nodes in a sensor network[67]. They envisioned millimeter-sized sensor nodes made that used small, power efficient MEMS-based sensors and actuators for sensing and communication. Even though sensor nodes of this size are still not yet practical, this vision is revolutionary and alluring. It has undoubtedly contributed to the widespread interest that now surrounds

wireless sensor network research.

The first real "dust motes" to be developed at Berkeley were built using commercial of the shelf (COTS) electronic parts. These "COTS motes" were much larger than smart dust (2 or 3-cm in the largest dimension), but they had the advantage of being relatively cheap and easily manufactured in any electronics research lab [33]. Most COTS motes had modest computational ability in the form of an 8-bit microcontroller, several kilobytes of RAM, and RF communication hardware. Other COTS platforms including our own Pushpin sensor nodes[42] were also developed during this time at other research institutions who had an interest in hardware testbeds.

Sensor nodes are now commercially available, though they are still only in the very early stages of adoption in most application fields. Several derivatives of the original Berkeley COTS mote are now available commercially from Crossbow Technologies[21]. Their success has been driven in part by TinyOS, an open source operating system that runs on the motes[8]. TinyOS has been widely adopted and is fueled by a large developer community abroad. Another company, Ember technologies, sells wireless transceiver chips and a comprehensive communication protocol for robust wireless communication on a mesh network. Their products cater to customers who wish to develop their own sensor node, but prefer to focus more on the application at hand rather than the underlying communication details. In general, sensor-network like technologies are appearing throughout the electronics industries as more and more devices are equipped with wireless technologies such as Bluetooth, 802.11, and ZigBee. Cars, cell-phones, and PDAs may become the most widely deployed "distributed sensor" systems simply because they already enjoy widespread acceptance and use.

As the hardware for sensor networks has matured, many researchers have addressed the theoretical aspects of developing a distributed sensor system. These range from engineering challenges such as power conservation and dynamic routing in a mesh network to the general theories for writing software for a distributed system. Distributed programming paradigms and algorithms[18], distributed estimation, distributed data storage, in-network data compression, and sensor network operating systems have all become large fields of research unto themselves. It has also been

recognized that any practical sensor network implementation must include basic services for locating sensors in time and space: localization and synchronization. In this work, we are focused entirely on localization. The interested reader is referred to [52] for an excellent discussion of all aspects of synchronization.

The importance of localization in a sensor network can not be understated. A sensor reading is rarely useful without the time and location where it was recorded. Imagine if the measurements of a weather station did not include this meta-data. We would know that it is raining, but we wouldn't be able to say when or where! The position of every weather station is known because a human operator with a map or GPS has noted its location. The purpose of a sensor network is also intimately tied to the location where it is deployed, hence it is important that the positions of every sensor node be known. Unlike the weather station, however, nodes in the sensor network are too numerous to have their positions marked by hand. It may be possible to equip every sensor node with its own GPS receiver, but this is a burden in both cost and complexity. Furthermore, GPS signal do not penetrate into some environments (buildings, caves, other planets), and they have limited resolution (several meters for civilian GPS). Other location infrastructures such as the AT&T BAT [66], Microsoft RADAR[12], and MIT Cricket[55] systems have demonstrated centimeter accuracy in indoor environments, but such systems are not widely deployed. Furthermore, there is no need to rely on any infrastructure; the best solution is to let the sensor network localize itself using whatever means it has at its disposal. Fortunately, due to their powerful sensing, communication, and distributed processing capabilities, sensor nodes are very well suited to this task.

1.2 State of the Art

Ad hoc localization in a sensor network is based on generating distance constraints between sensor nodes and fixed *points of correspondence* in space. A point of correspondence may be the location of another sensor node or it may be the origin of a global sensor event that multiple nodes have detected. The coordinates of the sensor

nodes and the points of correspondence need not be known beforehand (though it helps tremendously if there are a few "anchor" nodes that do know their position ahead of time); the distance constraints alone are often sufficient to find unique node positions in a coordinate system up an arbitrary translation, rotation, and possible reflection.

Existing sensor network localization schemes differ primarily from each other in two ways: the ranging technique used to generate constraints and the localization algorithm used to turn the constraints into an estimate of node coordinates. In practice, these can generally be divorced from each other. The ranging technique that is employed does not generally dictate the best localization algorithm to use and vice versa. Below, we will give a brief overview of the state of the art in ranging techniques and localization algorithms.

1.2.1 Generating Range Constraints

There are numerous ways to generate localization constraints in a sensor network using the great number of varied sensors, actuators, and communications channels available on a typical sensor node. Perhaps the simplest of these is to use logical distance over the network (hop count) to approximate physical distance[44]. Network hop count metrics are appealing because they do not require any additional ranging hardware. The tradeoff is that a relatively large, dense network is required to make this method practical for precise positioning. With additional in-network averaging, Nagpal et. al. report that accuracy of 0.1 times the radio range can be achieved if nodes have 15 or more neighbors[45]. Network ranging can also be refined with additional anchor nodes[32], and it still finds use in applications where only rough positioning is required or as an augmentation to another more precise localization technique[54].

Sensor nodes that use radio frequency (RF) communication hardware often use received signal strength (RSS) on the radio channel to approximate distance[12]. This is often cited as a poor ranging technique due to the susceptibility of RF to multipath, scattering, occlusion, and varied attenuation through different media; all

of which contribute to non-isotropic signal attenuation. Nonetheless, RSS or even quantized RSS has been shown to produce more accurate localization results than network connectivity alone[49].

When precise range measurements are needed, ultrasonic ranging is commonly used. Typically, a brief ultrasonic pulse is generated simultaneously with another signal with near-instantaneous propagation speed such as a flash of light or a radio signal. The time difference of arrival (TDoA) is measured between this signal and the ultrasound pulse[16, 55, 59, 66], and this is used to infer distance by dividing by the speed of sound. The high directionality of ultrasound can be restrictive in certain sensor network geometries. Audible sound signals are more omnidirectional, however their larger wavelength leads to a reduced accuracy when detected with a simple rising edge discriminator. The accuracy can be increased considerably with a more sophisticated detector. If audible signals are modulated with a known signal (sometimes called a "chirp"), position can be decoded from the phase of the chirp, which is extracted with a matched filter on the receiver[27, 28].

When possible, it is preferable to take advantage location-rich information available for "free" in the sensor network's environment. A sensor node can glean considerable localization information simply by listening on its standard complement of sensors[16, 50]. No special localization hardware is required. An external stimulus can be environmental, such as lightning and thunder from a storm, or man made. For example, Taylor et. al. have developed a technique for simultaneously localizing a sensor network and tracking a moving target merely by measuring the range to the target[64].

1.2.2 Localization Algorithms

The advent of sensor networks has generated a significant body of new research on localization algorithms. Many techniques, both old and new, have been explored in the recent literature. Localization algorithms are generally divided into two categories: anchor-based techniques and anchor-free techniques. Anchor-based techniques incorporate the positions of certain "anchor" nodes with *a priori* knowledge of their

coordinates. Anchor nodes severely constrain the localization problem and generally result in simpler localization algorithms. On the other hand, anchor-free techniques attempt to localize sensor nodes based solely on distance constraints between nodes. As we will discuss in Chapter 3, localization using distance constraints but no a priori knowledge of node positions (i.e. no anchor nodes) is a non-linear problem that must be solved using an iterative optimization technique.

Much early localization research centered around anchor-based techniques. The most widely known and commonly used technique is called *lateration*[18, 44, 58], which is similar to triangulation, but can take into account more than three reference points via a linear-least squares fit. A very light-weight alternative to lateration for determining a node's position involves computing whether it falls inside of a polygon defined by several anchor points[32, 59]. The computation required to do this is minimal, often only requiring the pairwise comparison of two numbers. A general overview of this and other anchor-based methods can be found in [40].

Several anchor-free techniques have been proposed for localizing a sensor network that contain no nodes with prior knowledge of their coordinates. These techniques typically employ some form of non-linear optimization. A simple, intuitive approach is to solve the localization problem by simulating a similar physical system. The relaxation of a mesh of balls (which represent sensor nodes) and springs (which represent distance constraints) has been shown to produce accurate results[34]. A more theoretical technique is to formulate localization as a semi-definite programming problem[14, 26]. The fact that such a computation can be distributed[14] and solved in polynomial time[63] are attractive theoretical properties that lend credence to this approach. Shang et. al. have adapted a technique known as metric multi-dimensional scaling (MDS) for sensor network localization. MDS originated as a psychometric tool for drawing a graph given only a set of edge constraints between vertices. MDS is solved via an optimization technique known as *majorization*[15].

In general, non-linear optimization problems are prone to false extrema. This may result in a false solution to the localization problem. Priyantha et. al. refer to this phenomenon as "folding," because false solution frequently correspond to layouts

where some nodes are given coordinates that are reflected across from their correct coordinates[54]. The best way to avoid false minima is to start the optimization with a good initial guess of the nodes' coordinates. Priyantha proposes a "fold-free" algorithm for generating such a guess based on a set of simple heuristics that take network distances as well as measured constraints into account. Howard et. al. uses integrated inertial sensor measurements (which are inexact because they accumulates error over time) as an initial guess for a mesh relaxation technique[34]. A different technique called spectral graph drawing (SGD) has been proposed by Koren and Gotsman for generating an initial guess of network layout[29, 37]. Unlike metric MDS, which generates coordinates that match the provided edge constraints as closely as possible, SGD generates an *approximate* layout. A distributed implementation of SGD has been shown to scale in excess of 1000 nodes[29].

Though we have summarized some of the most popular techniques here, we direct the interested reader to [11] for a more complete introduction to sensor network localization.

1.3 Pushpin Localization

In this work, we present localization systems that have been implemented on the Pushpin Computing platform – our dense sensor network test bed that consists of approximately 50 nodes spread over an area of 1-m^2. Our ranging technique involves measuring the time difference of arrival between a flash of light and an ultrasound pulse that are generated by a handheld device called the "Pinger." The Pinger can be triggered anywhere above the Pushpin network to generate a point to which all nodes can measure their distance. These measurements serve as distance constraints for ad hoc localization. Though the Pinger is artificial, it is meant to emulate the behavior of a global sensor event that a large sensor network might expect to detect "in the field."

Two different systems that use global constraints for ad hoc localization are described in this paper. First, a system based entirely on triangulating node positions

using trigonometry and lateration is proposed. This system, called the *Pushpin Lateration System* relies on the single assumption that the global stimulus from the Pinger occurs somewhere directly above some node (any node), and orthogonal to the plane of the sensor network. The extreme density of the Pushpin network enables us to make this assumption without significant loss of accuracy, since nearly any position above the Pushpin network will be within a few centimeters of being directly above some node. We have achieved a localization error of 5-cm using this technique.

The second system described here is based on non-linear optimization via mesh relaxation. To avoid false minima in the localization process, we find an initial guess of the node layout using a technique called spectral graph drawing (SGD) which, given sufficient distance constraints between nodes, is guaranteed to produce a set of coordinates that approximates the correct layout of the sensor network. Spectral graph drawing and mesh relaxation (MR) are used to find the coordinates of a small subset of "anchor" nodes as well as the coordinates of the global events generated by the pinger. The remaining nodes then determine their coordinates using lateration. This system is called the *Pushpin SGD/MR system.* The SGD/MR system is more complex than the lateration system, but no assumptions are made about the location of the global events relative to the sensor network.

The remaining chapters are organized as follows. Chapter 2 described the hardware and simulation platforms on which this research was carried out; namely the Pushpin Computing hardware test bed and the Pushpin Simulator. Chapter 3 is dedicated to mathematical formulations of the algorithms used in Pushpin localization. These are lateration, spectral graph drawing, and mesh relaxation. Though these algorithms play a central role in localization, they are merely a part of the two overall localization *systems* described above. These are outlined in more detail in Chapter 4. The accuracy and effectiveness of our techniques is presented and discussed in Chapter 5. Finally, Chapter 6 contains future work and closing remarks.

Chapter 2

Pushpin Computing

Developing a new algorithm for a wireless sensor network is much like designing an electronic integrated circuit. Once the pen and paper design is complete, it must be verified first in simulation, then through a physical prototype. Both approaches have their merits. Simulations are used to rapidly determine optimal system parameters and to facilitate exhaustive testing over a wide range of inputs without invoking an expensive fabrication process. Such tests are impractical or too time-consuming to implement in a physical system. On the other hand, prototyping is essential for verifying that the design is robust to noise, non-isotropic signal propagation, part variability, processing and power limitations, and other non-ideal characteristics of the real world. A prototype also serves as a final sanity check as to whether the system works at all.

The best designs are those that have been subjected to both techniques. Despite this, most contemporary sensor network researchers have relied solely on simulations to verify their designs and algorithms. This is due in part to the fact that, until recently, there have been relatively few sensor network hardware testbeds available for prototyping. Even systems that now enjoy widespread use such as the Crossbow Motes[21] are still evolving as development tools (though they are improving rapidly thanks to large developer communities). In addition, many sensor network researchers are theoreticians. They do not have the experience, time, or interest in working with electronics or software. As a result, there is relatively little research that has been

implemented in both simulation and hardware.

The algorithms discussed in this thesis were prototyped on the Pushpin Computing test bed. This chapter contains a detailed description of our platform. First, we discuss its history and motivation. Next, we describe the Pushpin hardware (particularly the sensing and actuation hardware used in localization), and "Bertha", the operating system for Pushpin computers. We also describe the system and infrastructure for developing code on the Pushpins. Finally, we describe our own simulation software called Pushpin Simulator, which realistically simulates a 50-100 node sensor network performing localization. As an addendum to this chapter, we describe the "Tribble"; a soccer-ball shaped, wired sensor network based on the Pushpins that is meant to behave as a distributed robotic system. While the Tribble has nothing to do with localization, it does embody the Pushpin research vision of the sensor network as an electronic skin, which we now discuss.

2.1 Platform Overview

Pushpin Computing was originally designed as a hardware instantiation of the nodes in Bill Butera's *Paintable Computing* simulator [18]. Paintable Computing envisions the sensor network as a ubiquitous, pervasive computing platform that covers (is painted onto) or is manufactured into the surfaces of everyday objects. These unusually large and dense *paintable* networks are meant to act as the interface between humans and the environment around them, providing an infrastructure for dense sensing, substantial computational and storage resources, and distributed information display that can be tapped by a nearby user. Paintable computing builds upon the legacy of past works in distributed systems; for example the Smart Matter research at Xerox Parc[30] and the Amorphous Computing initiative at MIT[9].

The basic algorithmic unit of the Paintable computer is a self-contained, self-replicating piece of mobile code called a *process fragment* that can move and copy itself node-to-node and execute code on any node where it has been accepted. In this respect, it acts very much like a virus, taking control of the resident node and

changing its behavior. The inspiration behind this idea is one of many examples where the behavior of natural systems has influenced sensor network development. A prime example of biological insiration appears in the early distributed systems research of the Amorphous Computing group[9] at the MIT AI Lab (now CSAIL). This research explored many aspects of self-organizing behavior that are relevant to sensor networks, including robust communication and routing channels and self-organized coordinate systems [45]. Not surprisingly, members of the Amorphous Computing group have also authored recent papers on synthetic biology and the programming biological cells[68]. This suggests that the fields of distributed computing research (including sensor network research) and biology are coming tantalizing close together. This trend has led us to develop our own research vision of an ultra-dense sensor network modeled after a biological system with distributed, dense sensing capabilities: skin.

2.1.1 Sensor Networks as Skins

The rapid decrease in the size and cost of electronic integrated circuits, the emergence of new sensor and actuator manufacturing technologies such as MEMS, and the emergence of novel sensor materials such as piezoelectrics and magnetoresistors have led to a recent revolution in extremely tiny sensors. Thanks to mass manufacturing in large industries such as automobiles and aerospace, sensors that measure most modalities, including pressure, acceleration, and orientation in a magnetic field, are now cheap and commercially available, often at sizes 10-100 times smaller than their predecessors. We believe that this revolution in tiny sensors will soon be followed by a corresponding revolution in small, extremely dense sensor networks. This is the guiding vision of the Pushpin Computing platform: to develop the sensor network as an electronic skin [48]. The Pushpin testbed is a first step in this direction. It achieves an unusually high node density for a sensor network (100 nodes/m^2), and we see this as the first step towards even higher densities in our future research.

It is no longer an exaggeration to say that sensors are the size of a speck of dust. Take, for example, the P592 Lucas NovaSensor PiezoResistive Pressure sensor that is 1-mm by 1-mm by 0.6-mm [56]. This is similar to the dimensions of the

mechanoreceptors (i.e. touch receptors) in the human hand, which have an active area that ranges from 2-mm^2 to 50-mm^2 [36] [1]. Small sensor arrays will be logistically more convenient and easy to install than existing technologies. Designers will be able to instrument structures such as aircraft wings where sensors were previously too bulky and obtrusive to be practical. An early example of this can be found in a wind tunnel at NASA Glenn Research Center where an array of three Lucas NovaSensors replaced a 20-ft tube that piped air to an older, larger pressure sensor. The tubing was necessary because the bulky old pressure sensor would have perturbed the air flow it was meant to measure if it had been placed directly in the tunnel itself [56].

Arrays of small sensors promise to bring previously unheard of sensing density to numerous fields including fluid dynamics measurement, robotic telepresence systems (e.g. NASA's Robonaut platform [25]), prosthetics, and sensing for autonomous robotic systems. For example, it is suggested by Xu et. al. in [70] that measurements of flow separation on the leading wing of an unmanned aerial vehicle might be used in an autonomous control system (indeed, the phrase "smart skin" dates back well over a decade in aerospace circles). Xu demonstrated a flexible shear-stress MEMS sensor with an area of 15-cm^2 and a sensor density of 4 sensors/cm^2 that could be used for this purpose. Another sensor array effort has achieved a density of 250 sensors/cm^2 using the technique of scanning through two perpendicular rows of wires separated by a deformable Ag-filled polymer whose resistance changes with deformation [47]. A similar effort that used capacitive rather than resistive sensing reports a sensing density of 1 sensor/cm^2 [60]. As the fabrication techniques for these arrays continue to improve, they will match or exceed the average density of mechanoreceptors in the skin of the human hand, which ranges from 20 receptors/cm^2 in the palm to 140 receptors/cm^2 in the fingertip[36].

While these advances in dense sensor array technology are extremely exciting, they may be stifled by the limited bandwidth available to route data from so many

[1] For the reader interested in learning more about the sensing capabilities of the biological skin, we recommend the publications of the Laboratory of Dexterous Manipulation at Umea University in Sweden [23]. In particular, [36] is an excellent introduction to the physiology of the skin on the human hand.

sensors. Consider sampling 8-bit sensors at 100-Hz in a 1-m^2 patch of electronic skin with a density of 140 sensors/cm^2. This patch alone would generate 1.12-Gbps of data, which is on the same order of magnitude as the bandwidth of a fast AGP graphics bus on a modern PC. The task of routing this data, let alone processing it and acting upon the results would be impossible in a centralized system even at this very limited scale.

On the other hand, a distributed network of 100 sensor nodes, each managing a 10-cm^2 array of sensors would be quite capable of handling this volume of data. A distributed system of sensor nodes would not attempt to route all of this data off the network (it has been shown mathematically that this is prohibitive in a mesh network due to bandwidth limitations [51]). Rather, the data would be processed and reduced in the sensor network itself. We see something of this phenomenon in the human nervous system. Responses to extremely unpleasant stimuli that require an immediate reaction by the nervous system (such as a burn or cut) are generated by autonomous spinal reflexes that directly trigger a muscular contraction. Lateral excitation and inhibition of spatially adjacent nerve signals is also common at synaptic junctions in the spinal column and in the thalamus[24]. Inhibition of stationary, low-priority signals ensures that sensory bandwidth can be made available for critical signals to pass through when necessary. We have incorporate functions that mimic this behavior in our Tribble sensor network testbed (The Tribble is discussed in more detail in Section 2.5).

These biological metaphors have inspired us to seek a hardware platform with a high node density and to develop software that mimics biological systems. However, as these ideas were being developed, we discovered another compelling and immediate application for the Pushpin network. A dense sensor network is small and easy to manipulate and affect. This makes it an ideal prototyping platform for developing general sensor network algorithms. This is exemplified by our work on localization. Despite being developed on our dense sensor network, the localization algorithms we have explored are general, and have implications for a sensor network of any size. We now turn to a more in depth discussion of the prototyping features of the Pushpin

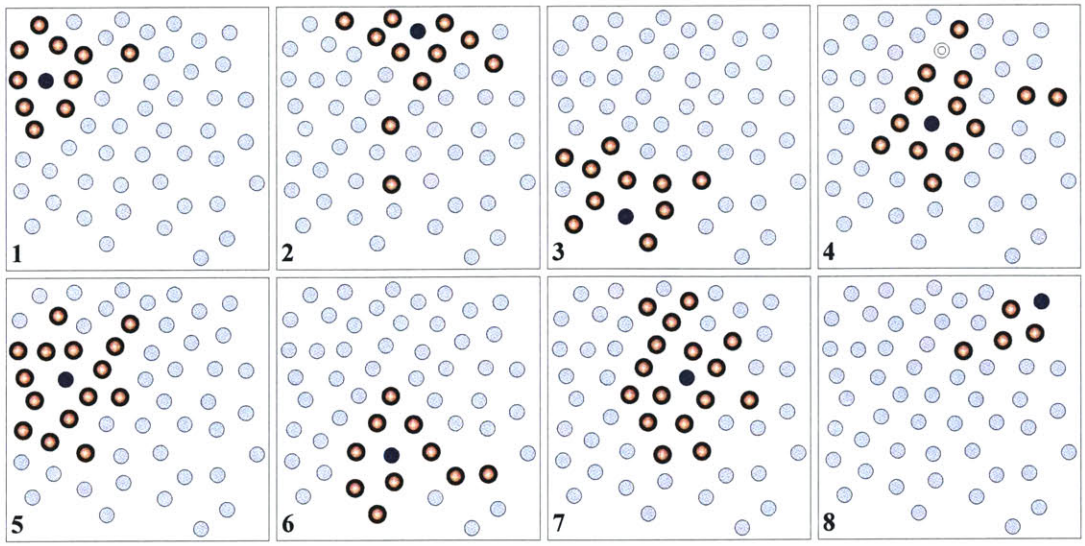


Figure 2-1: The neighborhood of nodes with which a Pushpin can communicate varies considerably from Pushpin to Pushpin. The dark, solid circle in each of the above frames indicates a Pushpin that is constantly transmitting packets via IR. The circles with bold outlines indicate which of the surrounding nodes reliably receives the packets directly from the originating node. The single outlined node in frame 4 indicates a node that sporadically enjoys good reception. Aside from this one exception, all other nodes in the neighborhood received virtually 100% of the transmitted packets. Note that there was no other network traffic and the neighborhoods shown only depict one-way communication. In particular, note the transmitter in frame six belongs to the neighborhood of the transmitter in frame 2, but not the other way around; clearly, the neighborhoods shrink if two-way communication is required for membership.

Computing platform.

2.1.2 Sensor Network Prototyping

Most sensor network researches will agree that programming hundreds (or even tens) of tiny wireless sensors is a time consuming, onerous task. Various methods have been proposed to ease this process. These vary from pre-programming the nodes during manufacturing to injecting virally spread mobile code fragments into the network[18]. While these approches may work in thoroughly debugged, production sensor networks, they are not good solutions for the sensor network developer trying to debug a new algorithm.

Just as an electrical engineer relies on the pliability of a proto-board to aid him

in the early stages of circuit design, we have found that having the entire sensor network within arms reach (and more importantly, within easy reach of our computer and debugging tools) can be very advantageous when developing operating-system level software for the sensor network. In fact, since most sensor nodes have extremely limited persistent memory stores and RAM resources, the "operating system" will often also include the application code as well. In any case, the number of debug cycles necessary to get software running on the sensor node can be overwhelming unless code can easily be uploaded to every node in the network in a short amount of time. Though the Pushpin network was originally envisioned to be a prototype of an electronic skin, we have learned that a more practical, immediate use is as a debugging tool used during the development process in between simulation (which is already used widely throughout sensor network research) and deployment.

The Pushpins have several characteristics that make them well suited for debugging work. First and foremost, they are small and easily manipulated by hand, yet numerous enough to realistically realize a large sensor network. The process of relocating a Pushpin is as simple as moving a pin on a corkboard. Controlling the sensor stimulus experienced by nodes in the network for testing purposes is also simplified due to their proximity. In our experiments, we cast shadows of shapes onto the network to test edge detection and shape recognition algorithms. For our localization experiments, a large, natural global stimulus such as a thunder cloud was generated on a smaller, more manageable scale using a custom signal generating device called the Pinger (Section 2.3.2). Also, the RGB LED on the top of each node enables the sensor network developer to assess the state of individual Pushpins or the entire network at a glance. Dynamic phenomena such as communication gradients and routing patterns that are normally difficult to debug or visualize are often intuitively revealed on the Pushpins.

The extreme density of the Pushpin network is made possible in part by the nodes' infrared (IR) communication hardware, which results in a communication radius of approximately 20-cm and a network neighborhood of ten neighbors per node on average (See Figure 2-1). Radio Frequency (RF), which are used for communication in

most sensor networks, would have been difficult to constrain at this close range. Furthermore, a PC can communicate with the entire network of Pushpins simultaneously over a serial port connected to an IR spotlight that casts light over the entire network. This one-way communication port from a PC to the sensor network is used to send control message and operating system updates to all nodes simultaneously. Using this global programming bus, an update of the operating system and application code for every Pushpin takes less than one minute. All of this makes for an extremely fast debug cycle that enables rapid prototyping of new sensor network algorithms on the Pushpin testbed. The IR spotlight, as well as other infrastructure, is described in more detail in Section 2.2.3

2.2 Pushpin Hardware and Software

The Pushpin system was originally developed and presented by Josh Lifton in [42]. This section briefly describes the Pusphin hardware and software. A more complete overview appears in Lifton's masters thesis [41].

Several major infrastructural improvements have been made to the Pushpin platform during the course of this work. These include an improved communication library, global programming of the network from a PC, and an open source development environment for testing and debugging sensor network code. Since the Pushpin localization work would have been infeasible without these improvements, they are also discussed here.

2.2.1 Anatomy of a Pushpin Node

A Pushpin (shown in Figure 2-2) is comprised of a stack of four circuit boards, one for each major function of a sensor node: power, processing, communication, and sensing. The Pushpin moniker is derived from the two tensile metal pins that protrude from the bottom of the node. These pins make contact with two parallel sheets of foil that are embedded into a 1.2-m by 1.2-m foam composite board. The Pushpins derive their power and ground from this connection. This is convenient, as it frees the

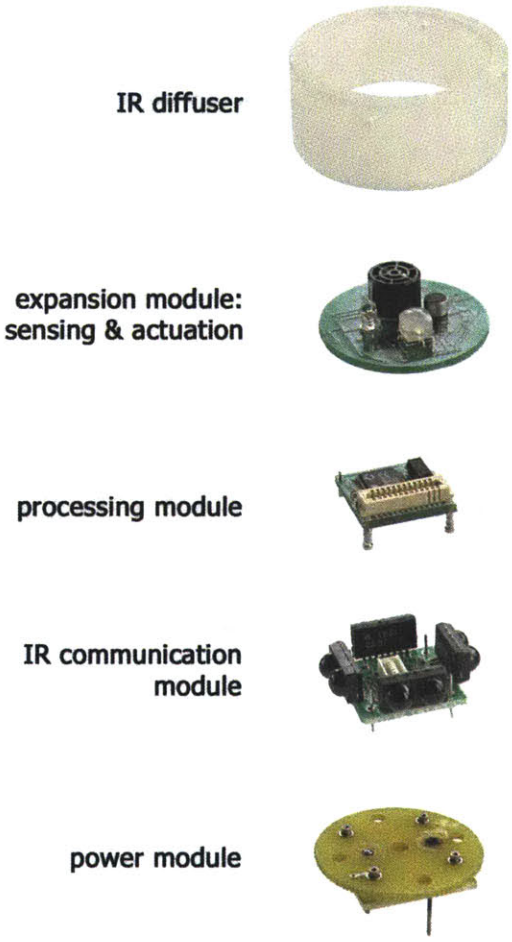


Figure 2-2: This expanded view shows the four functional layers of the Pushpin node. When assembled, the Pushpin measures 3-cm in diameter by 3-cm high.

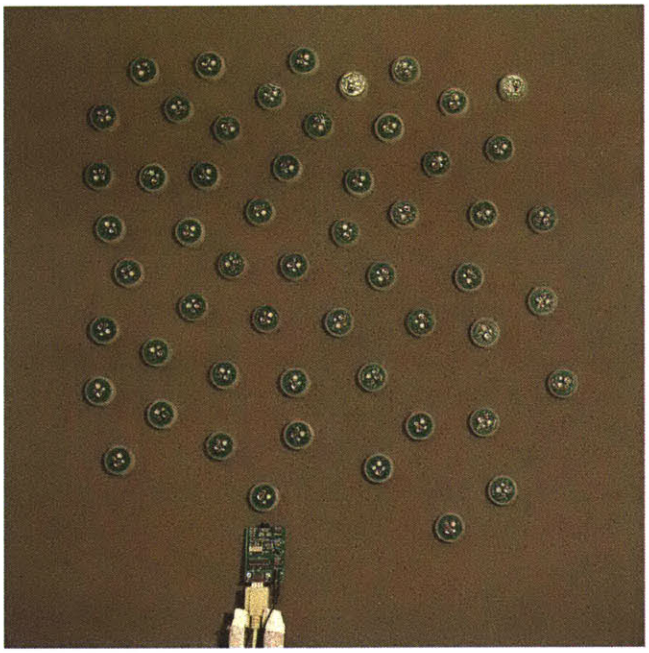


Figure 2-3: The 1.2-m by 1.2-m substrate into which the Pushpins are pressed provides a power and ground connection for the nodes, alleviating the need for batteries. The gateway node at the bottom of the image provides two-way communication between the network and a PC.

sensor network developer to focus on other aspects of sensor network research rather than worrying about re-charging batteries or conserving power.

Pushpins communicate with each other at 96-kbps using infrared (IR) light. Figure 2-1 shows that each node has roughly ten neighbors using the IR communication scheme, hence the Pushpins are an exceptionally compact platform that realizes a wireless sensor network. Establishing a neighborhood of this size using radio-frequency (RF) communication hardware would have been challenging, since RF is difficult to constrain over such small distances. The IR communication layer provides relatively isotropic communication coverage via four infrared transmitter/receiver pairs pointing in the four cardinal directions. To further diffuse the light, each Pushpin is surrounded by a frosted polycarbonate ring. Figure 2-4 shows the distribution of IR light with and without this ring as seen through a black and white IR-sensitive camera.

At the heart of the Pushpin node is an 8-bit, 22-MIPS microcontroller made by

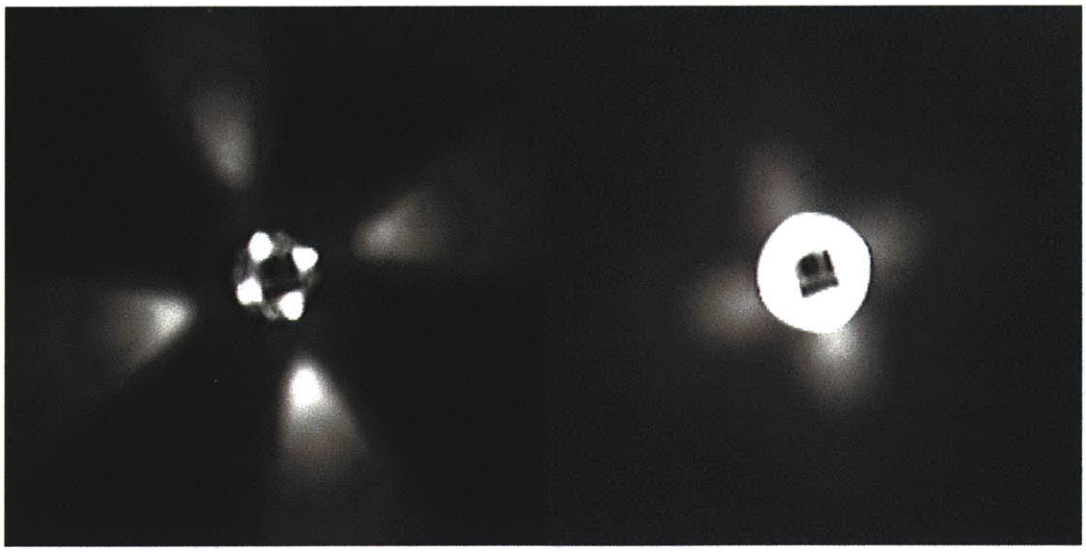


Figure 2-4: The left image shows the distribution of IR from the Pushpin communication layer as seen by an IR-sensitive black and white camera. The right image shows the same Pushpin after a frosted polycarbonate ring is added as a filter to help diffuse the IR. Clearly, the diffuser ring helps considerably in creating an isotropic communication channel; a cylindrical ring lens may do even better.

Silicon Laboratories[2] (previously Cygnal) [6]. In addition to the processing core, which is based on the Intel 8051 architecture, this chip includes a host of digital and analog circuitry for reading sensors and driving actuators, including A/D converters, D/A converters, pulse-width modulators, and external interrupt lines. The chip has 32-kilobytes of flash memory for persistent program storage as well as 2048-bytes of external and 256-bytes of internal volatile RAM. When the processor is running at a full 22-MIPS, it draws roughly 12-mA of current at 3.3-V. For comparison, the entire Pushpin node can draw as much as 55-mA when communication, actuation, and sensing hardware is running, though most of this current is used to drive the RGB status LED. The cygnal processor does have low power modes that throttle down the processor speed or put the processor core to sleep, but these modes are not used by the system software since power is not a scarce resource in the Pushpin system.

The top circuit board in the stack, the expansion module, is dedicated entirely to sensing and actuation. Since sensing and actuation packages are highly application

[2]Silicon Laboratories Part #c8051f016

specific, the expansion module can be easily redesigned and replaced. An expansion module that measures sonar time-of-flight was designed for these localization experiments. It includes a phototransistor, 40-kHz ultrasound transducer, and an electret microphone for sensing and an RGB LED for actuation. This module is described in detail in Section 2.3.1.

2.2.2 Operating System

Bertha is the operating system for Pushpin computers. It provides abstract interfaces for commonly used services such as reliable communication and sensor and actuator access. These services, which are made available through a set of APIs, also include utilities such as a real time clock, random number generation, and access to interrupt routines. The localization software, for example, uses the interrupt routines and the real time clock to precisely measure the difference in arrival times between a light flash and ultrasound ping.

Due to the microcontroller's limited memory, application code is compiled into the operating system code directly. This complete package is then uploaded to a node as an operating system update. Operating system updates are received by a bootloader application that resides permanently in flash memory. The bootloader performs version checking, error detection, and error correction on these updates, and then writes the update to bootable non-volatile program memory. We use a large 108-LED *IR spotlight* to beam updates to every Pushpin simultaneously from a desktop computer where new code is written and compiled (see Figure 2-5). Using this interface, we can reprogram 100 nodes in parallel with an entirely new operating system software in less than a minute, thus significantly reducing the time required for a debug cycle.

Communication Routines

The most complex service provided by Bertha is a comprehensive communication library for sending and receiving data packets. This library is built around two

primitive packet types: an unacknowledged packet with no guarantee of delivery to neighbors and an acknowledged packet with a high degree of certainty as to whether a transmitted packet was properly received. The difference between these packet types is similar to the differences between the UDP and TCP Internet protocols; unacknowledged packets are used for sending a large amount of data where lost packets do not seriously hamper the performance of the system, while acknowledged packets are a more precise tool for sending critical data to a single network peer. In Bertha, reliable packets are implemented using an automatic repeat request (ARQ) protocol. Regardless of their type, all packet headers and data payloads are subject to an 8-bit cyclic redundancy check (CRC) to detect transmission errors.

In order to facilitate the addressing of packets to other nodes, each Pushpin has a network ID. These IDs need not be globally unique; in some sensor network applications, it is sufficient for a node to have an ID that is only unique amongst the set of nodes with which it can directly communicate. For example, a sensor node which is tracking a target only needs to know the ID of the immediate neighbor who it will wake up when the target enters that neighbor's sensing range. The Bertha communication library can automatically choose a random ID that is unique to the network neighborhood of a node, or it can use a pre-programmed ID that has been individually assigned by the sensor network operator. For this work, each node was assigned a globally unique ID in order to simplify the task of gathering localization data from the sensor nodes to a desktop computer.

An implementation of directed diffusion [19] provides basic routing services for the network. When a node wishes to transmit a message such that every node in the network receives it at least once, it can send a *broadcast* message to its neighbors. The neighbor chooses whether or not to retransmit a broadcast message based on the message hop count. If the hop count is less than or equal to the hop count from the last time the node transmitted this message, or if the message has not been seen before, the node resends the message to its neighbors. This prevents backpropagation and ensures that the retransmissions will eventually die out. Broadcasts are used in the localization system for sharing the coordinates of anchor nodes with *passive* nodes

that do not yet have enough information to localize themselves.

2.2.3 Pushpin System Infrastructure

The first version of Bertha was completed in 2002 at the same time as the Pushpin node hardware was completed for Josh Lifton's Master's thesis[41]. However, there was still much work to be done before the Pushpins became a mature and useful development platform. For example, it was still very difficult to program the nodes with new code. A rudimentary PC interface had been built to inject updates in the form of mobile code that would spread virally through the network, but this technique was very slow and prone to error. The mobile code updates, which were an attempt to emulate the behavior of Butera's Paintable nodes, were quickly set aside so as to concentrate on developing more fundamental functionality.

Mobile code updates have been replaced in the current Pushpin nodes by a small bootloader program that resides in flash memory and monitors the IR communication hardware for updates that contain both the operating system and application code. When updates are received, they are checked via a 8-bit CRC for accuracy and then placed at a bootable location in flash memory. The bootloader does not retransmit the update to other nodes. Rather, code updates are transmitted to all nodes simultaneously from a desktop PC, using an *IR spotlight.* The IR spotlight is an array of 108 IR LEDs plus signal conditioning circuitry for converting RS-232 serial line levels to levels for driving the LEDs. The spotlight is suspended approximately 1.5-m in front (and slightly above) the Pushpin network as shown in Figure 2-5.

Development on the original Pushpins was also slowed by the lack of infrastructure for input and output from the network. A very limited facility for controlling the Pushpins (starting them, stopping them, and selecting the current state) did exist, however no software had been developed to query and collect data from the network for analysis. Under these circumstances, it was nigh impossible to tell what was going wrong with an algorithm in the network. The best source of debugging information came from the LEDs on the Pushpins themselves which, while very useful for debugging certain problems, were not sufficient on their own.

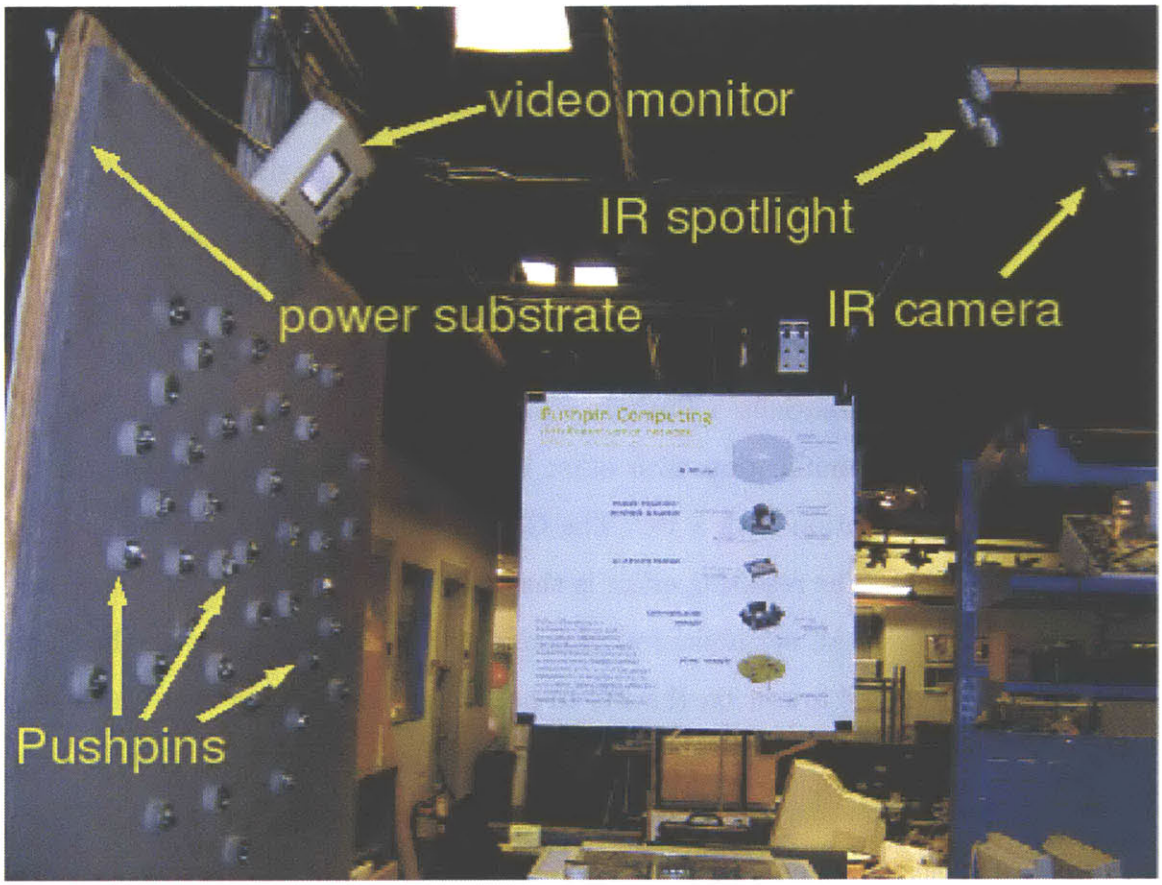


Figure 2-5: The full Pushpin experimental apparatus includes an IR spotlight for parallel programming and a communications gateway (not shown) for debugging the sensor network from a PC. An IR sensitive black and white camera and video monitor is used to see gross communication patterns on the network.

We saw a clear need for a powerful software tool that could help us program, control, debug, and query the Pushpins from a PC. Over time, we have developed such a tool, which we call the *Pushpin Debugger*; a cross-platform, open-source development environment written in Python [4]. It interfaces with SDCC, an open source tool chain for compiling and linking code for the Intel 8051 architecture[3] on which the Pushpins are based [5]. Code is written in an external editor, then compiled using SDCC. The code is then packaged by the Pushpin Debugger and uploaded to the Pushpins over the IR Spotlight. The Pushpin Debugger also uses the IR spotlight to send global control messages, such as start and stop messages, to all nodes simultaneously.

The IR Spotlight channel is one-way only: from the PC to the Pushpins. In order to query the network, the Pushpin Simulator sends and receives packets over a separate interface called the *Gateway Node.* This node, which appears in the bottom left of Figure 2-3, is essentially a Pushpin equipped with two communications interfaces: (1) IR for communication with other Pushpins in its neighborhood and (2) RS-232 for communication with a PC. The Gateway Node reflects packets between these two interfaces. The resulting effect is that the Pushpin Debugger can send and receive packets to a small neighborhood of nodes within close proximity to the Gateway Node. This allows a user on the PC to query and collect data from the sensor network through this port. Queried data can then be saved to disk for later analysis. We have used this capability to collect time-of-flight measurements from the network for offline analysis and simulation.

A final tool that has been very useful for debugging the communication stack in Bertha is an IR sensitive black and white camera. This camera has allowed us to see the dynamics of the communication algorithms. With it, it is very easy to observe the diffusion of broadcast messages and gradients in the network and to see when the communication channel is being over-used. The camera is connected to a small display that dangles just above the Pushpins, so it is easy to assess the health of the network at any time.

[3]SDCC supports a wide range of architectures including the Intel 8051, Maxim 80DS390, Zilog Z80, Motorola 68HC08, and Microchip PIC16 and PIC18 series.

To close this section, here is a brief summary of the major new infrastructure that has been developed during the course of this work:

- **IR spotlight** Global programming and control of the Pushpin network from a PC.
- **Gateway node** Local I/O port. Allows a PC to communicate as if it were a node on the network.
- **IR sensitive camera** Allows visual monitoring and debugging of the communication channel.
- **Open source development tool set** Cross-platform software infrastructure for programming, debugging, controlling, and querying sensor nodes.

2.3 Sensing Hardware for Localization

Using the localization system discussed in future Chapters, a sensor network deployed "in the field" could use global phenomenon detected in common across several nodes such as lightning strikes or exploding munitions to aid it in ad hoc localization. To test these algorithms on the Pushpin network, we needed to develop a similar stimulus (a flash of light followed by an acoustic pulse) on a much smaller scale. To this end we have built a device, which we refer to as the "Pinger." A Pushpin expansion module was developed to receive these signals and process them for localization. In this section, we describe the Pinger and the new ultrasound time-of-flight expansion module.

2.3.1 Ultrasound Time of Flight Expansion Module

The ultrasound time-of-flight (TOF) expansion module is a new sensing layer that was designed specifically for Pushpin localization experiments. It contains three sensors (a phototransistor, a sonar transducer, and an electret microphone) and one actuator (an RGB LED). Of these, the phototransistor and ultrasound transducer are used for localization, while the microphone and LED serve as additional I/O. The module is shown attached to a Pushpin in Figure 2-6.

The localization sensors (the phototransistor and ultrasound transducer) were chosen to complement the transmitters on the "Pinger" device (described below), which emits simultaneously a flash of light and a 40-kHz ultrasound pulse. When a Pushpin detects a flash, it starts a hardware timer that is stopped when it detects the sonar ping. The value of the timer divided by the speed of sound is taken as the measured distance to the Pinger. An analysis of the error characteristics of this measurement can be found in Section 5.2.

Detection of the camera flash is handled by the TOF module using a dedicated flash detector circuit that follows the phototransistor. This circuit, which is essentially a saturating high pass filter with a cutoff frequency of 2-kHz, outputs a digital pulse whenever a flash is detected. This output is wired directly into an external hardware interrupt on the Pushpin processing layer. The use of an external interrupt ensures that flash detection is given a high priority, but it allows the processor to focus on other tasks until an interrupt is received. However, if the processor wishes to read the unfiltered output of the phototransistor directly, it may do so using the ADC.

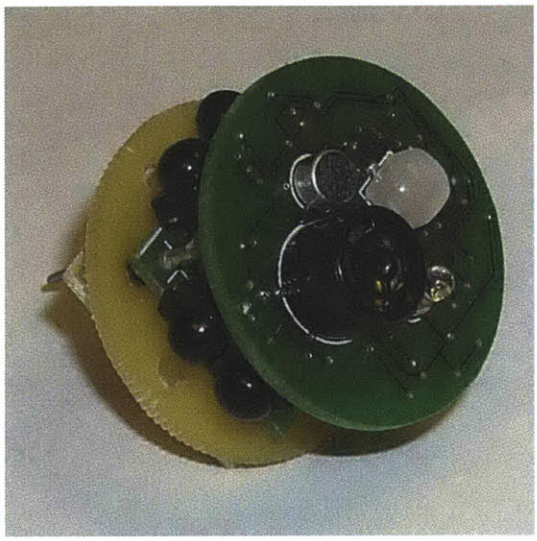


Figure 2-6: A pushpin with the Time of Flight (TOF) Expansion module shown attached to a Pushpin. The TOF module contains an RGB LED, microphone, photo-transistor, and sonar transducer. The last two of these are used to measure sonar time of flight for use in localization.

The sonar detection circuitry behaves in a similar manner to the flash detector. The sonar transducer is followed by a simple unipolar rising edge discriminator circuit that triggers an external hardware interrupt on the processing layer when the amplitude of the sonar signal exceeds a preset threshold. The value of the threshold can be adjusted using a DAC on the processing layer. Two versions of the sonar signal are available at the ADC for direct sampling: a raw version of the waveform directly from the sonar transducer and a low-pass

filtered version (with a cutoff frequency of 400-Hz) of the same signal.

The microphone is not used to aid in localization. Rather, it is intended as a general purpose sensor for use in a future localization-enabled application. For example, localized Pushpins could form an acoustic phased array for localizing the source of a sound in a room [13, 20]. Both the raw signal and an enveloped version of this signal (low pass filtered with a cutoff frequency of 87-Hz) are made available to the processing layer through the ADC.

The RGB LED is for general purpose I/O. It serves to display the current state of a Pushpin. This is an invaluable asset when debugging new code, since it enables the Pushpins to display their status at all times. The LED can also serve as a display element when a Pushpin is used as a "smart pixel" in a distributed display task. Refer to Section 5.3.1 for an example in which localized Pushpins were used to animate a vertical line "wiping" across the coordinate system. The intensity of each of the three LED colors is digitally controlled by the Pushpin processor using pulse-width modulation.

Complete schematics for the TOF module can be found in Appendix A.

2.3.2 The Pinger

The "Pinger" (pictured in Figure 2-7) is a handheld device that generates a simultaneous flash of light and burst of 40-kHz sonar. These signals are detected by the phototransistor and sonar receiver on the TOF Expansion modules. A Pushpin estimates its distance to the Pinger as the time difference of arrival of these two signals divided by the speed of sound in air (343.6 m/s at 20°C). To generate a global stimulus for localization, the Pinger is held somewhere (anywhere) above the Pushpin network and triggered with the press a button.

The Pinger is an artificial signal source, but it is meant to emulate a global sensor stimulus that might appear in a natural setting. For example, a thunder cloud also generates a bright flash of light that is accompanied by a propagating sound front. This is a familiar example of ranging using time-of-flight measurements because many of us were taught to count the number of seconds between a lightning strike and a

thunder clap when we were children. In a sensor network, it is as if all of the children in an area are sharing their observations over a radio. With a little help from their parents to do the math, they can figure out where they are relative to each other.

A thunder cloud might help localize a sensor network spread over the area of several square kilometers. Engineers at NASA Kennedy Space Center have demonstrated that localization using weather phenomena is at least a possibility by localizing lighting strikes using a sensor array spread over a several kilometer area [10]. Nonetheless, the thunder cloud is just one example of a natural phenomenon that generates a pair of signals with different propagation speeds. In principal, any pair of signals with coincident time origins and differing propagation speeds could be used to generate time-of-flight distance constraints. Munitions, fireworks, and gunshots are examples of phenomena that generate flashes of light and audible wavefronts. Signals also propagate at different speeds through different materials. For example, it might be possible to measure the distance to an explosion by detecting it both in the air and through the ground. Additionally, since the first pulse primarily serves to synchronize the sensor nodes, it should be possible to localize by measuring the time of arrival of a single signal if the clocks of the sensor nodes are synchronized by some other means.

Figure 2-7: The "Pinger" device simultaneously generates a flash of light and a 40-KHz ultrasound pulse at the push of a button. By measuring the time difference of arrival between these two signals and dividing by the speed of sound in air, a Pushpin node can measure its distance to the Pinger to within about 1-cm.

2.4 Pushpin Simulator

A proto-board is a useful tool for designing electronic circuits. Similarly, the Pushpin platform is a useful tool for developing algorithms for wireless sensor networks. However, rapid prototyping tools are best used in conjunction with simulations of the design where the mathematical and theoretical underpinnings of an algorithm can be thoroughly tested over a wide range of system parameters and inputs. When we implemented on the Pushpins our first localization algorithm that was based on lateration alone, it was simple enough that we were able to tune it up and test it without the aid of a simulator. However, as we began to explore complicated non-linear optimization methods such as spectral graph drawing and mesh relaxation, we quickly learned that we would need a simulator to help us choose the correct algorithm, debug its implementation, and tune its parameters.

The Pushpin Simulator is a means to this end. It is software for MacOS X written in Objective-C and Python meant to simulate a network of up to 100 Pushpin nodes distributed on a virtual 1-meter by 1-meter plane. Each virtual Pushpin is given its own memory for state and a thread of execution on the host machine via POSIX threads. Pushpin threads interact just as real sensor nodes would: by passing data packets to their nearest neighbors. In the Pushpin Simulator, this is accomplished using standard inter-process communication (IPC) mechanisms. Because this architecture closely resembles the distributed nature of real Pushpins, code written for simulated Pushpins is very similar to code for Pushpins in the hardware testbed. Pushpins in the simulator must collaborate, share state, and respond to sensor data by passing messages in exactly the same manner as real Pushpins would. In this respect, the Pushpin is more similar to the high-level Swarm[7] or HLSIM[2] simulators than lower level simulators such as NS-2[3], which focuses on accurate simulation of network protocols, or Avrora[65], which provides a cycle-accurate, instruction-level simulator for the Crossbow Motes.

Virtual Pusphins are fed "sensor" data in the form of sonar time-of-flight measurements – either real measurements recorded on the Pushpin hardware testbed or

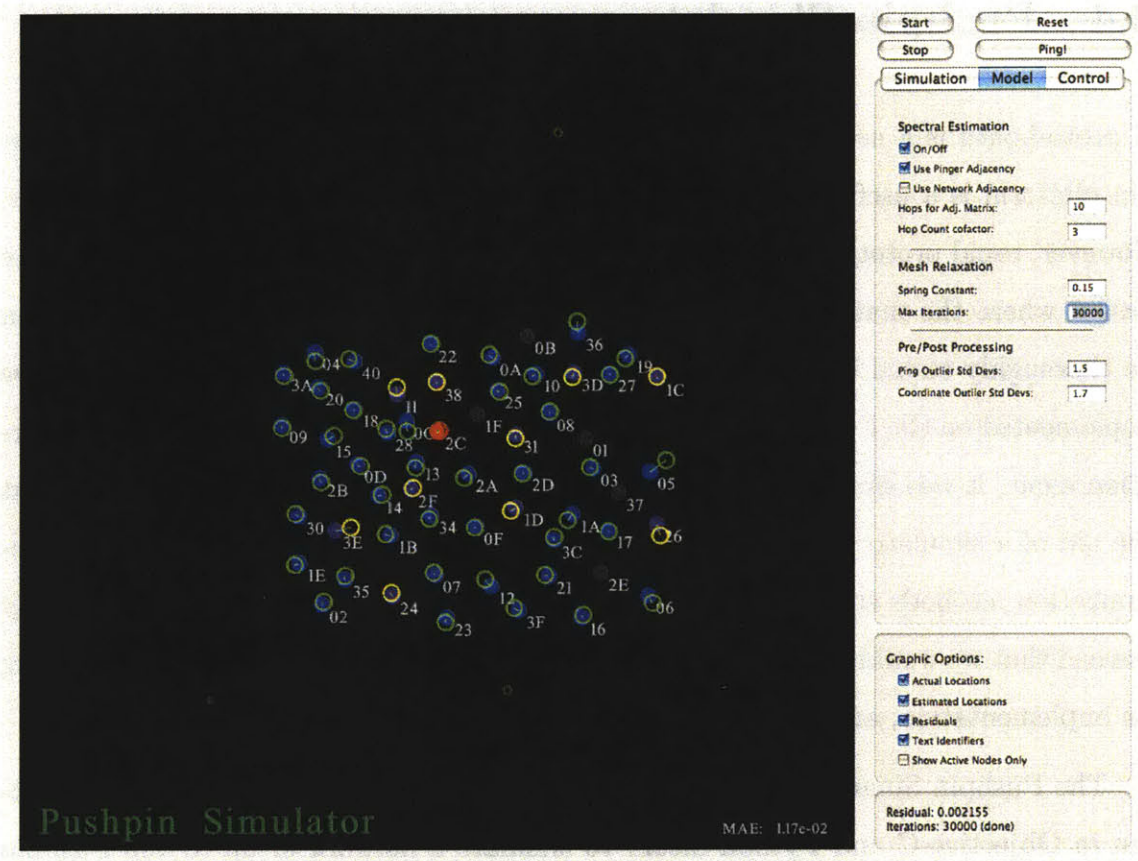


Figure 2-8: A screenshot of the Pushpin Simulator running a localization trial with 58 nodes.

simulated measurements produced by a statistical model based on the real measurements. Similarly, the placement of the simulated Pushpins in the virtual environment can be random, or it can be set to the actual "ground truth" positions of the real Pushpins that have been extracted from a digital photograph of the array. As a result, the simulator is capable of using completely realistic or completely simulated data. The degree to which the localization accuracies agree in the real and simulated scenarios indicates how well the statistical model of error in the simulated sensor measurements matches the actual error. We will discuss our statistical model and its accuracy in Chapter 5.

Figure 2-8 shows a screenshot of the Pushpin simulator. The primary display in the interface is a dynamic plot of the actual positions (closed circles) of the Pushpins and their locations as estimated by the localization algorithm (open circles). Thin lines

connect the actual and estimated positions for a given node. The graphical display is useful for monitoring the dynamics of the localization algorithm in ways that would be inconvenient or impossible on a hardware platform. Common localization errors such as skewing or folding of the coordinate system, instability and oscillation in the algorithm's convergence, or nodes with outlying coordinates are immediately obvious in the dynamic plot.

Graphical widgets for controlling the simulation appear to the right of the dynamic plot. Here the user can quickly adjust critical localization parameters, such as the spring constant of the mesh relaxation, the communication radius of the nodes, or the outlier rejection threshold for eliminating spurious errors in the time-of-flight measurements.

The Pushpin Simulator is a highly specialized piece of software designed to face the specific problem of Pushpin localization (namely, a network of roughly 50 nodes that are localized using range measurements to global phenomenon). Nonetheless, it is open source and freely available to anyone who wishes to use it for their own purposes [1]. At the very least, it is fun to experiment with different simulation settings to get a sense for how the spectral graph drawing and mesh relaxation algorithms behave under different circumstances. It could also be adapted as a framework for testing other localization algorithms.

2.5 A Brief Diversion: The Tribble

We now turn our attention away from localization to briefly discuss a project that is tied to the Pushpin research and our research vision of pursuing electronics skins; namely the Tribble (Pictured in Figure 2-9(a)). The Tribble is our version of sensor network skin. It is composed of the wired sensor nodes, each of which has a plethora of different sensor and actuator modalities. These give the Tribble a rich ability to sense and respond to changes in its environment. The behavior of the Tribble is driven by algorithms that are modeled after biological systems – it has been given a rudimentary nervous system of sorts that can respond directly to its stimulus. For example, we

have experimented with functions that emulate lateral inhibition and excitation in the human nervous system. Communication messages acting very much like chemical gradients or pheromones excite or inhibit the sensory responses of adjacent patches of Tribble skin. When one node on the Tribble experiences significant stimulation, it responds to that stimulation and sends out pheromone messages, thereby exciting its neighbors and causing them to mimic the response if they are also touched.

2.5.1 Hardware Overview

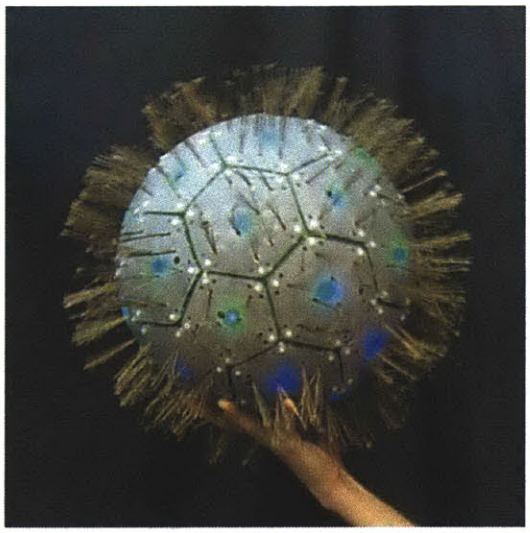
Geometrically, Tribble resembles a large soccer ball (i.e., an inflated version of a polyhedron known as the truncated icosohedron, which is composed of 20 hexagonal faces and 12 pentagonal faces). Each of the 32 faces can be considered as an individual 'patch' of skin. The patches screw into a plastic frame and can be individually removed and/or replaced at any time during Tribble operation. All of Tribble's processing capabilities reside in distributed form in these patches; there is no central controller or master patch.

Four NiCd D Cell batteries and accompanying voltage regulation circuitry are suspended at the center of the frame, providing approximately 5000mAh distributed among all 32 patches via a star configuration of RJ22 cabling emanating outward from the center. Tribble can also be powered by an external DC power supply. See Figures 2-9, frames (b).

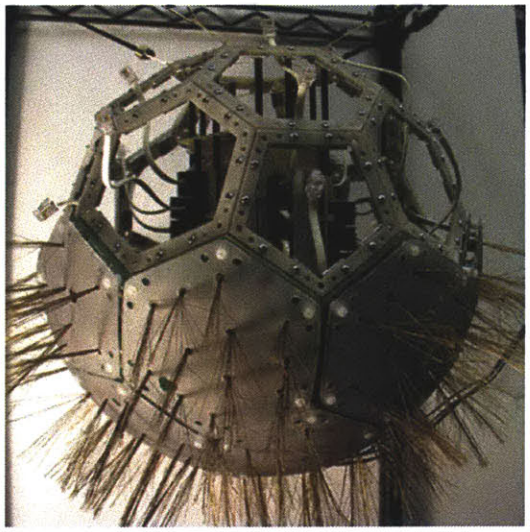
The same RJ22 cabling also provides a global communications bus as a means of programming and debugging all patches in parallel from a personal computer. The global bus is not otherwise used by the patches themselves during normal Tribble operation. Rather, the patches communicate neighbor to neighbor.

The same screws that mechanically secure a patch to the frame also provide a 115200kbps communication channel to each of the five or six neighboring patches via a direct electrical connection through conductive brackets fixed to the frame. These communication channels are fed through a multiplexer to the patch's 8-bit, 22MIPS microprocessor, located on the underside of each patch.

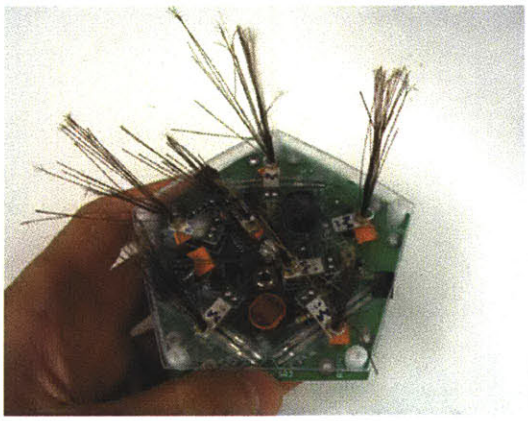
This microprocessor locally manages the patch's sensor data collection, actuator



(a) The fully assembled Tribble.



(b) A partially assembled Tribble showing the interior frame.



(c) Single Tribble panel with a clear shell.

(d) The Tribble on display at *Artbots: The Robotic Talent Show* in New York City

Figure 2-9: The Tribble is a sensor network organism. Each of its 32 panels of "skin" contains one Pushpin node that communicates with adjacent panels through hard connections in the interior frame. The hair is not just for show – each follicle is instrumented with its own piezoelectric sensor. These, as well as one sensor for light, three for pressure and one for audio on each panel, imbue the Tribble with a generous sensor bandwidth that is only possible because of the distributed nature of the system.

response and communication with neighboring patches. For actuation, each patch has at its disposal a vibrating pager motor, a RGB LED, and a small speaker. As for sensing, each patch is equipped with 7 (pentagonal patches) or 12 (hexagonal patches) piezoelectric whisker sensors, three force sensitive resistor (FSR) pressure sensors (enabling determination of the magnitude and center of applied force), a solid-state temperature sensor, a light dependent resistor (LDR) light sensor, and an electret microphone.

A hard, diffusive polyethylene shell covers the exposed side of each patch. The whiskers consist of nylon/polyester paintbrush bristles extending from a piezoelectric cantilever and protruding from the protective shell. Polyurethane foam transduces pressure applied on the shell to the FSRs below. See Figure 2-9 frame (c).

All told, Tribble has 516 channels of 10-bit sensor input being sampled at approximately 1000Hz per channel, on average (actual sampling rates depend largely on the sensor being sampled.) Thus, the Tribble as a whole has approximately 5-Mbits/second of sensory bandwidth. (An empirically tested upper bound of Tribble sensory bandwidth is actually closer to 22-xsMbits/second, but this is impractical as there would be no CPU resources left over to process the data.)

2.5.2 Electronic Skin

We hope to use the lessons learned from building the Tribble to continue electronic skin research by developing a small "patch" of electronic skin that contain 100 nodes in a 10-cm by 10-cm square area. Like the Tribble, each of these nodes will itself be equipped with an array of pressure sensors as well as individual sensors for modalities such as temperature, light, sound, and proximity. A network of this size and density will truly begin to realize the sensing densities and modalities of human skin.

In the meantime, we plan to continue developing the Tribble for both research and exhibition. In 2003, the Tribble traveled to New York City to be featured in *Artbots: the Robotic Talent Show* (see Figure 2-9 (d)). At Artbots, people of all ages, particularly children, were attracted by the interesting, evocative appearance of a whisker-covered ball. In general, anyone who passes the Tribble is tempted to touch

it and learn how it responded to their stimulus. Even the simplest sensing modalities and response models generate excitement and empathy in passers-by. Furthermore, once we have someone's attention, we are able to explain the Tribble's distributed nature, and we have found most people to be fascinated and receptive to the idea of an electronic skin. The Tribble has been a continuing attraction for sponsors and visitors at the MIT Media Lab, and we hope to continue to develop its behaviors for future art installations, demonstrations, and electronic skin research.

Chapter 3

Localization Algorithms

This chapter provides mathematical descriptions of the three algorithms employed in the Pushpin localization system: lateration, spectral graph drawing, and mesh relaxation. The aim here is to summarize these techniques and point out specific features that we have found relevant to our work.

The following discussion of localization *algorithms* has been purposely kept separate from the discussion of localization *systems*, which is the subject of the next chapter. While localization algorithms can be described in succinct, mathematical terms, a description of a localization system summarizes the architecture, design choices, and implementation details of a system for localizing sensor nodes. A localization system has many interacting components of which localization algorithms are just one part. Nonetheless, the localization algorithms are by far the most important, hence they are given special treatment in this chapter.

In addition to describing the algorithms themselves, we discuss how results of a localization algorithm are interpreted; specifically, how the resulting coordinates can be translated into a reference frame of an external observer, and how they can be aligned with a set of "ground truth" coordinates to assess localization error.

Before we jump into descriptions of the algorithms and their results, let us develop some terminology that will help to frame future discussion. We start by formulating the basic sensor network localization problem.

3.1 The Basic Problem

Consider a set of n sensor nodes, each of which has a set of 3-dimensional coordinates[1]. Adopting graph theoretic terminology, we can call a sensor node a *vertex* in a graph. The ith vertex in the graph is denoted V_i, and it is associated with a 3-dimensional vector of coordinates $\mathbf{X}_i$.

When the sensor network is first deployed, few (if any) of the vertex coordinates are known. However, sensor nodes are uniquely equipped to communicate and collaborate with their neighbors, hence a node is able to constrain its position by measuring the distances to neighboring nodes and sharing these measurements over the network. Just as ionic and covalent bonds hold atoms in a molecule rigidly in place, the distance constraints generated in this manner serve to draw the coordinates of sensor nodes into the correct physical layout. Distance constraints are *edges* in the graph model. An edge that connect two vertices V_i and V_j is said to have a *weight* w_{ij} related to the distance between the nodes. Most often, this weight is the actual physical distance between two nodes plus some error due to the measurement process.

Sometimes, rather than measuring the distances between nodes, it is more convenient to measure the distance from a node to a distinct global sensor stimulus in the environment surrounding the sensor network. Consider a lightning strike and crash of thunder. Such a phenomenon could be detected by multiple nodes distributed over a large area. Nodes within range of the lightning strike independently measure their distance to the source of the strike by counting the number of seconds between when they observe lightning and thunder. The location of a global event like the lightning strike is called a *global point of correspondence.* It is a piece of knowledge that is shared by multiple nodes in the network. However, a global stimulus has identical properties to the vertices already in our graph model; namely, it has a distinct set of coordinates and it is connected by edges to one or more other vertices. Therefore, for the remainder of this chapter, a vertex may refer to either a sensor node or to a

[1]Using four dimensions rather than three is a potential way to include a coordinate for time at each node, which may be useful for simultaneously localizing nodes in space *and* time. Section 6.1 discusses the possibility of combining localization and synchronization in more detail.

distinct global sensor stimulus.

Having established definitions for vertices and edges in the context of a sensor network, their physical meanings can be temporarily set aside as we formulate the following optimization problem in more abstract terms. The basic localization problem can be summarized as follows:

> Given only a set of edge weights $W = \{w_{ij}\}$ between pairs of n vertices in the set $V = \{V_i\}$, find a set of vertex coordinates that satisfy the following Euclidean distance constraints.

$$||\mathbf{X}_i - \mathbf{X}_j||^2 = w_{ij}^2 \; \forall w_{ij} \in W \tag{3.1}$$

Here, $||\cdot||$ is the standard Euclidean norm. For the sake of localization, where the goal is to find coordinates that resemble the correct physical layout of the sensor nodes, we hope to find a solution to this system that is *rigid*. That is, it must be unique up to an arbitrary translation, rotation, and possible reflection. This will naturally depend on having sufficient edge weights to constrain the problem. Unfortunately, it is difficult to make a statement about exactly how many edge weights are required. However, in the Pushpin scenario with global stimuli, it is possible to place a lower bound on the minimum number and type of edge constraint that is necessary before a rigid solution *may* exist. In the next section, we develop this bound.

3.1.1 Finding a Unique Solution

As discussed in Chapter 2, edge weights between the Pinger events and the Pushpin nodes are ultrasound time-of-flight (TOF) measurements made by the time-of-flight expansion module. The following analysis based on TOF measurements is similar to the analysis carried out by Moses et. al. in [43], though their analysis was more complex because it also considered time-of-arrival (TOA) and angle-of-arrival (AOA) measurements.

For a graph with n vertices, there will be $3n$ unknowns (for 3-D coordinates), hence

we will be seeking a point in $3n$-dimensional space that solves the system of edge-constraint equations in 3.1. In the extreme case, if edges are known between every pair of vertices, the system will consist of $\binom{n}{2}$ edge constraint equations. However, in a sensor network, n is typically large enough that $\binom{n}{2} \gg 3n$. In other words, the plethora of potential equations greatly outstrips the number of unknown coordinates. This suggests that a rigid solution to the localization problem can most likely be found if a small subset of all of the possible edge weights are known. At least $3n$ equations will be needed to find the solution, though more may be necessary since the system is non-linear. Therefore, the number of equations (created by new edge weights) needed to find a unique (rigid) localization solution has a lower bound of 3 times the number of vertices in the system.

Now consider the Pushpin localization scenario in which vertices represent either sensor nodes or global sensor events. Suppose that there are j Pushpin nodes in the network capable of participating in a localization algorithm. These nodes observe k distinct Pinger events. The goal of the localization algorithm is to find $3(j+k)$ coordinates for the sensor nodes and Pinger events.

For now, assume that the Pinger events are global and error free; i.e. each sensor node measures an edge weight between itself and the Pinger location for each global event. Therefore, j new edge weights - one for each node in the network - are measured for each Pinger event. After k Pinger events, there are jk known edge weights (and jk equations). The solution to the system of equations *may* be found when the number of equations meets or exceed the number of unknowns. That is, the lower bound on the number of edge weights can be expressed as,

$$3(j+k) \leq jk$$

Manipulation of this equation yields,

$$j \leq \frac{-3k}{3-k} \tag{3.2}$$

Equation 3.2 implies that there is a tradeoff between the number of global events

used to generate constraints and the number of participating nodes. Add additional global events from the Pinger, and the number of sensor nodes required to participate in localization is reduced. Similarly, additional nodes result in the need for fewer global events. Figure 3-1 shows this relationship over a range of values for j and k. The plot asymptotically approaches 3 nodes on the vertical scale and 3 global events on the horizontal scale, suggesting that *at least* 4 of each is required to find a rigid localization solution in 3-dimensions. In order to localize using 4 Pinger events, at least 13 nodes must participate in localization, and vice versa.

Based on this analysis, we chose to use 5 pings and 10 nodes for most of our tests. While this resulted in consistent localization solutions, we found that the solutions were not actually rigid as we had desired. Specifically, we found that they were invariant under a skewing transformation of the coordinate system. We delve into this phenomenon in more detail in Chapter 5.

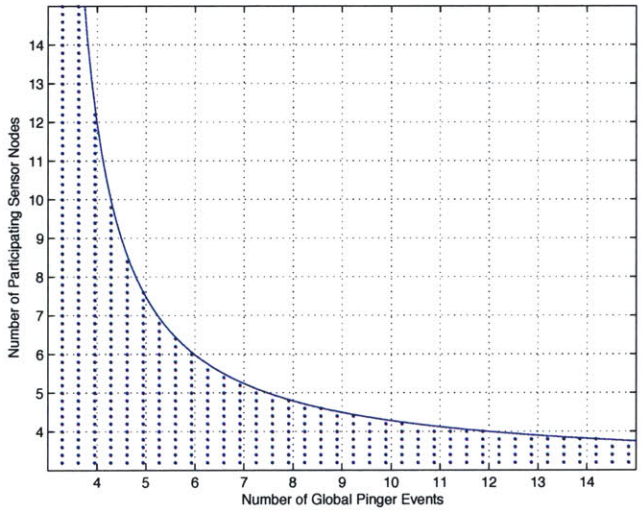


Figure 3-1: There is an inherent trade-off between the number of nodes participating in localization and the number of global Pinger events required to find a localization solution. The line represents a lower bound on the number of Pushpins and Pinger events required for a solution to exist. The actual number needed for a rigid solution falls somewhere outside of the shaded region.

3.1.2 Anchor Nodes

In a practical sensor network deployment there may be some nodes with *a priori* knowledge of their position. They may be equipped with special localization hardware to advantage of existing localization infrastructure such as GPS or the MIT Cricket system [55], or they may be preprogrammed with their coordinates by hand or by a robotic deployment system [39]. Nodes that come with prior knowledge of their coordinates are commonly referred to as *anchor nodes* or *beacons*.

When they are available, anchor nodes provide powerful constraints that greatly simplify the localization problem. For example, if a node knows the distance between itself and three or more non-colinear anchor nodes, it can triangulate its 2-D position by solving a linear system of equations using a technique called lateration (see Section 3.2). A 3-D position can be computed in this fashion with four or more such anchor nodes.

The advantage of using anchor nodes is obvious – just a handful of well placed anchor nodes can greatly simplify the localization problem by adding valuable additional constraints. Furthermore, if anchor nodes are programmed with coordinates in a *global* reference frame external to the sensor network, they will constrain the localized coordinate system to line up with the global coordinate system, thereby eliminating the need for a transformation between reference frames inside and outside the sensor network.

Anchor nodes have certain disadvantages as well. In order to infer its position, an anchor must be equipped with specialized equipment to help it determine its coordinates, such as a GPS receiver or a highly specialized, accurate sensor or actuator like a laser range finder or ultrasound transmitter. Localization researchers tend to avoid such additions, since they can hamper the scalability, low-cost, and universal utility that sensor nodes gain by being identical and therefore interchangeable and easily mass produced. Specialized "heavy-weight" nodes with extra capabilities are thought to be taboo due to the cost and the complexity a second type of node adds to the system. Several researchers have shown that the overall accuracy of coordinates

derived using anchor-based localization algorithms are highly dependent on the placement, distribution, and number of anchor nodes . In particular, it has been shown that localization accuracy for nodes outside the convex hull of the anchor nodes can be particularly low[43, 49, 57].

Despite the aforementioned disadvantages, anchor nodes appear in some form in the majority of the localization algorithms in the sensor network literature. This is likely because the use of anchors greatly simplifies localization, enabling the use of simpler, smaller, faster algorithms for computing the position of unlocalized nodes. For a sensor node with Roughly 10-MIPS of processing power and several thousand kilo-bytes of RAM, the simplicity of the localization algorithm is a very important consideration. For example, a typical anchor-based localization algorithm is the APIT algorithm proposed in [32]. In APIT, unlocalized nodes determine whether they are within various bounding triangles created by random triplets of three anchor nodes. They determine their coordinate as the centroid of the intersection of these bounding triangle, which is easy to compute. APIT is similar to but more sophisticated than the *min-max* method, where the bounding shapes are boxes centered at the anchor coordinates and sized according to a node's range from an anchor [58, 40]. In min-max, determining the intersection of the squares from many anchors can be accomplished by simple pairwise comparisons of the bounds of the squares.

The most common anchor-based localization technique is lateration, which is similar to triangulation. While lateration is not as computationally simple as APIT and min-max, it is still possible to run lateration on a microcontroller with modest processor specifications. Furthermore, it yields more accurate results and requires fewer anchor nodes. Due to its importance in Pushpin localization, a dedicated description of lateration appears in the next section. For a general discussion and comparison of these and other anchor-based localization techniques, the interested reader should consult [40].

3.2 Linear Algorithm: Lateration

Lateration is a linear localization algorithm for triangulating the position (x, y, z) of a vertex V in 3 dimensions given the positions (x_i, y_i, z_i) of at least 4 other "anchor" vertices and the measured distances d_i between V and these other vertices[2]. In the Pushpin localization model, vertices can represent other sensor nodes or the locations of global events generated with the Pinger. Measured distances are ultrasound time-of-flight measurements.

Given the positions of and distances to the anchor vertices, we can apply a basic Euclidean norm to form a system of equations. In three dimensions,

$$\begin{aligned}
(x_1 - x)^2 + (y_1 - y)^2 + (z_1 - z)^2 &= d_1^2 \\
(x_2 - x)^2 + (y_2 - y)^2 + (z_2 - z)^2 &= d_2^2 \\
(x_3 - x)^2 + (y_3 - y)^2 + (z_3 - z)^2 &= d_3^2 \\
(x_4 - x)^2 + (y_4 - y)^2 + (z_4 - z)^2 &= d_4^2
\end{aligned}$$

All quantities except (x, y, z) are known. The system can be linearized by subtracting the fourth equation from the first three and collecting terms:

$$\begin{aligned}
2(x_1 - x_4)x \;+\; & 2(y_1 - y_4)y + 2(z_1 - z_4)z \\
&= x_1^2 - x_4^2 + y_1^2 - y_4^2 + z_1^2 - z_4^2 + d_4^2 - d_1^2 \\
2(x_2 - x_4)x \;+\; & 2(y_2 - y_4)y + 2(z_2 - z_4)z \\
&= x_2^2 - x_4^2 + y_2^2 - y_4^2 + z_2^2 - z_4^2 + d_4^2 - d_2^2 \\
2(x_3 - x_4)x \;+\; & 2(y_3 - y_4)y + 2(z_3 - z_4)4 \\
&= x_3^2 - x_4^2 + y_3^2 - y_4^2 + z_3^2 - z_4^2 + d_4^2 - d_3^2
\end{aligned}$$

This system has the form $A\mathbf{x} = \mathbf{b}$, and it can be readily solved by inverting A. If there are more than 4 anchor nodes, the system will be over-determined and a linear

[2]In general, lateration can be generalized to find coordinates in k dimensions given $k + 1$ anchor nodes.

least square estimate will yield the optimal solution: $\mathbf{x} = (A^T A)^{-1} A^T \mathbf{b}$.

Lateration has been thoroughly explored in recent sensor network localization research[40, 58, 57]. In particular, it has been shown via computation of the Cramer-Rao bound, which establishes the lower bound on the error of an minimum-variance unbiased estimator (i.e. the most ideal estimator possible for a given measurement error model), that localization accuracy degrades if the node is outside of the convex hull of the anchor points [49, 57]. This implies that nodes very near the edge of the network will be localized with lower accuracy, though adding additional anchors does appear to help somewhat.

3.3 Non-Linear Algorithm: Spectral Graph Drawing and Mesh Relaxation

Due to its simplicity, lateration is a good localization algorithm to choose when there are enough anchor nodes to make it accurate. However, some sensor network deployments do not have anchor nodes at all, and even those that do must localize the anchor nodes prior to localizing the remaining nodes. Barring the use of some external infrastructure, the anchor nodes, if not the entire network, must localize using distance constraints alone. This is the situation described by a system of equations like Equation 3.1. The solution to this problem can not be obtained directly, since this system is non-linear. Instead, the solution must be computed an iterative non-linear optimization approach.

Non-linear optimization is a burgeoning field dedicated to solving, or at least trying to solve, non-linear problems that do not have directly computable, closed-form solutions. In order to solve such problems, it is often effective to minimize a "fitness function" – a many-to-one mapping from the parameter space of the problem to a one-dimensional space – whose extrema correspond to interesting points in the system. In particular, the fitness function is carefully chosen so that the global extremum corresponds to the solution to the problem at hand. One can imagine that the fitness

function creates a many dimensional surface or "optimization landscape" with many peaks and troughs, some higher and lower than others. The correct solution to the problem lies at the highest peak or deepest trough. To find these global extremum, the landscape must be traversed one point at a time.

This problem may sound deceptively simple, but it is quite challenging due to the enormous size of the space that must be searched. Techniques for traversing this landscape in search of the global extremum are numerous. These range from always moving downhill (gradient-based methods) to jumping around on the landscape randomly, but generally in the downhill direction (stochastic methods such as simulated annealing or genetic algorithms). Sometimes the structure of the problem will enable an educated guess about which direction is the best or how large a step to take (semi-definite programming or convex optimization), or the problem may emulate the behavior of a physical system which has an intuitive solution (ball-spring model such as mesh relaxation). Each technique has its own strengths and weaknesses. However, for localization in a sensor network, the following characteristics are most important.

1. The algorithm must converge reliably to the global extremum in the optimization landscape (this is the point that corresponds to coordinates that are arranged in the same shape as the actual sensor nodes).

2. The algorithm must be easily distributed and scalable when run on the nodes themselves (preferably only requiring pairwise communication exchanges with immediate neighbors so as not to flood the network).

3. The algorithm must converge in a reasonable amount of time and minimize the energy expended on computation and communication.

We have chosen to perform non-linear optimization using mesh relaxation (MR); a simulation of a physical system of masses (which represent vertices in the graph) connected by springs (which represent edges). The position of the vertices is initially unknown, so they must be initialized either randomly or with an educated guess of the correct coordinates. The tension of a spring is calculated as the difference between

the current estimated length between two vertices (computed using the Euclidean norm between the current best estimated of the coordinates of two nodes) and the measured or ground truth distance between two nodes (e.g. an ultrasound time-of-flight range measurement). This value is then multiplied by a scaling value called the *spring constant*. Just as a real spring relieves tension by exerting a force on the objects it connects, the estimated positions of the vertices in the mesh relaxation model move in the direction that brings the estimated and measured distance more closely in line. When overall tension, or *stress*, on the mesh approaches zero, the estimated position of the vertices match their measured position up to a translation, rotation, and possible reflection.

Unfortunately, mesh relaxation does not always generate the correct localization solution on its own. If the initial vertex coordinates are assigned at random, the algorithm will converge to a false minimum with high probability. Priyantha et. al. [54] observe that these false solutions often correspond to a folding of the graph, where one or more vertices will be given coordinates that are reflections across some line (in 2-D) or plane (in 3-D) from the correct coordinates. We have observed similar behavior in our mesh relaxation model. Convergence to a false minimum is unacceptable for a robust localization system. In order to avoid this behavior, we sought a technique that would help us to avoid graph folding in mesh relaxation.

Since local minima plague most non-linear optimization problems, many techniques have been developed for avoiding them. One standard method involves adding inertia to the vertices so that they tend to move out of small local minima on their way downhill toward the global minimum. Stochastic methods such as simulated annealing also avoid local minima by adding a sort of "kinetic energy" to the system that diminishes over time [53]. However, the best method of all is to start the optimization at a point in the optimization landscape that is closer to the global extremum than any other extrema. If such a point can be found, it leaves very little work for the optimization algorithm, which does not have to travel far to find the correct solution. Seeding mesh relaxation with such a point virtually guarantees that the global minimum corresponding to the correct localization solution will be found.

Other researchers have used an initial guess to seed a mesh relaxation algorithm. Howard et. al. demonstrate a localization system on a mobile robot platform where inertial measurements are used for rough positioning, then refined with mesh relaxation[34]. Priyantha et. al. propose a set of heuristics for building a fold-free initial guess in a sensor network in their anchor-free localization (AFL) algorithm[54]. They found it easier to create a fold-free approximation using network hop count metrics, then to apply this as an initial guess to be refined by mesh relaxation.

Another alternative is a technique adapted from algebraic graph theory called spectral graph drawing, which was recently introduced to the sensor network community by Koren[37] and Gotsman[29]. We have adopted spectral graph drawing in Pushpin localization due to its strong theoretical underpinnings; eigenvector calculations are very well understood and relatively fast and the conditions in which they do and do not converge to a solution are well understood and easily detected. Also, while we have not implemented it in our work, Gotsman has presented a fully distributed implementation of spectral graph drawing that would be ideal for use in a sensor network. We now turn the discussion to a more detailed description of spectral graph drawing.

3.3.1 Spectral Graph Drawing

Spectral Graph Drawing (SGD) is a technique for producing a set of vertex coordinates k dimensions (where you specify k) given only a set of edge lengths that connect the vertices. The resulting coordinates closely adhere to the constraints imposed by the edge lengths. Like multi-dimensional scaling, force-directed graph drawing, and principal component analysis, SGD was conceived as a technique to help visualize high dimensional data in a low dimensional space. The technique itself is quite old, dating back to the work of Hall [31] in 1970. However, it has seen little use since that time and has only recently been introduced as a technique potentially useful in sensor node localization by Yehuda Koren[37] and Craig Gotsman[29].

If the distance constraints of our localization problem are used as edge weights for SGD, the algorithm produces a set of positions that closely resembles the correct

sensor node layout. The SGD-produced coordinates are not exact, however. SGD solves a problem that is related to, but fundamentally different from the localization problem we have formulated so far. Like many graph drawing techniques, SGD is meant to produce "aesthetically pleasing" drawings in which the coordinates of the vertices are evenly distributed within a bounded area and spaced relative to the size of the edge weights. In SGD, the size of this bounding area is adjustable, and the edge distances are not precisely matched. The full localization problem, on the other hand, aims to find a set of coordinates that match the distance constraints *exactly*. However, while the SGD-produced coordinates are not exact enough to be used directly in localization, they are the perfect initial guess for an optimization technique such as mesh relaxation. With a good initial guess, mesh relaxation will avoid false minima and settle quickly to the correct solution.

Like the localization problem formulated above, the SGD problem can be posed as an optimization problem. The advantage of using spectral graph drawing as opposed to another optimization formulation is that the SGD problem has been proven to be *equivalent* to another problem which is much better understood: an eigenvector computation on a matrix. Hence, we are able to solve a particular optimization problem that is relevant to localization by solving a much simpler problem in linear algebra. This is the power of spectral graph drawing.

In the remainder of this section, we present a basic overview of SGD and summarize some of its important features. However, SGD is too complex for us to fully do it justice here. Many additional details can be found in the original papers by Koren[37] and Gotsman[29].

SGD Overview

A sensor network can be described abstractly as a set of vertices $V = \{V_1, ..., V_n\}$ and edges $E = \{\langle i, j \rangle\}$ that make up a graph $G(V, E)$. Edges have associated weights w_{ij} that are proportional to the *adjacency* of two vertices V_i and V_j. The weight is larger if the vertices are more closely connected (e.g. if they are physical closer to each other). If two vertices are not connected, $w_{ij} = 0$. It is significant that adjacency is inversely

related to distance. In practice, Koren recommends that a measured distance between two nodes can be converted to an adjacency weight via $w_{ij} = -exp(d_{ij})$ or $w_{ij} = \frac{1}{1+d_{ij}}$ (though we used the former in our implementation)[37].

We can summarize the connectedness of the graph by placing these weights in an *adjacency matrix* A, where

$$A_{ij} = \begin{cases} 0 & i = j \\ w_{ij} & i \neq j \end{cases}$$

Next, we define the *degree* of a node to be the sum the weights between itself and other nodes:

$$deg(i) = \sum_j w_{ij}$$

The values of $deg(i)$ are placed into a diagonal matrix D such that $D_{ii} = deg(i)$.

These definitions allow us to formulate a 1-dimensional graph drawing problem. Specifically, we would like to find an n-dimensional vector called x that contains the coordinates of n vertices by solving the following constrained optimization problem:

$$\arg\min_{\mathbf{x}} S(\mathbf{x}) = \arg\min_{\mathbf{x}} \sum_{<i,j>\in E} w_{ij}(x_i - x_j)^2 \tag{3.3}$$

$$given : \mathbf{x}^T D\mathbf{x} = 1, \mathbf{x}^T D1_n = 0$$

where 1_n is a vector of length n that contains only 1's. There are two forces at play in this minimization problem. Minimizing $S(\mathbf{x})$ tends to shorten the lengths between the vertices proportionally according to the weights w_{ij}, thereby pulling the graph into the correct shape. At the same time, the constraint that $\mathbf{x}^T D\mathbf{x} = 1$ provides a repulsive force, preventing this minimization process from collapsing into a degenerate solution in which all edge lengths are all equal to zero. The second constraint, $\mathbf{x}^T D1_n = 0$, removes translational ambiguity from the solution vector by forcing $\mathbf{x}$ to be zero-centered.

The presence of D in the constraints achieves what is called *degree normalization* of the solution vector $\mathbf{x}$. Without it, a vertex that has a much lower degree than the rest (if it had missing measurements or had fewer constraints to begin with) has

significantly less attractive force acting on it in the optimization. As a result, it will be overly separated from its neighbors when its final coordinates were computed, and the remainder of the nodes will be closely clustered near the origin. A degree normalized solution corrects for this, adding an extra repulsive force to nodes with higher degrees, thus preventing them from bunching up at the origin while one vertex with a substantially lower degree is left as an outlier. Degree normalization is one of Koren's primary contributions in [37]. It is essential for creating a well-distributed layout of vertices that is useful as an initial guess for mesh relaxation.

The power of spectral graph drawing lies in the fact that this minimization problem has a very convenient solution: *The vector of coordinates* $\mathbf{x}$ *that minimizes equation (3.3) according to the given constraints is the eigenvector* v_2 *associated with the* **second** *largest eigenvalue of the matrix*

$$Z = \frac{1}{2}(I + D^{-1}A) \tag{3.4}$$

A lengthy, though not complicated, proof of this can be found in the appendix of [37]. One notable detail, however, is that 1_n is the eigenvector associated with the largest eigenvalue of Z. The method for finding coordinates in two dimensions is identical except that $\mathbf{x}$ is forced to be orthogonal to both 1_n and v_2. In this case, the solution is the eigenvector v_3 that is associated with the third largest eigenvalue of (3.4). Coordinates in three dimensions can be found if $\mathbf{x}$ is forced to be orthogonal to 1_n, v_2, and v_3, and so on.

In essence, the problem of determining a geometrical layout for a set of vertices has been reduced to an ordinary eigenvector computation. Many well-understood algorithms exist for finding eigenvectors. In this case, power iteration [22] is an appropriate choice, since it is convenient to apply the constraints at each step in the iteration ($\mathbf{x}^T D\mathbf{x} = 1$ via normalization, $\mathbf{x}^T D 1_n = 0$ via Gram-Schmidt Orthogonalization). Koren's spectral graph drawing algorithm with degree normalization is based on this scheme. Pseudocode for this technique appears in Algorithm 1.

Algorithm 1 Spectral Graph Drawing (adapted from [37])

function SpectralGraphDrawing(A - adjacency matrix , k - dimensions)
% Computes $u_2, \cdots, u_k$, the top (non-degenerate) eigenvectors of $D^{-1}A$

$\epsilon \leftarrow 1.0e-9$ *% Desired tolerence*
for $i = 2 : k$ **do**
  $u_i \leftarrow random$
  $u_i \leftarrow \frac{u_i}{||u_i||}$

  repeat
    $\hat{u}_i \leftarrow u_i$;

    % D - orthogonalize against previous eigenvectors
    for $j = 1 : i-1$ **do**
      $u_i \leftarrow u_i - \frac{u_i^T D u_j}{u_j^T D u_j} u_j$;
    end for

    % Perform the power iteration step
    $\hat{u}_i \leftarrow \frac{1}{2}(I + D^{-1}A)u_i$

    % Normalize
    $\hat{u}_i = \frac{\hat{u}_i}{||\hat{u}_i||}$
  until $\hat{u}_i^T u_i < 1 - \epsilon$ *% Halt when direction change is negligible.*
end for
return $u_2, \cdots, u_k$

Distributed Spectral Graph Drawing

An alluring property of spectral graph drawing is that it can be formulated as a fully distributed algorithm that requires only neighbor-to-neighbor communication transactions. Gotsman and Koren demonstrate in [29] that the power iteration technique is mathematically equivalent to directing a node to repeatedly move its estimated coordinates to the centroid of the estimated coordinates of its neighbors. This technique, sometimes referred to as the *diffusion* technique for localization[11], has also been developed by Balusu et. al.[17]. However, these researchers do not explicitly draw a connection to spectral graph drawing in their implementations, and therefore do not benefit from the deeper theoretical insight of the more mathematical formulation given by Gotsman and Koren.

Gotsman and Koren demonstrate the distributed SGD implementation in a simulated sensor network containing 1000 nodes. Localization proceeds in two phases: (1) Distributed spectral graph drawing is used to obtain an initial guess for optimization. (2) Optimization is carried out using a majorization technique (this is the same technique that performs the optimization step in metric multi-dimensional scaling [15]). Gotsman and Koren show that spectral graph drawing outperformed both random node placement and Priyantha's heuristic-based anchor-free localization (AFL) technique[54] for generating an initial guess for optimization.

Gotsman and Koren have demonstrated distributed SGD to be both scalable and accurate, however their technique is currently limited to finding coordinates in two dimensions. The process of producing 3-D coordinates in a distributed setting has not yet been fully developed. In particular, no elegant technique has been proposed for distributing the Gram-Schmidt orthogonalization and ensuing normalization steps that prevent the loss of numerical precision in limited precision arithmetic. Further work on this aspect of the algorithm will be required if distributed SGD is to become widely used as a technique for sensor network localization.

3.3.2 Mesh Relaxation

Mesh relaxation simulates a physical system of masses connected by springs. The masses, which represent sensor nodes in this instance, have coordinates in three dimensional space. They are connected by springs that represent distance constraints: a range measurement made by a sensor to a global stimulus. A global value called the spring constant determines the stiffness of the springs and is the primary means of controlling the speed of convergence of the algorithm. A higher spring constant leads to stiffer springs and faster convergence.

Mesh relaxation is well suited for a distributed system since information need only be passed between nodes that are connected with a spring constraint. Many sensor network localization scenarios depend entirely on neighbor-to-neighbor constraint ([61, 69] for example), and in these cases mesh relaxation would require only neighbor-to-neighbor communication transactions. This is the best possible information passing scenario since it incurs the least communication overhead. However, in a system like the Pushpins that uses distance constraints to global sensor events, the updated estimated of the global event's location must be broadcast to the entire network at every step in the relaxation. In this case, it is better to run a centralized version of mesh relaxation that runs on a small subset of nodes so as to minimize network overhead. Such a system is described in Section 4.2.

Mesh Relaxation Overview

Consider a graph, or mesh, with n vertices. Each vertex V_i has a 3-D vector of estimated coordinates $\mathbf{X}_i[t]$. This estimate is incrementally improved during each discrete time step t. Certain distances have been measured between nodes. Each vertex V_i retains a set of measurements between itself and other nodes in the mesh, $D_i = \{d_{ij}\}$. Estimated coordinates can initially be chosen randomly or they can be selected using the results of the spectral graph drawing technique in section 3.3.1 (or by some other technique for generating a good initial guess such as inertial measurements integrated over time[34] or an approximate coordinate system built using logical distance over

the network [54]).

During each discrete time step in the relaxation process, every node computes the *force* acting on it due to the constraints imposed by the positions and relative distances of its neighbors.

The force due to each neighbor is proportional to the difference between the estimated distance (i.e. the norm of the difference between two sets of estimated coordinates) and the measured distance to the neighbor. This is a vector with a magnitude and direction equal to

$$\begin{aligned} |\mathbf{f}_{ij}[t]| &= k(||\mathbf{X}_i[t] - \mathbf{X}_j[t]|| - d_{ij}) \\ \angle\mathbf{f}_{ij}[t] &= \angle(\mathbf{X}_i[t] - \mathbf{X}_j[t]) \end{aligned}$$

In the equation above, $||\cdot||$ is the Euclidean norm and k is the spring constant that controls the speed of convergence of the algorithm. We found that the value of k must be tuned depending on the number of vertices and the lengths of the distance constraints. If k is too large, the system will be unstable and oscillations will result. If it is too small, convergence will be slow. We found that $k = 0.15$ was appropriate in our simulation with 15 vertices (10 anchor nodes and 5 pings) and average distance constraints of 1.0-m. It is important to point out that the sensitivity of k to the number of vertices and edges would certainly be a drawback to mesh relaxation in a system that does not have a fixed number of vertices and edge constraints, as we do.

The total force acting on a vertex is the sum of these individual forces; $\mathbf{F}_i[t] = \sum_j \mathbf{f}_{ij}[t]$. A node's new coordinate estimate is the sum of the old estimate plus this force; $\mathbf{X}_i[t+1] = \mathbf{X}_i[t] + \mathbf{F}_i[t]$.

To measure convergence, the following definition of the *stress* in the mesh is used. For a give set of coordinates for a given iteration of mesh relaxation, stress is computed as,

$$stress = \sum_{i \neq j} abs(||\mathbf{X}_i[t] - \mathbf{X}_j[t]||_2 - d_{ij})$$

As the estimated distance between nodes i and j approach d_{ij}, the stress approaches

zero.

It is worth noting that the preceding formulation is a *first order* spring model, where the node's *velocity* is proportional to the force it observes, as opposed to the node's acceleration. This model was chosen because it is easier to implement than a second order model (a single state variable – the node's position – must be stored between relaxation steps rather than both its position and velocity), and its convergence characteristics were adequate for our needs. A second order model may be worth considering if faster convergence is desired.

Chapter 4

The Localization System

The Pushpin localization system has been formulated twice during the course of this thesis research.

The first incarnation of the system was implemented exclusively on the Pushpin hardware testbed. It relied on a single assumption about the location of global events: that the Pinger must be triggered directly above some node (any node) in the network[16]. The justification for this assumption and its consequences are described in more detail in the following section, but suffice it to say for now that with it, it is possible to pick a set of anchor nodes, determine their coordinates, and determine the range from each node to the anchor nodes. At that point, the lateration algorithm can be directly applied to localize the non-anchor nodes. No additional algorithms besides lateration are required in this localization scheme. For this reason, we will refer to this as the *Pushpin Lateration System.* Using the lateration system, we have achieved localization error of 5-cm and an error standard deviation of 3-cm on the Pushpin hardware platform.

Soon after we completed and characterized the Pusphin lateration system, we attempted to build a second localization system; this time without the assumption that the Pinger be held directly above some node in the network. Without this constraint, we know of no linear technique for finding the coordinates of the sensor nodes and global events. Instead, we resorted to non-linear optimization techniques.

At first, we attempted to use a technique called metric multi-dimensional scaling

(MDS). MDS originated as a psychometric technique for visualizing data points that are related by a measurement of "similarity" [15]. For example, experimental subjects were asked to rate the similarities of various pairs of songs. The results could be fed to MDS to create a low-dimensional map of musical tastes and similarity. Such maps would often expose the underlying structure behind people's preferences such as musical genre, or the affect of the song. When "similarities" correspond to physical metrics such as distance, MDS can be used to determine the structure of a graph given only the distance relationships between the graph's vertices. This ability has piqued the interest of sensor network researchers such as Shang et. al., who have recently used MDS to localize sensor nodes based on inter-node ranging with received radio signal strength[61].

We adapted Shang's MDS-MAP algorithm to our use of global events rather than node-to-node ranges. The resulting algorithm consistently determined the positions of the nodes and global events with a high degree of accuracy, however it had its drawbacks. First, no distributed implementation of MDS currently exists. Distance measurements must be collected from every node in the network and processed in a central location. Furthermore, the MDS computation itself is complex and memory intensive. It is not well suited to run on the limited resources of the average sensor node.

Deterred by these disadvantages, we turned our attention instead to mesh relaxation [34]; an easily distributed non-linear optimization algorithm based on a physical ball/spring relaxation system. Rather than attempt to develop the mesh relaxation algorithm on the Pushpin hardware, which makes debugging and parameter tuning difficult, we elected to develop the new localization system on a virtual Pushpin network in the Pushpin Simulator (Section 2.4). Mesh relaxation worked as expected on the simulator except in one respect: the algorithm would often settle in local minima in which certain nodes were given coordinates that were reflected across some line or plane from their correct coordinates. This phenomenon is referred to as coordinate system folding by Priyantha et. al. [54]. It is a known shortfall of localization using mesh relaxation (and most likely in other non-linear optimization techniques as well,

though we have not observed this directly). As a remedy for this behavior, Priyantha proposes an algorithm for creating a "fold-free" initial guess that starts mesh relaxation nearby the correct solution. Ultimately, we adopted a different algorithm for finding a fold-free initial condition: the spectral graph drawing described in Section 3.3.1.

The second Pushpin localization system consists of two phases. In the *primary phase*, the coordinates of 10 anchor nodes and 5 Pinger events are determined using spectral graph drawing (to find a fold-free layout to be used as an initial guess) and mesh relaxation (to find the correct solution to the localization problem). Once the coordinates of the anchors and global Pinger events are found, lateration is employed to localize the remaining nodes. The lateration step is referred to as the *secondary phase* of localization. This new Pushpin localization system is distinguished by its use of spectral graph drawing and mesh relaxation, hence we refer to it as the *Pushpin SGD/MR System.* In contrast to the Pushpin Lateration System, the SGD/MR system has been implemented entirely in the Pushpin Simulator. It has not yet been rewritten or tested directly on the Pushpin hardware testbed, although implications for this are discussed in Chapter 6.

The remainder of this chapter is dedicated to detailed descriptions of both the Pushpin Lateration System and the Pushpin SGD/MR System.

4.1 The Pushpin Lateration System

This section gives an overview of the localization system developed and implemented on the Pushpins. As previously mentioned, it combines aspects of ultrasound time-of-flight and lateration into an approach that does not require any prior knowledge of the location of any of the sensing nodes or the pinger emitting the pulse of ultrasound and flash of light.

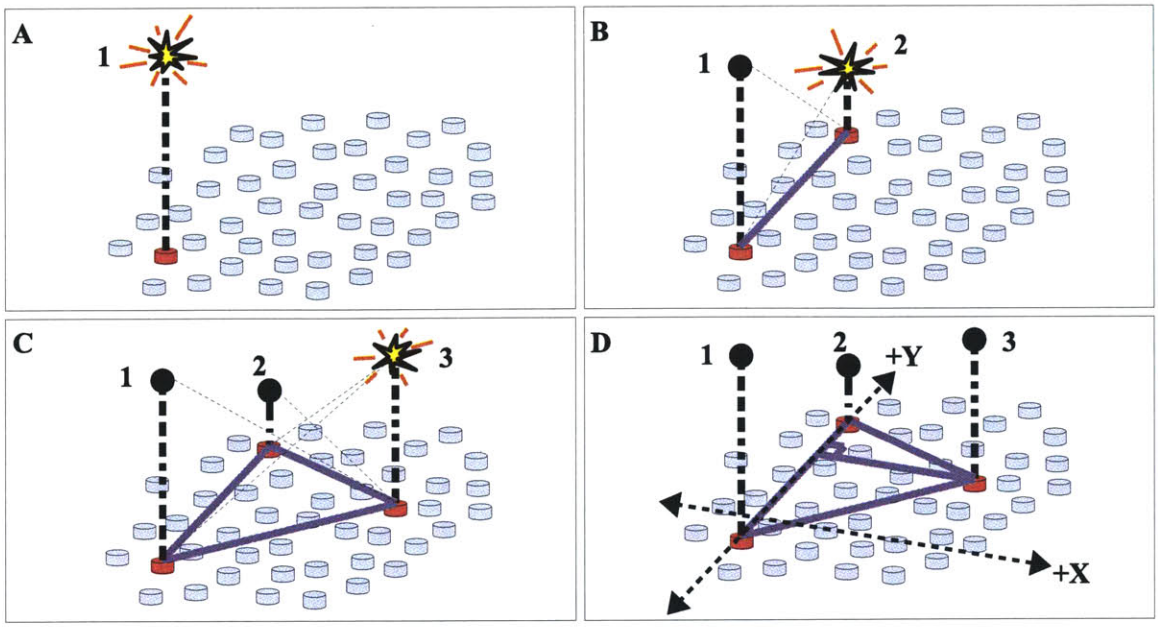


Figure 4-1: A) Each firing of the ultrasound/light pinger allows the network to select a single anchor point, namely the node directly beneath the location of the pinger. B) Once two anchor points are selected, they can determine the distance separating them by simple geometry. C) Three non-colinear anchor points guarantee that a planar coordinate system can be built up. D) The orientation and handedness of the coordinate system is arbitrarily, but consistently, chosen.

4.1.1 Establishing Anchor Nodes

The first flash of light signals the beginning of the localization process. The flash of light is detected roughly simultaneously by all nodes[1], which each start a hardware timer that runs either until they detect an ultrasound pulse or the timer overflows. If the timer overflows, the node assumes it has missed the ultrasound pulse and it stops participating in the localization algorithm altogether. If the sonar pulse is received, the node computes the distance from itself to the pinger, which is simply proportional to the time of flight of the ultrasound pulse and is easily calculated given the speed of sound in air (343.6 m/s at 20°C).

After a short delay ensuring every node has had a chance to hear the sonar ping, a leader election algorithm begins to determine which node was closest to the pinger.

[1]The detection speed of the flash is dictated by the size of the array and the speed of light – nanoseconds in our case.

The leader election algorithm consists primarily of pairwise comparisons of each node's measured time-of-flight for the ultrasound pulse in question. These pairwise comparisons continue until the network agrees which node is closest to the pinger event. This node is elected as an anchor point. A new anchor point is elected in this manner each time the pinger is triggered. In parallel with this process, newly elected anchor points compute their position. Figure 4-1 illustrates the process of computing anchor node coordinates.

The anchor point elected first is arbitrarily chosen to be the origin of the new coordinate system, and is given the coordinates $(0, 0)$ (frame **A**). The second anchor point is assigned coordinates on the new Y-axis. To determine this anchor's y-coordinate, anchor points 1 and 2 share information over the network about their respective distances to the first and second pinger location. Using these distances, which show up as the boldface and hairline dotted lines in frame **B**, the baseline distance - which is the y-coordinate of the second anchor - can be computed using basic trigonometry. The coordinates of the final anchor point are placed in the $+X$-half plane. These coordinates are computed using the triangle relations illustrated in frame **C** of the figure.

An important assumption is made about the geometry of this trigonometric calculation: the pinger is assumed to always be triggered directly above some node in the network. The node immediately below the pinger is nearest to the pinger, thus it will become an anchor node. With this assumption, any node can compute its baseline distance to an anchor point by solving a simple right triangle relation. Without this assumption, additional constraints would be necessary to determine the baseline distance to an anchor. The right angle constraint was chosen because it is easily justified for a dense sensor network: if the pinger is held at a random location above the Pushpin network, the inter-node spacing is small enough that the pinger is not likely to be more than a few centimeters away from being directly above some node. Also, considering that the width of the ultrasound transmitter array on the pinger is roughly as wide as the average distance between nodes, this is not a very severe constraint.

4.1.2 Lateration

Once three anchor points have been established, there is sufficient information to find a complete localization solution by solving a set of linear equations. This technique, called lateration, is described in [40], among others, and was summarized in Section 3.2. In the Pushpin lateration system, we rely on a 2-D version of lateration, which requires minimum of 3 anchors. This is a simplification of the 3-D formulation we provided in Section 3.2.

The lateration approach outlined above has a simple implementation, low communication overhead, and as Section 5.3 will show, it results in a relatively low localization error of approximately 5-cm. For many applications, it may be selected based on these merits alone. However, the restriction that the Pinger must be held over some node in the network is not ideal in all situations. The right-angle approximation implicit in the constraint on Pinger positioning given above leads to a localization error that is of the same order of magnitude as the inter-node spacing of the sensor nodes. The right angle approximation would not suffice in a network in which localization error is desired to be much lower than the average spacing in between the nodes. In cases where assumptions about the positioning of the Pinger cannot be made or when they lead to excessive inaccuracy, another localization approach must be used. The next section describes such an approach.

4.2 The Pushpin Spectral Graph Drawing / Mesh Relaxation System

In this section, we outline a more general localization system based on a combination of spectral graph drawing, mesh relaxation, and lateration. This approach does not require that the global stimulus occur above some node in the network. However, removing this restriction leads to a system that is a great deal more complicated than the simple lateration-only based system described. In addition to the use of non-linear optimization techniques, which are inherently more complicated than lat-

eration alone, this localization system also includes pre- and post-processing steps for rejecting outliers. The overall process of localizing using SGD/MR can be summarized as follows:

1. **Pre-Processing** – Compare time-of-flight measurement with neighboring nodes and reject your measurement if it differs substantially from the median of the neighbors' measurements.
2. **Establish Anchor Nodes** – Elect one node as the origin of the coordinate system and ten nodes as anchor nodes. These nodes must have a full complement of good (not outlying) time-of-flight measurements.
3. **Primary Localization** – Anchor nodes and the origin node collaborate to localize using spectral graph drawing followed by mesh relaxation. This results in coordinates for anchor nodes and the Pinger events.
4. **Secondary Localization** – Remaining nodes, termed *passive* nodes use the global event coordinates that are produced during primary localization to localize themselves using lateration.
5. **Post-Processing** – Any node with localized coordinates that grossly disagree with the coordinates of their neighbors' are rejected as outliers.

To clarify the detailed explanation of each of these steps, the following names are given to describe the various roles played by nodes during the localization process:

- **Root Node** – The node charged with selecting one origin node and ten anchor nodes, all of which must have a full compliment of good (non-outlying) time-of-flight measurements.
- **Origin Node** – The origin of the coordinate system. This is also the node where the SGD/MR computations are carried out.
- **Anchor Node** – A participant in SGD/MR during primary localization. These nodes share their measurements with the origin node, where the SGD/MR algorithms are actually executed.
- **Passive Node** – The majority of the nodes in the network are passive nodes. They must wait until the origin and anchor nodes have localized SGD/MR before they can be localized themselves using lateration.

4.2.1 Pre-Processing: Reject Outliers in Sensor Data

Simulations and hand calculations do not always take into account the characteristics of the real world sensor stimuli. While most localization algorithms for sensor

networks have been subjected to thorough simulation or a manual error analysis such as computation of the Cramer-Rao Bound [49, 57], these assessments are only as accurate as the system and noise models on which they are based. Such assessments about the accuracy of a localization algorithm are not always born out when real world sensory stimulus is used for ranging. Spurious outliers are common in real sensor measurements, and they must be detected and rejected. In a sensor network, this must occur in a distributed manner.

We have observed some of this spurious behavior in the ultrasound time-of-flight measurements made between the Pushpin TOF module and the Pinger device. In early tests, roughly 5 out of 50 nodes would not detect the sonar wavefront generated by the Pinger[2]. We determined that this was due to destructive interference between the array of nine narrow-band sonar heads on the Pinger device (see Figure 2-7). We also noticed that nodes sometimes missed the direct line-of-sight ping (due to destructive interference), and instead detected a delayed version of the Ping that has reflected off of a nearby object. This introduced outlying time-of-flight measurements that fouled the localization results. Mesh relaxation proved to be particularly sensitive to outlying measurements. A single outlier was enough to affect the coordinate estimates of every node as the mesh distended to accommodate the outlier. A preprocessing step for detecting outliers was clearly needed to make our system robust to these errors. To this end, we have formulated the following distributed method for eliminating spurious outlying measurements.

1. After a Ping is detected, each node requests that its neighbors send their time-of-flight measurement. The responses are collected and stored in memory.

2. After a short time sufficient for all neighbors to reply, each node finds the median and standard deviation of the stored values.

3. If a node's own time-of-flight measurement is more than 1.5 standard deviations from the median of its neighbors' pings (this parameter is adjustable in the Pushpin Simulator), the measurement is labeled as an outlier. This is assuming the node has more than one neighbor (a necessary provision which helps to avoid the degenerate case where both nodes may reject their measurements).

[2]A later revision of the Pinger reduced the destructive interference considerably, however we still do observe occasional missed pings and outliers.

Ultimately, a node can only participate as an anchor node or origin node in SGD/MR if it has no outlying measurements. We have found that roughly ten nodes are typically discounted from participating in SGD/MR when this technique is used. In our network of 55 nodes, this leaves plenty of candidates for the important jobs of origin and anchor nodes. It is important to note, however, that nodes with rejected measurements are still able to localize as passive nodes using lateration as long as they have at least 4 good measurements (the minimum required for lateration). We observed that an average of 2 nodes could not participate in localization because they do not have enough good time-of-flight measurements.

4.2.2 Establishing Anchor Nodes

In many sensor networks, anchors (sometimes called beacons) play an import role in localization. Typically, anchor nodes come pre-programmed with their coordinates. Unlocalized nodes use anchor coordinates and range measurements to multiple anchors to triangulate their position.

In the SGD/MR localization system proposed here, the purpose of the anchor nodes is slightly different. Since range measurements in the Pushpin system are between sensor nodes and global Pinger events rather than sensor nodes and anchor nodes, the critical pieces of information that allows passive nodes in the Pushpin system to determine their coordinates via lateration are the coordinates of the Pinger events. The coordinates of the anchor nodes themselves are incidental to the localization process. In SGD/MR, the primary role of the anchors is to provide a full complement of good time-of-flight measurements as constraints so that the positions of the pinger events can be accurately determined using SGD/MR. However, since they do contribute to "anchoring" the locations of the global events, we still refer to these nodes as anchor nodes.

A single node called the *origin* node also plays a unique and important role in primary localization. It gathers time-of-flight measurements from the anchor nodes and runs the SGD/MR computation. The origin node also controls when the localization process starts (after how many pings), and it signals the beginning of the secondary

phase of localization.

No node in the Pushpin network is different from any other except for their network ID, so the origin and anchor nodes can be selected at random. However the selected nodes must have a full complement of good time-of-flight measurements (no outliers and no missed pings), otherwise the there will not be enough constraints to perform non-linear optimization via SGD/MR. This selection criterion can not be applied until after several pings have been received, so there must be some node in the network that waits until several pings have been received, then elects several nodes with good Pings as anchors and the origin. This node is referred to as the *root* node. The root node is elected when the sensor network is powered up using the distributed leader election algorithm described in Section 4.1.1. Once a sufficient number of pings have been received to run SGD/MR, it queries the network for nodes that have no outlying measurements. It chooses 1 origin and 10 anchor nodes at random from the respondents. Once these nodes have been elected, the origin node takes over as the arbiter of the localization process, and the root node enters the pool of passive nodes.

4.2.3 Primary Localization: Anchor Nodes

The newly elected origin node plays the most important role in the localization process. It broadcasts a request over the network, instructing all anchor nodes to send their time-of-flight measurements to the origin. These are collected and stored in memory in an $n + k \times n + k$ adjacency matrix, where n is the number of anchor nodes and k is the number of Pinger events. The SGD algorithm expects that nodes that are closer together (more adjacent) have a larger value in the adjacency matrix. Therefore the values in the adjacency matrix must be converted from distances d_ij to weights w_{ij} via $w_{ij} = -exp(d_{ij})$ (see Section 3.3.1).

Alternatively, the system can be configured to run spectral graph drawing using node-to-node constraints generated by network distance (hop count) estimates, rather than the distance constraints imposed by global Pinger event. This is simply an alternative method of generating adjacencies for spectral graph drawing that may be

useful when sensor-based range measurements are not available. We have included this capability to demonstrate that spectral graph drawing is still effective when network-based distance metrics are employed.

Either way, the new adjacency matrix is fed directly to the Spectral Graph Drawing algorithm which runs on the origin node. It produces an approximate, fold-free, 3-D set of coordinates for the anchor nodes and pinger locations.

Next, the adjacency matrix is converted back from weights to distance (using $d_{ij} = -\ln(w_{ij})$), and the origin node runs mesh relaxation, using the coordinates provided by SGD as an initial guess. The resulting coordinates are the final estimates of the location of the anchor nodes and pinger events. The origin disseminates this information by broadcasting it over the network, thereby updating the locations of the anchor nodes and providing the passive nodes with the locations of the Pinger events, enabling them to begin the secondary phase of localization.

Nearly all of the processing in the primary phase of localization occurs on the origin node. Fortunately, these algorithms are fairly light-weight and fast. An 8-bit microcontroller with 128-kB of memory (for the adjacency matrix and temporary variables) and a 22-MIPS processor would be capable of running both algorithms in under a minute. These requirements do not require that the origin node be different from the other nodes in the network. In particular, it does not need to be a "heavy-weight" node with a more powerful processor or additional sensors.

4.2.4 Secondary Localization: Non-Anchor Nodes

During primary localization, most of the nodes are passive; they are non-participants in the localization process. However, these nodes do collect time-of-flight measurements whenever they detect a Pinger event, and they confer with their neighbors and reject outliers as described in Section 4.2.1. When a passive node hears a broadcast from the origin, the second phase of the localization process begins.

Passive nodes localize using lateration. The broadcast from the origin node contains the positions of the Pinger events, which, when combined with the distances a passive node has measured between itself and the Pinger events, is the only in-

formation necessary for passive nodes to compute their position. There is no need for them to collaborate further with other passive nodes or with the anchor nodes. However, each passive node must have at least four good time-of-flight measurements for lateration to yield coordinates in three dimensions. If a passive node has fewer than four good measurements, it withdraws from the localization process.

4.2.5 Post-Processing: Rejecting Outlying Coordinates

After both primary and secondary localization have been completed, the network performs one final check to ensure that no node has been assigned an unreasonable set of coordinates. To do this, the physical proximity of neighbors on the network is used as a sanity check on the estimated position of the nodes. Post-processing occurs in two phases.

1. Each node shares its estimated coordinates with its immediate network neighbors. As the node receives its neighbors' coordinates, it calculates the distance between the estimated coordinates of itself and the neighbor. The median of these distances is stored in a variable called `medianDistanceToNeighbors`.

2. Each node then shares `medianDistanceToNeighbors` with its neighbors. As the node receives its neighbors' median distances, it computes the *median of its neighbors'* `medianDistanceToNeighbors`. If this value is more than 1.7 time the node's own `medianDistanceToNeighbors`, it is an outlier, and it withdraws from the localization process.

To summarize, if the `medianDistanceToNeighbors` of a node is more than 1.7 times the median of its neighbors' `medianDistanceToNeighbors`, it stands down. This technique for distributed outlier rejection requires only two rounds of neighbor-to-neighbor communication transactions.

Chapter 5

Experimental Results

This chapter presents a characterization of the Pushpin Localization Systems described in Chapter 4. All tests were carried out using either the Pushpin hardware testbed, the Pushpin Simulator, or both. Global range constraints for localization were generated by the Pinger device described in Section 2.3.2.

The localization systems we have developed generates coordinates that are in the reference frame of the Pushpins. In order to assess localization accuracy, this coordinate system must somehow be aligned with a set of "ground truth" coordinates that have been measured in a reference frame external to the Pushpin network. We begin this chapter by developing two such transformations. We then turn our attention to a characterization of the distance constraints used for localization; namely, the ultrasound time-of-flight measurements made using the Pushpin TOF expansion module. With these two preliminary sections out of the way, we then proceed with a characterization of the Pushpin Lateration System and the Pushpin SGD/MR System.

5.1 Aligning Coordinates & Measuring Error

There is no global coordinate system. When sensor nodes work together to create a coordinate system, the result is a relative coordinate system; i.e. a local coordinate system in the frame of reference of the sensor nodes. The orientation of this coordinate system is usually incidental to the localization algorithm that is employed. It

rarely lines up with common external coordinate systems such as earth latitude and longitude or the x-y position in meters on the floor of a room. For example, the Pushpin lateration system presented in Section 4.1 produces a set of coordinates in an arbitrary reference frame that depends on the position of the three anchor points, which in turn are determined by the location of the Pinger at each light/ultrasound event.

Most visions of practical sensor networks include passing location-specific information to agents outside the network for processing. In order for these data to be useful to an external recipient, a transformation between the internal and external coordinate systems must be found. This transformation also comes in handy when assessing the accuracy of a localization system. In the Pushpin system, we use a transformation that aligns the coordinates produced by the localization algorithm with a set of "ground truth" coordinates measured by hand. Ground truth coordinates are obtained by photographing the array of Pushpins using a digital camera with a telephoto lens (to reduce the effects of image warping). A 0.1-m by 0.1-m square is included in the image for scale and to verify that no image warping occurs due to the camera lens. This image is imported into image manipulation software where the coordinates are manually extracted.

We refer to the coordinates generated by the localization algorithm as *estimated* coordinates while the "ground truth" coordinates are called *measured* coordinates. We consider two techniques for aligning measured and estimated coordinates: (1) A transform comprised of only a translation, rotation, and reflection (RTR); and (2) a more general homogeneous affine transform that can also include scaling and shearing. When the fit is computed as in (1), we refer to it as an *RTR fit*. The fit computed as in (2) is referred to as an *LLSE fit* because the affine transformation can be found most easily by computing a linear least squares estimate.

5.1.1 RTR Fit

A RTR fit aims to minimize the mean square error between the estimated and measured coordinates by applying some translation, rotation, and possible reflection. To

simplify the calculation, the centroid of the estimated coordinates is translated to the origin. This centers the coordinates at a point around which they can be easily rotated. The measured coordinate systems is translated in a similar manner, centering its centroid at the origin as well.

The 3D rotation that aligns the estimated and actual coordinate systems is called a Euler rotation sequence. It is named after Leonhard Euler who, among his many other significant contributions to Math and Science, proved that any two arbitrary 3D reference frames can be related with no more than three rotations [38]. The rotation sequence is composed of the rotations $R_z(\psi)$, $R_x(\phi)$, and $R_y(\theta)$, where the subscript denotes the axis of rotation and the function argument specifies the angle of rotation. These rotations can be applied in any order, however the angles which yield an equivalent rotation change depending on the order in which they are applied, so the order must be consistently chosen. To find the angles that minimize mean square error between estimated and actual coordinates, we have used MATLAB's `fminsearch` optimization function. This optimization is run several times for several different reflections of the original coordinates. The trial that achieves the lowest error is taken as the correct reflection and rotation.

5.1.2 LLSE Fit

Finding the LLSE fit is a straightforward matter of finding the linear least squares approximation. More explicitly, an affine transformation A must be found that, given homogeneous estimated coordinates $(\hat{x}, \hat{y}, \hat{z}, 1)$ in the estimated coordinate system, yields a pair of transformed coordinates $(\hat{x}', \hat{y}', \hat{z}', 1)$ in the measured coordinate system. The goal is to minimize the average error between the transformed, estimated coordinates $(\hat{x}', \hat{y}', \hat{z}'1)$ and the measured coordinates $(x, y, z, 1)$ over every node in

the network. This can be accomplished by solving the following linear system:

$$A \begin{pmatrix} \hat{x}_1 & \hat{x}_2 & & \hat{x}_N \\ \hat{y}_1 & \hat{y}_2 & \cdots & \hat{y}_N \\ \hat{z}_1 & \hat{z}_2 & & \hat{z}_N \\ 1 & 1 & & 1 \end{pmatrix} = \begin{pmatrix} x_1 & x_2 & & x_N \\ y_1 & y_2 & \cdots & y_N \\ z_1 & z_2 & & z_N \\ 1 & 1 & & 1 \end{pmatrix}$$

$$AX = B$$

where X is an array of untransformed, estimated coordinates and B is an array of measured coordinates. This over-determined linear system has a linear least squares solution of the form:

$$A = BX^T(XX^T)^{-1} \tag{5.1}$$

5.1.3 Measuring Localization Error

After the coordinates have been fit using one of the above methods, the localization error can be assessed. The objective is to determine the average error between the estimated, transformed coordinates $(\hat{x}', \hat{y}', \hat{z}')$ and a set of measured coordinates (x, y, z) obtained manually by photographing the array as described above. We start by defining the error for a single node i:

$$e_i = \sqrt{(x_i - \hat{x}_i')^2 + (y_i - \hat{y}_i')^2 + (z_i - \hat{z}_i')^2}$$

The following *mean absolute error* metric can then be used to compute localization error for n sensor nodes:

$$\bar{e}_{mae} = \frac{\sum_{i=1}^{n} e_i}{n} \tag{5.2}$$

The variance σ^2 (the square of standard deviation) of localization error is computed using the following unbiased estimator:

$$\sigma_{mae}^2 = \frac{1}{n-1} \sum_{i=1}^{n} (e_i - \bar{e}_{mae})^2 \tag{5.3}$$

Other statistics including the median, minimum, and maximum values are computed in a similar manner using other standard statistical formulae.

5.2 Pinger Characterization

Pushpin localization relies on ultrasound time-of-flight (TOF) measurements to generate constraints that aid in ad-hoc localization. It is reasonable to assume, therefore, that the accuracy of the localization system is highly correlated with the accuracy of the TOF measurements. The fundamental limit on the precision of detecting a sonar signal using the unipolar threshold detector in the Pushpin TOF expansion module is roughly one cycle of the sonar wave, or approximately 1-cm. However, many other effects come into to play that further degrade the accuracy of these measurements. This section will present an error analysis of the TOF measurement hardware and present several improvements that were made to enhance overall localization accuracy. These include a hardware modification to the Pinger device that eliminated destructive sonar interference and a simple linear calibration scheme that eliminated the systematic error in the measurements.

5.2.1 Eliminating Destructive Interference

The TOF measurement system is comprised of a single transmitting device called the "Pinger" (Section 2.3.2) and several identical receivers; one sensor suite on each Pushpin's Time of Flight Expansion Module (Section 2.3.1). Neither of these elements are ideal. For example, the ultrasound transducers on the Pinger are arranged in a roughly triangular array. Each side of the triangle has a length of approximately 5-cm, which is about half the inter-node distance between Pushpins. Therefore the ultrasound source from the Pinger is far from being an ideal point source. We have also noted that the close proximity and high frequency of these transducers results in destructive interference of the sonar signal at certain points in the space in front of the Pinger. Pushpins that happen to be in an area of destructive interference cannot detect the direct sonar signal and will instead trigger off of a delayed, reflected signal

or not at all.

These "missed pings" contribute substantially to localization error, and must be carefully detected and rejected as a pre-processing step before localization. However, an overabundance of outliers can confuse the outlier detection algorithm and greatly degrade overall system performance. Ultimately, we found that a hardware modification to the Pinger was necessary to achieve acceptable levels of error. Hence, we developed two versions of the Pinger hardware. The *Pinger v.1* has a full complement of 9 sonar transmitters. Its successor, the *Pinger v.2* has only one sonar transmitter. The Pinger v.2 eliminates nearly all of the destructive interference and greatly increases localization accuracy. In each of the test results below, we will be careful to specify whether the Pinger v.1 or Pinger v.2 was used.

5.2.2 Characterizing Time-of-Flight Measurement Error

The sonar detector on the TOF module also has flaws that degrade detection accuracy. The TOF module includes circuitry for a fixed-threshold, uni-polar rising-edge discriminator. Ideally this circuit should detect the sonar signal at some point within the first full cyle of when it arrives at the Pushpin. In practice, we have found that the receiving transducer takes several cycles to "ring up" to the discriminator threshold. This induces a delay that is proportional to the distance between the Pinger and the TOF module. This phenomenon is frequently referred to as "time walk" for a decreasing signal with a constant threshold.

In order to characterize the impact of time walk as well as any other source of systematic error in our system, a test was set up in which measurements were taken between the Pinger v.1 and a single node. The pinger was held opposite the stationary node at distances between 0-m and 3.5-m at 0.1-m intervals. At each measured distance, four TOF measurements were recorded. Figure 5-1 shows the error between the measured (ground-truth) distances and the estimated distances measured by the TOF hardware.

The plot shows a baseline error of 0.13-m and an additional linear error that is proportional to distance with a slope of 0.022. We believe the baseline error to be

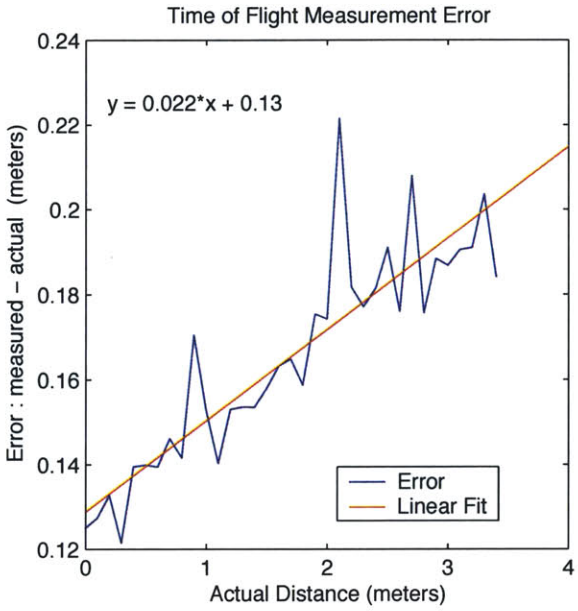


Figure 5-1: Error in ultrasound time-of-flight measurements. The plot shows a clear linear trend between increasing distance and measurement error.

the result of the delay introduced by processing the sonar signal in the sonar detector circuit, while the distance related error is primarily the result of time walk as discussed above. Overall, this combined baseline plus distance error is referred to as *systematic error*. The systematic error is easily removed from the measured TOF distance d_{tof} using a linear calibrator:

$$d_{corrected} = d_{tof} - (0.022 * d_{tof} + 0.13)$$

5.2.3 Modeling the Error

After applying this calibration, what remains can be termed *random error*. We believe this error to be the result of several of the properties of propagating sound waves including dispersion, speckle, changing air currents, interference among ultrasound transmitters, and part variability. Figure 5-2 shows a histogram of the random error. Two interesting things can be noted from this plot. First, the center peak appears to be a Gaussian with a standard deviation of 0.57-cm. Second, the random

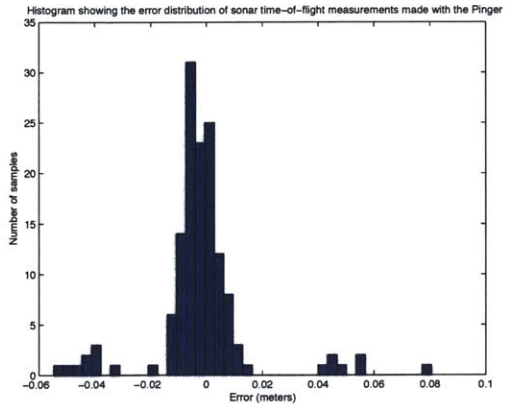


Figure 5-2: Random error in ultrasound time-of-flight measurements after the linear calibrator was applied. The plot shows that the rising edge discriminator detector will sometimes be off by one or two full cycles of the sonar wave.

error is multimodal. Peaks appear at regular intervals, though with the limited number of samples in this dataset, it is difficult to guess exact spacing of the intervals. Nonetheless, we believe that peaks represent full cycles of the sonar signal between crossings of the detection threshold. This behavior is expected of a signal that rings up in a unipolar discriminator. Such a detector will occasionally be off by one or more cycles depending on how quickly the receiving signal rings up for any given ping.

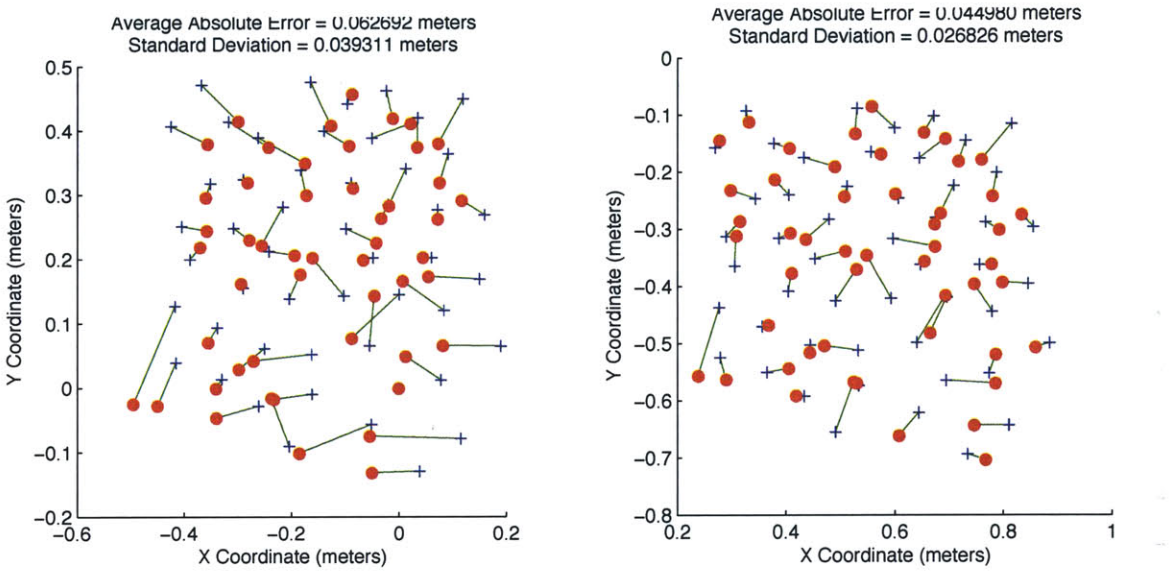
We have found it instructive to test the validity of the preceding analysis by comparing the SGD/MR localization results produced using real time-of-flight data to simulated pings that are generated at random locations by the Pushpin Simulator. Simulated pings are generated using one of the following statistical models:

- **No error** - Measurements between the Pushpin and Pinger are exact.
- **Gaussian error** - Measurement error is modeled as a single gaussian with a standard deviation of 0.57-cm.
- **Gaussian mixture (GM) error** - Measurement error is modeled as a multimodal gaussian with a distribution that closely resembles that in Figure 5-2.

Localization results using both real and simulated pings are presented along with other SGD/MR results in Section 5.4.

5.3 Pushpin Lateration System

Unless otherwise noted, the results presented in this section were derived on the Pushpin hardware testbed when it was configured with 55 nodes. Five localization trials were carried out, each consisting of three ultrasound/light pings from the Pinger v.1 that were randomly located (but non-colinear) within the 1 cubic meter space above the Pushpin network.



(a) RTR Fit

(b) LLSE Fit

Figure 5-3: Estimated Pushpin coordinates after being fit using a rotation, translation, and possible reflection (RTR fit) or a more general Affine transform that includes scaling and shearing (LLSE fit). The crosses denote measured coordinates and the circles denote coordinates estimated using lateration. The line connecting each cross to a circle indicates which estimated coordinate corresponds to which measured coordinate.

Estimated coordinates were generated on the actual Pushpins in the testbed via code that implements the Pushpin Lateration System described in Section 4.1. The resulting coordinate estimates were collected over the network by the Gateway node, which was wired to the serial port of a PC. The entire data collection process took less than 30 seconds for 55 Pushpins. Once the data was collected, outliers were rejected offline using the same method used to reject outlying coordinates in the SGD/MR

Trial	RTR Fit		LLSE Fit	
	Mean	Std Dev	Mean	Std Dev
1	6.09	2.77	4.51	2.91
2	5.58	3.87	5.83	4.02
3	6.61	4.93	4.78	3.03
4	6.27	3.93	4.50	2.68
5	5.93	3.99	5.01	2.60

Table 5.1: Mean absolute error statistics for the Pushpin Lateration System over five Pinger configurations . All distances are in centimeters.

system (Section 4.2.5). We found that there were typically 3-4 outliers per localization trial.

As given, the algorithm produces a set of coordinates in an arbitrary reference frame that depends on the position of the three anchor points which in turn, are determined by the location of the pinger at each light/ultrasound event. To account for this, the measured and estimated coordinate systems must be aligned using either the RTR or LLSE fit. When equation 5.2 is applied to estimated coordinates transformed using the RTR fit, we refer to this as *unscaled error*, since the distance metric in these coordinates has not been scaled by the transformation, and still reflects absolute distances in the physical world. The error in coordinates transformed using an LLSE fit is refered to as *scaled error*. Figure 5-3 shows the actual and estimated coordinates after being fit using the RTR and LLSE techniques.

Homogeneous affine transformations are a superset of translations and rotations, so we expect the scaled error to be smaller than the unscaled error since the scaled error includes extra degrees of freedom that allow for a better fit to the measured coordinates. The degree to which these two approaches produce similar results tells us how immune the estimated coordinate system is to shearing and scaling. Immunity to scaling and shearing is desirable in many applications where the distances measured in the estimated coordinate system must correspond to real, absolute distance in the physical world. For applications where only relative distance measurements are necessary, scaling and shearing effects can be tolerated.

Table 5.1 shows that the error results using the unscaled method of calculating

error are comparable to the error results using the scaled method, indicating that the localized coordinate system is nearly free of distortion due to scaling and shearing. As expected, the error resulting from the scaled method is slightly lower than the error resulting from the unscaled method, with the exception of trial 2 which is close enough to suggest that the discrepancy may not be statistically significant (we are looking into this). Also encouraging is the consistency between trials of mean error and error standard deviation.

5.3.1 Diagonal Wipe Demonstration

As a first test of the lateration system, we devised a simple visual demonstration meant to show that each node knows its coordinates. Once all nodes localize themselves, by referencing their common time-base as synchronized by the flashes, they individually simulate a visual "wipe" effect by animating a line that starts at x=-1 meter and moves toward x=+1 meters parallel to the Y-axis. Once the line passes the position a node has localized to, the node changes color from green to red. The results of this demonstration visually verify that the nodes have been spatially localized. In essence, the localized sensor network is being used as a simple display. Figure 5-4 shows a time lapse of the wipe demonstration.

5.4 Pushpin Spectral Graph Drawing/Mesh Relaxation System

The SGD/MR localization system was tested on a 58 node network in the Pushpin Simulator. The actual (ground truth) positions of the simulated nodes were imported from a photograph of the real Pushpin nodes, hence the simulated Pushpins were also distributed over and area of 1-m^2. The communication range of each node was circular with a fixed radius of 0.25-m. This gave each node an average of 12 neighbors. Unless otherwise specified, all tests were configured for 50,000 iterations

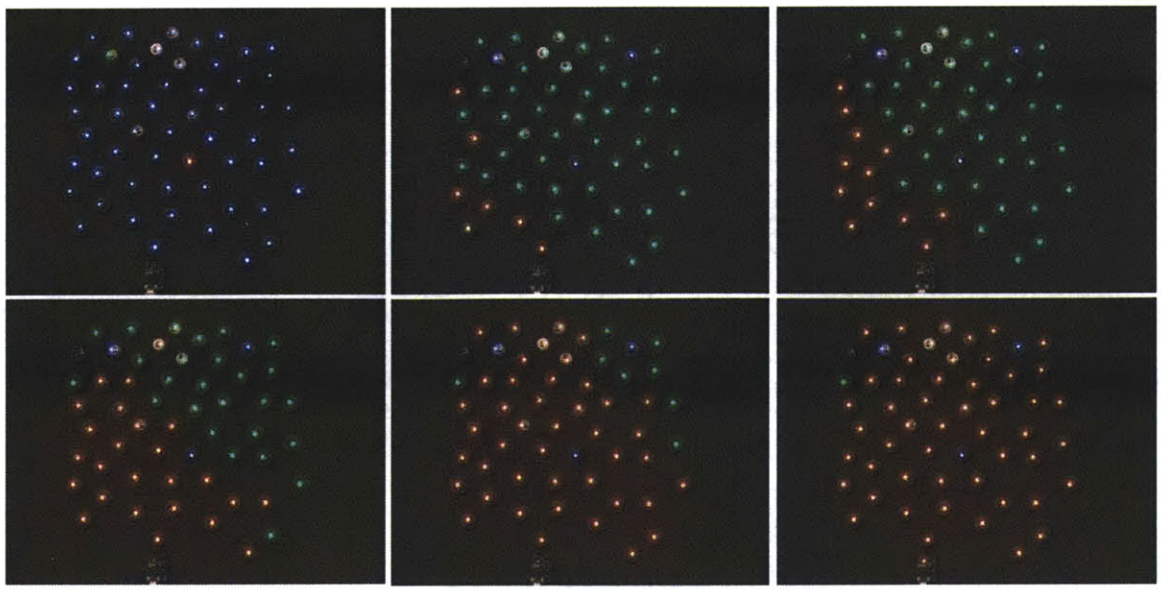


Figure 5-4: Starting in the upper left, going from left to right: 1) Two pings have been fired and their respective anchors elected. The first anchor (origin) is the red node in the middle, the second anchor is the green node in the upper left. The line connecting them defines the Y-axis in the localized coordinate system. 2) A third ping was fired and the third anchor elected in the upper right. The third anchor defines the handedness of the coordinate system. In this case, the +X direction is where it normally is for a right-handed coordinate system. All three anchors are now blue. 3) A simulated diagonal wipe can be seen entering from the -X direction propagating in the +X direction. The line that delineates the boundary of the wipe is parallel to the Y-axis. 4) The diagonal wipe effect is *not* a result of communication among nodes, but rather the result of all nodes sharing a global time base and coordinate system and using these to individually simulate the same moving wipe boundary. 5) The wipe maintains its structure as it passes over the network. A green outlier can be seen in the upper left. 6) The wipe has passed, but will wrap around toroidally.

of mesh relaxation[1], and outlier rejection thresholds for time-of-flight measurement outliers and final coordinate outliers of 1.5 and 1.7 standard deviations respectively.

Although the SGD/MR system was characterized entirely within the Pushpin Simulator, most tests were performed using real ultrasound time-of-flight measurements that were imported from the Pushpin hardware test bed. Ten trials of five Pings per trial were recorded on the Pushpin array and downloaded to a PC through the gateway node. In each trial, the Pinger was held at 5 random locations that were between 0.5-m and 3-m from the Pushpin network. It should be noted that the

[1]This was many more than was needed in most cases. Most trials converged in less than 5000 iterations.

Pinger was not always triggered directly above some node in the network as it was in the Pushpin Lateration System tests.

In addition to using time-of-flight measurements recorded on the real Pushpins, the simulator has the capability to generate simulated pings. The error model for these pings can be specified as one of (1) no error, (2) gaussian error or (3) gaussian mixture error (GM error), as described in Section 5.2.3. The degree to which localization results agree when subjected to real or simulated pings provides an idea of how well the system has been characterized. Any outstanding differences hint at un-modeled error characteristics in the time-of-flight measurements.

For most of the test variations described below, localization was run either 5 or 10 times on each of the 10 data sets collected from the Pushpin hardware testbed. During each of these trials, different anchor nodes were randomly chosen by the root node as described in Section 4.2.2. Different anchors produce slightly different localization results, so the accuracy for a given data set does vary over the 5 or 10 trials. When combined, the trials for all ten data sets produced 50 or 100 sets of estimated coordinates. Each set was individually aligned with the measured coordinates and the mean absolute error was assessed according to Equation 5.2. Other statistics including the median, standard deviation, minimum, and maximum error were also recorded for each trial. All statistics were themselves averaged over the 50 or 100 trials to produce the final results that appear in the tables and graphs below. Where it is appropriate, the number of unlocalized nodes (i.e. nodes that were rejected from localization by an outlier detection processes) have also been recorded. Unless otherwise noted, all distances in this section are in centimeters.

As a final note, all coordinate fits in the following results were performed using the LLSE method. Section 5.4.5 explains in detail why RTR results are omitted.

5.4.1 Anchor Accuracy

This test assesses the accuracy of anchor placement using the spectral graph drawing and mesh relaxation algorithms during the first phase of SGD/MR localization. The results in this section do not include the error due to passive nodes that are localized

	Network SGD	TOF SGD	Mesh Relaxation
Mean	5.53	3.07	2.12
Median	5.26	2.76	1.83
Std Dev	2.84	1.87	1.42
Min	1.73	0.75	0.49
Max	10.5	6.47	4.82

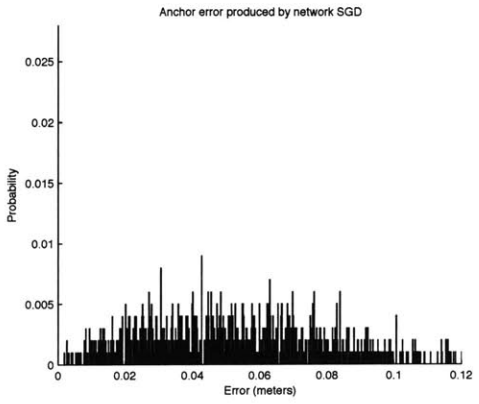
Table 5.2: Localization statistics for anchor nodes alone. These are averages over several trials and Pinger configurations. All data is fit using the LLSE. Measurements are in centimeters.

using lateration.

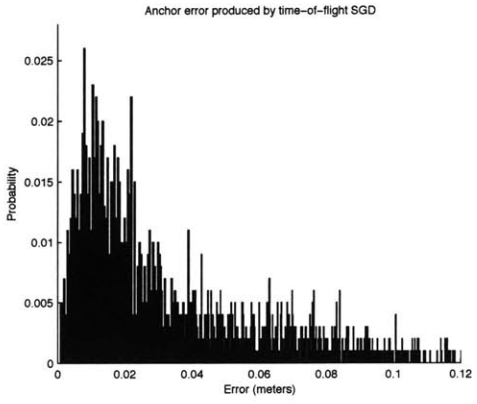
Two variants of spectral graph drawing have been considered. One uses the hop count between anchor nodes on the network as a distance metric (we refer to this as *network SGD*), while the second variant uses more precise time-of-flight distance measurements (this is *TOF SGD*). For each technique, we ran 50 Localization trials – 5 for each of the 10 data sets from the hardware testbed. The averaged statistics for these trials are summarized in Table 5.2. For comparison, the third column of the table also includes anchor accuracy after mesh relaxation is complete.

Table 5.2 shows that, as expected, network SGD is less accurate overall than TOF SGD. Mesh relaxation is more accurate than both SGD methods alone. However, TOF SGD is only one centimeter less accurate than mesh relaxation, hence SGD may be sufficient for localization on its own in a sensor network where the simplicity gained by omitting mesh relaxation is valued over accuracy.

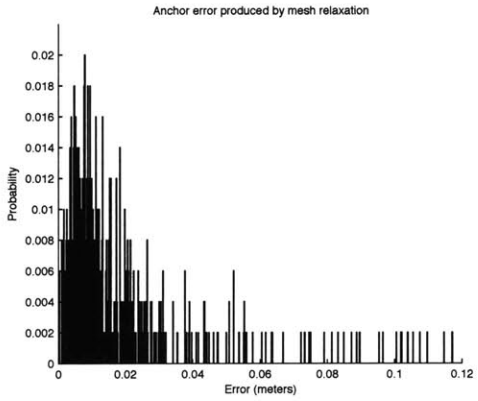
Figure 5-5 shows the error distribution for each of the three anchor localization techniques. Once again, it is apparent that network SGD is the least accurate technique. However, TOF SGD and mesh relaxation both achieve a very similar distribution. The peak of the mesh relaxation distribution occurs less than a centimeter to the left of the peak of the TOF SGD distribution. It is also worth noting that these distributions have "heavy tails" that extend off of the graphs to the right (the tails were truncated so that the majority of the distribution could be scaled to fit in the graph). This implies that a small but consistent percentage of nodes have high localization error.



(a) Network SGD



(b) Time-of-flight SGD



(c) Mesh Relaxation

Figure 5-5: Error distributions for anchor nodes localized using spectral graph drawing and/or mesh relaxation in the anchor accuracy tests. All three distributions have heavy tails that have been truncated on the right side of the graph.

5.4.2 Anchor Convergence

The primary reason that we employ spectral graph drawing in our localization implementation is to avoid false mesh relaxation solutions that do not correspond to the correct layout of the sensor nodes. Throughout our tests, we have found that spectral graph drawing is extremely effective in this regard. We did not observe a single instance where it was clear that mesh relaxation primed by spectral graph drawing converged to a folded or malformed layout.

However, we were quite surprised to learn that mesh relaxation actually performed very well on its own without spectral graph drawing. In trials where nodes were placed randomly in a 1-m^2 area, mesh relaxation converged to the correct solution for 48/50 trials, or 96% of the time.

This result was unexpected, since it ran counter to our intuition and the findings of other researchers[54]. Our early tests indicated that folding and false solutions would be a problem; in these tests, mesh relaxation converged less than half the time. However, these early tests constrained the nodes to move about in a plane, whereas our final implementation allows them to move freely in 3-space. We postulate that this extra degree of freedom considerably simplifies the optimization landscape for mesh relaxation. Therefore, while spectral graph drawing may not have been necessary to generate an initial guess in our final setup, it may still be useful in other more tightly constrained localization scenarios. Additionally, the anchor accuracy results in Table 5.2 suggest that, while not as accurate as mesh relaxation, spectral graph drawing can be used as a stand-alone localization technique if slightly lower accuracy is acceptable.

5.4.3 Overall Accuracy

The anchor accuracy tests assessed how well spectral graph drawing and mesh relaxation work to localize a small subset of nodes on the network. The overall accuracy test expands this error assessment to include passive nodes that have been localized during the second phase of the SGD/MR system using lateration.

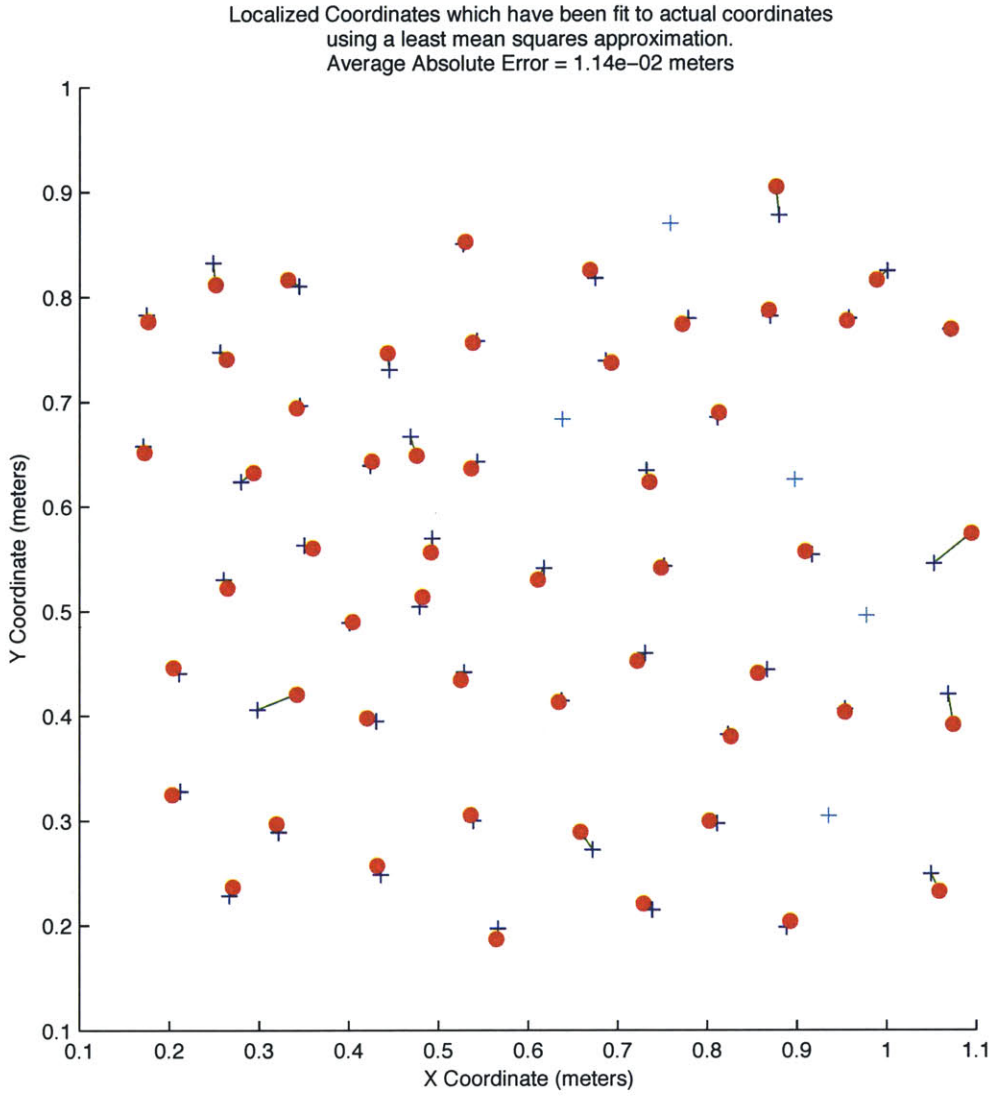


Figure 5-6: Estimated Pushpin coordinates generated using the SGD/MR system that have been aligned with measured coordinates via a LLSE fit. The crosses denote measured coordinates and the circles denote coordinates estimated using lateration. The line connecting each cross to a circle indicates which estimated coordinate corresponds to which measured coordinate. Cyan colored nodes have been rejected as outliers.

Before we characterize the overall accuracy of the SGD/MR system, we present Figure 5.4.3, which shows the results of a complete SGD/MR system localization trial using the Pinger v.2. It can be compared to the plots produced by the lateration system in Figure 5-3. However we remind the reader that the lateration system was tested with the Pinger v.1, which we now know causes destructive ultrasound interference and outlying measurements. Nonetheless, it is immediately obvious that localization accuracy in Figure 5.4.3 is considerably better than in 5-3.

Let us now turn our attention to a more in depth characterization of error in the SGD/MR system. Two types of time-of-flight range measurement are considered here: real pings recorded by the Pushpin hardware testbed and simulated pings generated by the simulator. Real pings from both revisions of the Pinger have been included, as well as three different error models for generating simulated pings including no error, gaussian error with a variance of 0.57-cm, and gaussian mixture error (GM error) that approximates the distribution in Figure 5-2. Outlier rejection was active throughout these tests. The percentage of unlocalized nodes that were rejected as outliers has been included in the results. The error statistics for overall localization accuracy appear in Table 5.3. The two most important statistics in this table are the mean (2.30-cm) and standard deviation (2.36-cm) of error using the Pinger v.2. *These are our final results for the Pushpin SGD/MR Localization System.*

It is instructive to compare the localization error using simulated and real pings. First, it is reassuring that simulated pings with no error lead to extremely low localization error (0.06-cm). This ensures us that the mesh relaxation and lateration algorithms do indeed yield exact results when distance measurements are also exact.

Next, simulated pings with a Gaussian error model produce results with a localization error that is approximately 1-cm less than the error produced by real pings from the Pinger v.2. The difference in these values is less that one standard deviation. It is encouraging to find that the error due to real pings is slightly higher that the error due to simulated pings, since the Gaussian error model is meant to be the simplest possible model of ping error.

Simulated pings with the multi-modal (Gaussian mixture) error (which were based

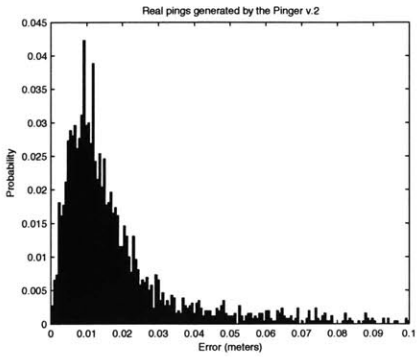
	Real Pings		Simulated Pings		
	Pinger v.2	Pinger v.1	No Error	Gaussian Error	GM Error
Mean	**2.30**	7.52	0.06	1.26	3.65
Median	1.69	6.56	0.04	1.08	2.66
Std Dev	**2.36**	4.75	0.07	0.82	2.99
Min	0.20	1.00	0.01	0.11	0.30
Max	13.5	22.6	0.47	4.39	13.1
% Unlocalized	10%	29%	0%	5%	3%

Table 5.3: Localization statistics for all nodes (anchors and passive nodes). Statistics are averaged over several trials and Pinger configurations. All data is fit using the LLSE. Measurements are in centimeters.

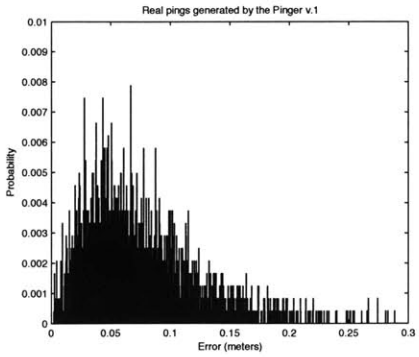
on a characterization of error using the Pinger v.1) are nicely bracketed by the real ping results for the Pinger v.1 and v.2. This demonstrates that localization error does degrade when you consider a multimodal distribution such as the one in figure 5-2. Once again, it is not surprising that the real measurements using the Pinger v.2 are slightly higher than the simulated measurements. The GM error model does not take other error inducing phenomenon into account such as destructive interference of the sonar signal known to be caused by the Pinger v.1.

It is also interesting to note that SGD/MR localization error using the Pinger v.1 is actually *higher* than the error values for the Pushpin Lateration System in Table 5.3 (recall that the Pushpin Lateration System was also tested with the Pinger v.1). This is a very significant result because it suggests that: (1) The assumption made in the Lateration system that the Pinger must be directly above some node in the network is valid in the extremely dense Pushpin network, and (2) The SGD/MR system is more sensitive to the outlying measurements generated by the Pinger v.1 than the Lateration system.

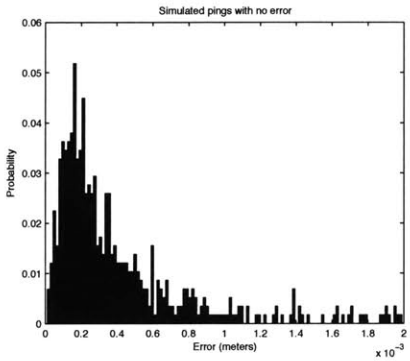
It should also be pointed out that the *median* node error is always lower than the mean node error. This is generally an indication that the *majority* of nodes have a lower error than the average would suggest. Put another way, the average is artificially inflated by a few nodes that have unusually high error. Based on our observations, we believe that these nodes are minor outliers that are missed by the outlier detection mechanisms. Therefore, we would like to emphasize that the *median* error is actually



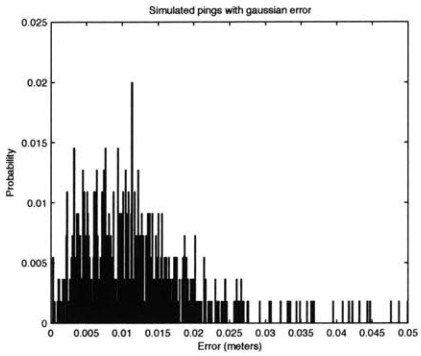
(a) Real Pings: Pinger v.2



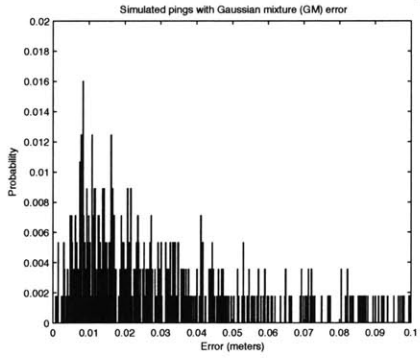
(b) Real Pings: Pinger v.1



(c) Simulated Pings: No Error



(d) Simulated Pings: Gaussian Error



(e) Simulated Pings: GM Error

Figure 5-7: Error distributions for all nodes (anchors and passive nodes) that have been localized via the Pushpin SGD/MR system. Note that axis ranges vary between sub-figures. Also, all five distributions have heavy tails that have been truncated on the right side of the graph.

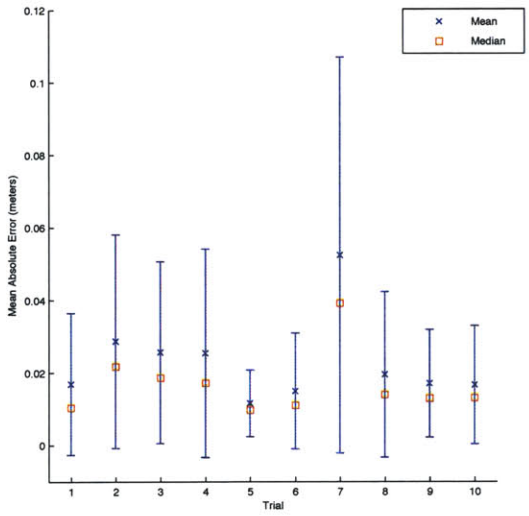


Figure 5-8: Error characteristics for each of ten time-of-flight data sets recorded on the Pushpin hardware test bed using the Pinger v.2. Error bars show one standard deviation from the mean.

a better indicator of the position error experienced by most nodes in the network.

This evaluation is corroborated by the statistical distribution of localization error, which is shown for each approach in Figure 5-7. The distributions in these graphs are all heavy tailed. In some cases, the tale extends out to several tens of centimeters (the tails in many of the graphs were truncated for viewing purposes). These heavy tails inflate the localization mean while the median remains closer to the peak of the distribution where the majority of nodes are best represented.

Finally, Figure 5-8 gives a more introspective look into how accuracy varies between different Pinger configurations. The figure shows separate localization statistics for each of the 10 individual time-of-flight data sets collected on the Pushpin hardware test bed using the Pinger v.2. An interesting feature of this plot is that certain data sets consistently produce localization error with a low mean and small standard deviation, while other data sets, particularly trial 7, produced varied results with a higher mean and significantly larger standard deviation. We observed that these data sets had a larger number of bad (inaccurate and outlying) ping measurements, and that this resulted in bad anchor placement, which in turn led to bad lateration

results in the secondary phase of localization. We also noticed that our outlier detection algorithms would become confused when there were many nodes with outlying coordinates. A common behavior was for a node with a good set of coordinates to be rejected if several of its neighbors were outliers. It is clear from these graphs that our localization approach grows more inconsistent (i.e. the standard deviation between trials increases) as the average error increases.

5.4.4 Outlier Rejection

Outlier rejection is a critical feature of the Pushpin localization system that enables high localization accuracy despite the persistence of bad sensor measurements. Even the very best ranging techniques are prone to spurious errors (and ours is far from being the best). In this section we attempt to quantify the effects of these errors on localization accuracy and the effectiveness of our outlier rejection techniques at eliminating them. We separately consider the two types of outlier rejection used in the SGD/MR system: (1) Rejection of outlying time-of-flight measurements prior to the primary phase of localization, and (2) rejection of outlying coordinate estimates after the second phase of localization. However, we would like to emphasize that both of these schemes must be used in concert for best results.

Rejecting Time-of-Flight Outliers

Let us first consider the effectiveness of a preprocessing step that attempts to identify and reject bad time-of-flight measurements (Section 4.2.1). The outlier rejection threshold is the number of standard deviations from the median of neighboring nodes' measurements for a given Ping. In our test, we tried 10 different values for this threshold that varied between 0.5 and 3.2. Ten localization trials were run per threshold value. The averaged trials are plotted in Figure 5-9. It should be noted that, for this test, the threshold for coordinate outlier rejection (the post-processing step) was set to 10, which essentially turns it off. This was done so as not to adulterate these results with another type of outlier detection.

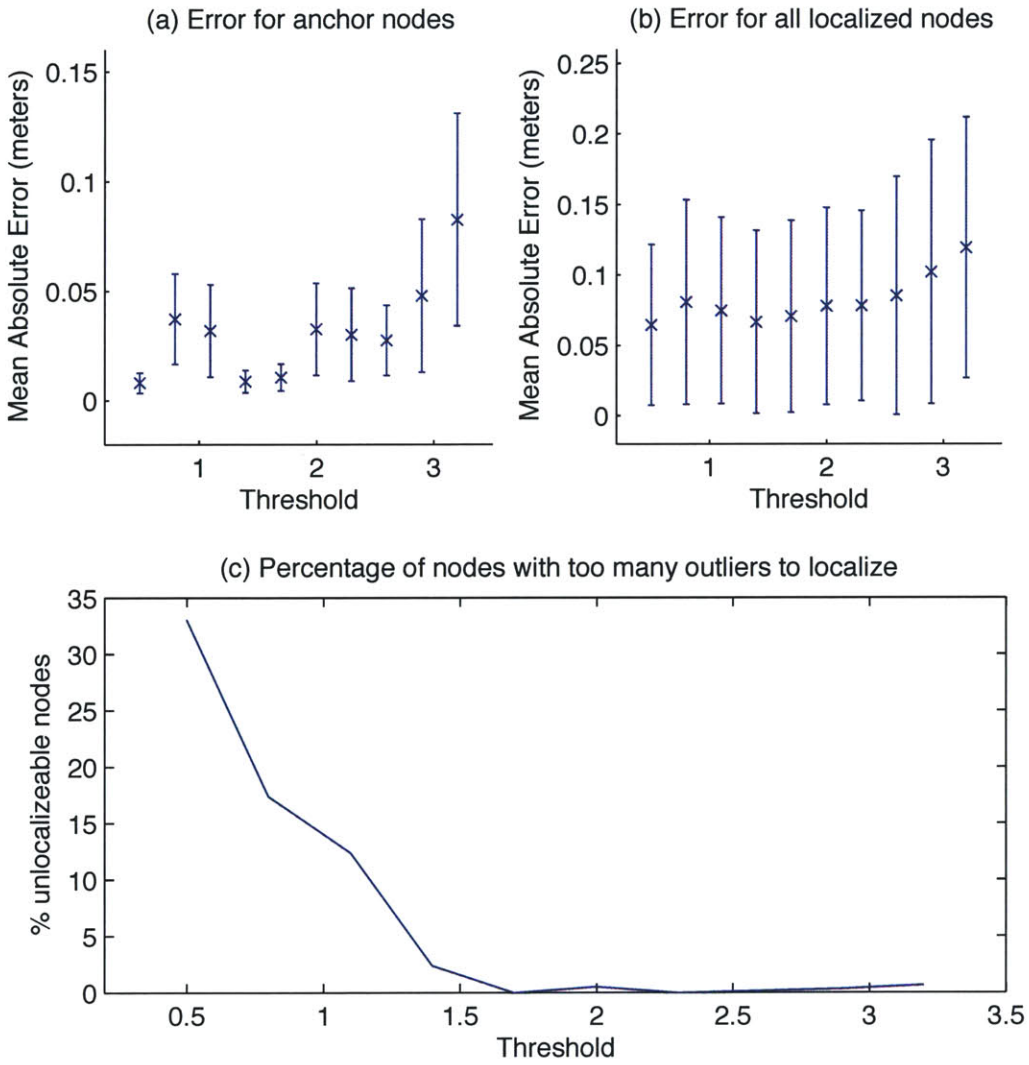


Figure 5-9: The effects of a preprocessing step that rejects outlying time-of-flight measurements. The threshold for rejection is the number of standard deviations from the norm of a node's neighobors' measurements.

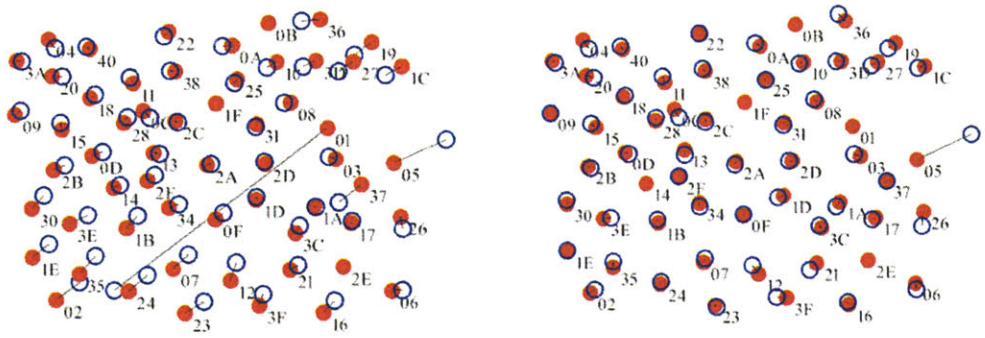
Frame (a) in the figure shows the localization accuracy of the anchors alone. One of the primary purposes of time-of-flight outlier rejection is to ensure that all anchor nodes have a full set of good range measurements. If they do not, the anchor nodes and Pinger event positions will be poorly estimated, and this will effect the accuracy of the passive nodes as well. Frame (a) illustrates this effect nicely. As the outlier threshold is increased, anchor nodes are selected that have increasingly bad ping measurements. Once the threshold exceeds 2.6, the accuracy of the anchors decreases rapidly. Frame (b) shows the effect this has on the passive nodes in the network. As the position error of the anchor nodes (and the global events) increases, the overall localization error also increases.

It is clear that a lower rejection threshold leads to lower localization error. However, frame (c) of Figure 5-9 shows that there is a trade-off to be made between the desired accuracy and the percentage of nodes that cannot localize because they have too many outlying measurements. For thresholds less than 1.5, a significant number of nodes is barred from participating in localization. A reasonable trade-off seems to be to choose a threshold between 1.5 and 2.6.

Rejecting Outlying Coordinates

Outlier rejection is also useful for identifying nodes that have been assigned coordinates that grossly disagree with their measured coordinates. Such nodes are easily detected since their estimated location is very far from the average of their neighbors' estimated locations. This test can easily be applied as an ad hoc post-processing step after localization.

In practice, this type of outlier rejection removes an average of one or two nodes with bad coordinates. Figure 5-10 shows a typical localization trial where node 01 is an outlier. This node skews the overall localization error, but more importantly, it must be detected and rejected so that it does not foul a future localization-enabled application. Consider an example of a poorly localized node that is tracking a military target. It could inadvertently direct a soldier or weapon system to direct their attack to the wrong location. Clearly this outlier must be reliably rejected if the sensor



(a) Before outlier rejection

(b) After outlier rejection

Figure 5-10: Coordinate outlier rejection is critical for selecting and rejecting nodes whose coordinates grossly disagree with their neighbors', such as node 01 in figure (a) above. Figure (b) shows the network after 01 has been automatically rejected. However a minor outlier, namely node 05, was missed on this pass.

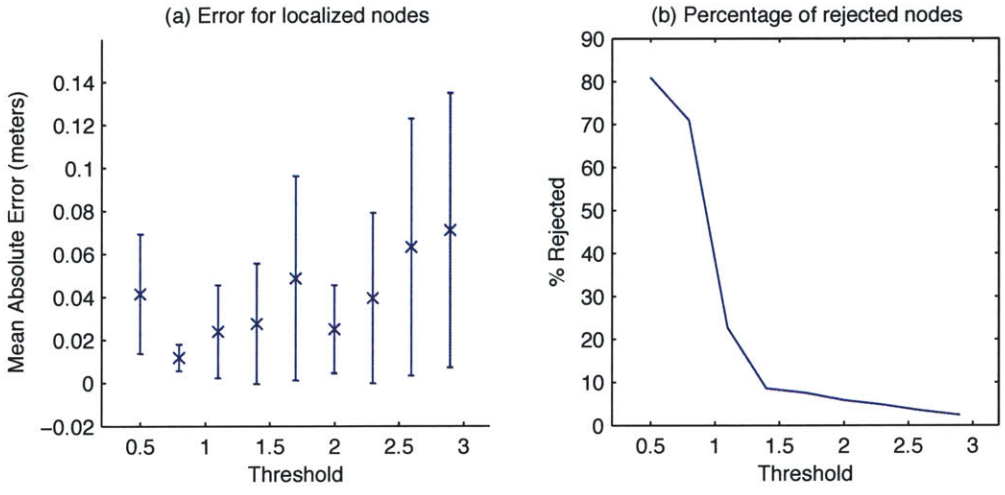


Figure 5-11: The effects of a post-processing step that rejects nodes with outlying coordinates. The threshold for rejection is the number of standard deviations from the median of a node's neighbors' coordinates.

measurements from the network are to be used to make important decisions. Frame (b) shows the network after the outlying node has been disabled using our technique.

The goal of the coordinate outlier rejection test is to assess how localization accuracy and the number of unlocalized sensor nodes varies as a function of rejection threshold. The threshold (which corresponds to the number of standard deviations from the centroid of neighbors' coordinates) was varied from 0.5 to 2.9 in increments of 0.3 standard deviations. Figure 5-11 shows the results of this test. A familiar increase in localization error and decrease in the percentage of unlocalized nodes can be seen as the threshold is increased. We ultimately decided that a threshold of 1.7 provided a good trade-off between the two.

5.4.5 Shearing and Scaling in the SGD/MR Coordinate System

In the results above, we have purposely omitted the localization error using the RTR fit. The reason for this is that the error is extremely high, often in the tens of centimeters. This result surprised us, especially considering the low error using the LLSE fit. The only difference between the LLSE and RTR fits is a compensation for scaling and shearing, so we immediately began to look at the raw (unfit) localization results to see if any shearing and scaling was in evidence. Figure 5-12 shows that there is, in fact, considerable shearing in the coordinates generated by the SGD/MR system. The figure shows a localization trial using simulated pings with no error. The final LLSE fit localization error for this trial was 0.09-cm, however the RTR localization error is 32.38-cm.

We were initially concerned that this shearing and scaling was the result of a bug in our implementation, but we have verified that the distance constraints imposed by the Pinger events are being satisfied. In the image pictured above, the largest disagreement between the measured time-of-flight distance to a Pinger event and the estimated distance to the Pinger (that is, the norm of the difference between the estimated coordinates of a node and a Pinger event) is 0.01-cm. Hence, we

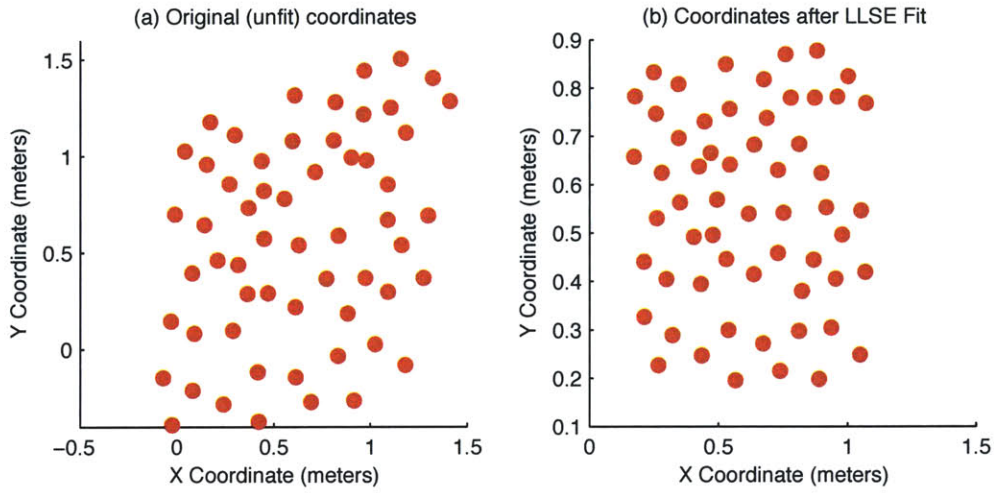


Figure 5-12: The SGD/MR system produces coordinates with a moderate amount of shearing and scaling without violating any of the distance constraints imposed by the Pinger events. The shearing and scaling is completely corrected for by the LLSE fit.

can conclude that the skewed coordinates in the figure do satisfy the constraints imposed by the localization problem formulated in Chapter 3. This may imply that a unique, rigid set of coordinates for the sensor nodes does not exist for the problem we have formulated. It is possible that the correct solution may come from a better understanding of the theory of the rigidity of a graph. An introduction to rigidity theory as it applies to sensor network localization can be found in [63].

In the meantime, the shearing and scaling of the localized coordinates has significant implications. First, the usefulness of the coordinates generated by SGD/MR for sensor network applications is reduced. In particular, the sensor network cannot run any application that depends on a distance metric that reflects physical distances in the real world, as the distance metric in the localized coordinate system cannot be guaranteed to be accurate. In other words, a sensor node cannot measure its range to an object and simply report its range. This measurement is likely to be distorted by the shearing and scaling of the coordinate system. Second, if location-labeled data is routed off the sensor network, it must be related to an external coordinate system via a transformation that is found by computing the LLSE fit. However, this is not a severe restriction, since some transformation from the frame of reference of the sensor

network to a global frame of reference would have been needed, regardless.

Despite the difficulties it may cause, shearing and scaling of the localized coordinates in the most interesting result in our SGD/MR characterization. It suggests that a stronger set of constraints are necessary to create a rigid, unique set of coordinates that is invariant up to a simple translation, rotation, and reflection.

Chapter 6

Closing Remarks

In this work, we have demonstrated two complete systems for localizing a network of roughly 60 sensor nodes. The first, a system based solely on lateration, has been shown to achieve an average localization error of 4.93-cm with an error standard deviation of 3.05-cm. The diagonal wipe demonstration in Section 5.3.1 provides additional visual verification that this system is functioning well. The disadvantage of the lateration system is that it assumes that the Pinger is held directly over some node in the network. While this approximation has been shown to work well in our extremely dense network, it may not be borne out in a practical deployment.

Although the low localization error in the lateration system is promising, there is plenty of room for improvement and additional verification. As we have previously mentioned, some recent papers [49, 57] show via computation of the Cramer-Rao bound (CRB) that the error variance of the location estimate can be very high near the edges of the convex hull created by the anchor nodes. It is likely that our setup is subject to similar limitations on localization accuracy, and this should be checked via computation of the CRB. Localization accuracy could also be improved through the use of more than three pings (and, correspondingly, more than three anchors). As we mentioned in Section 3.2, lateration can easily be made to incorporate additional measurements via a linear least squares approximation. It would also undoubtedly lower localization error in the Lateration system if we tested it using the Pinger v.2, which produces drastically lower localization error on the SGD/MR system. This

would also provide a firm basis for comparing the Lateration and SGD/MR systems more directly.

The second localization system we have proposed employs a combination on non-linear and linear techniques for localization. The SGD/MR system yields more accurate measurements than the lateration system alone: it produces coordinates with an average error of 2.30-cm and an error standard deviation of 2.36-cm. In the SGD/MR system, no assumptions are made about the position of the Pinger. The SGD/MR system also features additional pre- and post-processing steps for rejecting outlying measurements and coordinates.

While the SGD/MR generally performed as expected, it did diverge from our expectations in some of the tests. For example, it was surprising to learn that the coordinate systems produced by the SGD/MR system are subject to some amount of shearing. This is certainly a phenomenon we would have liked to understand better if there had been more time. In the meantime, we have verified that the distance constraints between the nodes and the Pinger are satisfied in the sheared coordinate system. Since the constraints are satisfied, it would seem that these sheared coordinate systems are valid solutions to the localization we have proposed. A more rigorous inquiry into the number and type of constraints necessary to create a *rigid* coordinate system should be pursued in future research.

Convergence of the mesh relaxation algorithm is a major subject of this thesis. The purpose of spectral graph drawing in the SGD/MR system is to help guarantee this convergence by avoiding false minima. We did not expect to find that mesh relaxation actually performs very well without the aid of a good initial guess. We postulate that this is because our mesh relaxation problem is fairly simple and unconstrained – it is allowed to relax in a three dimensional space and there are only 15 vertices and 50 edge constraints. This is corroborated by our early tests where we ran the same mesh relaxation simulation in a 2-D space. In these tests, we frequently observed the mesh settling into a false, "folded" solution – a phenomenon that has been corroborated by other researchers[54]. Incidentally, although spectral graph drawing has not turned out to be critical for avoiding local minima in our setup, we have demonstrated

in Section 5.4.1 that SGD is accurate enough to have some utility a stand-alone localization technique.

There are some obvious avenue for improving SGD/MR. First, the ability to make use of more than 5 Pinger events could improve overall localization accuracy. This is a simple matter, since the algorithms used in SGD/MR readily generalize to having more pings (by adding more vertices in the SGD and mesh relaxation problems and by utilizing linear least squares to approximate the lateration solution). Utilizing extra pings, it should be possible to minimize localization error to roughly 1-cm, which is the theoretical limit of the accuracy of detecting an ultrasound signal on a unipolar rising-edge discriminator with a fixed threshold.

Next, while mesh relaxation has worked well in our tests, it is rarely chosen for generic optimization problems because it converges relatively slowly compared to other methods. In our tests, it took an average of 5,000 iterations of relaxation before the mesh converged. Several more efficient approaches are well known in the field of optimization, and some of these have recently been applied to the problem of localizing sensor nodes. Some work in this area includes semi-definite programming [14], non-linear least squares [43], and distributed Kalman filters [58]. A third version of the Pushpin localization system would likely utilize one of these methods rather than mesh relaxation.

Finally, while the lateration system has been implemented on the Pushpin hardware testbed, the SGD/MR system has not. Doing so would demonstrate that localization using non-linear techniques are within the computational abilities of a sensor node with an 8-bit micro-controller. Much of the work in this respect has already been completed, since the code written for the virtual Pushpins in the Pushpin Simulator is very similar to the code which could ultimately run on the Pushpins. In fact, a tiny statistics and linear algebra library called the *PushpinMath* library was written in C for the virtual Pushpins that will run without modification on the real Pushpins. All of the computation carried out in this work including the eigenvector computations in spectral graph drawing utilize *PushpinMath*, hence porting to these algorithms to the real Pushpins would be a simple matter.

Both Pushpin localization systems have been shown to be effective for localizing nodes in the Pushpin hardware test bed. However, they differ in a number of respects. The Pushpin Lateration system is simple and light-weight. Computationally, it is far less intensive than the SGD/MR approach, since it relies only on lateration. It is also fairly accurate in our setup, though accuracy is expected to drop sharply in a sparse sensor network where the constraint on the Pinger location cannot be easily enforced.

The primary advantage of the SGD/MR system is its generality. Namely, it does not place any restriction on the location of global events. SGD/MR also yields very low localization error and error standard deviation when the Pinger v.2 measurements are used, though this comes at the cost of a complicated system design and computationally intensive localization algorithms. Nonetheless, we believe the SGD/MR system to be within the abilities of an 8-bit micro-controller.

6.1 Future Work

Having demonstrated a basic ability to localize nodes in our hardware testbed, several avenues of future work now lay open to us. First, the localization algorithm should rely as little as possible on the mechanism for generating global stimuli and instead treat such stimuli as parts of the environment rather than additional infrastructure. This work at least shows progress toward this goal by obviating the need for prior knowledge of the absolute position of the source of a pair of global stimuli. However, we would like to generalize our approach further by instead measuring the *time of arrival* of a *single* global signal. In this scenario, the absolute time origin of the signal becomes another parameter to be estimated. The solution to this higher dimensional search problem requires additional constraints and more computation, but these are readily available either from additional participating nodes, or from additional global events. The SGD, mesh relaxation, and lateration algorithms all readily generalize to giving coordinates in higher dimensions. Furthermore, it may even be possible to localize nodes in both space *and* time (i.e. to estimate an offset and clock skew correction coefficient in addition to a node's coordinates) simultaneously if enough

constraints are available from extra global events.

As mentioned earlier, localization is a fundamental tool for building applications with sensor networks. Now that we are well on our way to having the tool, the challenge now is to exploit it. An obvious choice as to where to begin in this regard is to create a more compelling and visually complex display application than the diagonal wipe demonstration presented in Section 5.3.1. A first step in this direction might include using the pushpin array to do both simple shape recognition and display. This could ultimately result in the use of the Pushpin sensor network as a programmable retina[46].

It has also been proposed that location-aware, tightly synchronized nodes with weak radio transmitters may collaborate to beamform a stronger radio signal[35]. Such a network would be able to transmit information over a great distance to a remote receiver. Furthermore, highly accurate synchronization would make it possible to employ the microphone-equipped nodes as an acoustic phased array[13, 20]. Such a system would demonstrate the ability to first localize itself, and then to further localize events in its environment, in this case audio sources[62].

6.2 Conclusion

With the completion of this work, a chapter of Pushpin development draws to a close. Much has been accomplished during this time. We now have an enhanced platform for sensor network development with new hardware and software for debugging and programming, a simulation environment for testing new algorithms, and a choice of two location services that will enable future location-aware applications.

In addition, this research has demonstrated the efficacy of three distinct localization algorithms as well as supplemental algorithms for outlier rejection, communication, and distributed display. Most importantly, we have demonstrated these algorithms working together in a unified system for localization that produces consistent results and low localization error.

Appendix A

Schematic Diagrams

Designator	Purpose
INT0	External interrupt line 0
INT1	External interrupt line 1
ADC0	Analog-to-digital converter line 0
ADC1	Analog-to-digital converter line 1
ADC2	Analog-to-digital converter line 2
ADC3	Analog-to-digital converter line 3
PHOTO	Photo transistor
MIC	Electret microphone
SONAR	10R-40P sonar receiver - *www.americanpiezo.com*

Table A.1: Key for Figure A-1

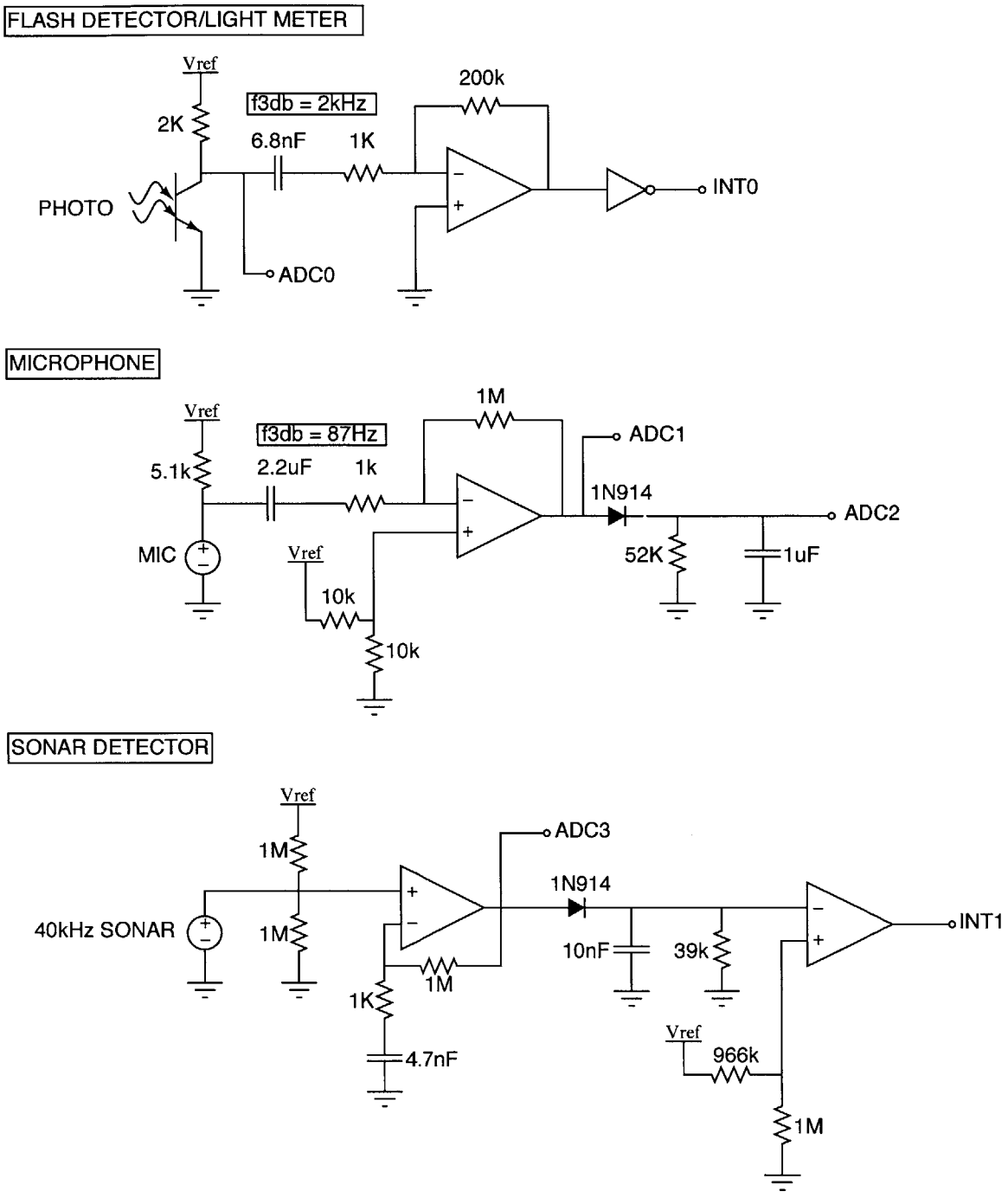


Figure A-1: Schematic diagram of the Pushpin Time of Flight Expansion Module

Bibliography

[1] Pushpin Computing Web Page. http://www.media.mit.edu/resenv/pushpin, 2004.

[2] Amorphous computing website. http://www.swiss.ai.mit.edu/projects/amorphous/, 2005.

[3] The network simulator – ns-2. http://www.isi.edu/nsnam/ns/, 2005.

[4] Python Programming Language. http://www.python.org, 2005.

[5] SDCC - Small Device C Compiler. http://sdcc.sourceforge.net, 2005.

[6] Silicon Laboratories. http://www.silabs.com, 2005.

[7] Swarm. http://www.swarm.org/intro.html, 2005.

[8] Tinyos community forum. http://www.tinyos.net/, 2005.

[9] Harold Abelson, Don Allen, Daniel Coore, Chris Hanson, George Homsy, Jr. Thomas F. Knight, Radhika Nagpal, Erik Rauch, Gerald Jay Sussman, and Ron Weiss. Amorphous Computing . *Communications of the ACM*, May 2000.

[10] Unknown Author. Efficient Processing of Data for Locating Lightning Strikes. Technical Brief KSC-12064/71, NASA Kennedy Space Flight Center, Unknown Year.

[11] J. Bachrach. Position determination in sensor networks. to appear in Stojmenovic I (Ed), Mobile ad hoc networking, John Wiley and Sons, 2004.

[12] Paramvir Bahl and Venkata N. Padmanabhan. RADAR: An In-Building RF-based User Location and Tracking System. In *Proceedings of IEEE INFOCOM*, 2000.

[13] Sumit Basu, Steve Schwartz, and Alex Pentland. Wearable Phased Arrays for Sound Localization and Enhancement. In *Fourth International Symposium on Wearable Computers (ISWC'00)*, page 103, 2000.

[14] Pratik Biswas and Yinyu Ye. Semidefinite programming for ad hoc wireless sensor network localization. In *Proceedings of the 2nd ACM international conference on wireless sensor networks and applications*, pages 46 – 54, 2004.

[15] Ingwer Borg. *Modern Multidimensional Scaling: Theory and Applications.* Springer, 1997.

[16] Michael Broxton, Joshua Lifton, and Joseph Paradiso. Localizing a Sensor Network via Collaborative Processing of Global Stimuli. In *Proceedings of the European Conference on Wireless Sensor Networks (EWSN) 2005*, 2005.

[17] Nirupama Bulusu, Vladimir Bychkovskiy, Deborah Estrin, and John Heidemann. Scalable, ad hoc deployable rf-based localization. In Grace Hopper Celebration of Women in Computing Conference 2002, Vancouver, British Columbia, Canada., October 2002.

[18] William Butera. *Programming a Paintable Computer.* PhD thesis, Massachusetts Institute of Technology, 2002.

[19] Ramesh Govindan Chalermek Intanagonwiwat and Deborah Estrin. Making Sensor Networks Practical with Robots. In *Sixth Annual International Conference on Mobile Computing and Networking*, August 2000.

[20] J.C. Chen, L. Yip, J. Elson, H. Wang, D. Maniezzo, R.E. Hudson, K. Yao, and D. Estrin. Coherent Acoustic Array Processing and Localization on Wireless Sensor Network. *Proceedings of the IEEE*, 98(8), August 2003.

[21] Crossbow Technologies. Intertial, Gyro, and Wireless Sensors from Crossbow. http://www.xbow.com, 2003.

[22] James W. Demmel. *Applied Numerical Linear Algebra.* SIAM, 1997.

[23] Department of Integrative Medical Biology. Laboratory of Dexterous Manipulation.
http://www.humanneuro.physiol.umu.se/default.htm, 2004.

[24] S. Deutsch and A. Deutsch. *Understanding the nervous system: an engineering perspective.* IEEE Press, 1993.

[25] M.A. Diftler and R.O. Ambrose. ROBONAUT, A Robotic Astronaut Assistant. ISAIRAS conference 2001, 2001.

[26] Lance Dohetry, Kristofer S. J. Pister, and Laurent El Ghaoui. Convex Position Estimation in Wireless Sensor Networks. In *Proceedings of Infocom 2001*, 2001.

[27] L. Girod, V. Bychkobskiy, J. Elson, and D. Estrin. Locating tiny sensors in time and space: A case study. In *Proceedings of the International Conference of Computer Design (ICCD) 2002*, 2002.

[28] Lewis Girod and Deborah Estrin. Robust Range Estimated using Acoustic and Multi-modal Sensing. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2001)*, 2001.

[29] Craig Gotsman and Yehuda Koren. Distributed graph layout for sensor networks. In *Proceedings of the International Symposium on Graph Drawing*, 2004.

[30] Oliver Guenther, Tad Hogg, and Bernardo A. Huberman. Learning in multiagent control of smart matter. AAAI-97 Workshop on Multiagent Learning, 1997.

[31] K. M. Hall. An r-dimensional quadratic placement algorithm. *Management Science*, 17:219 – 229, 1970.

[32] T. He, C. Huang, B. Blum, J. Stankovic, and T. Abdelzaher. Range-Free Localization Schemes in Large Scale Sensor Networks. In *Proceedings of the 9th annual international conference on Mobile computing and networking*, pages 81–95, 2003.

[33] Seth Hollar. COTS Dust. Master's thesis, University of California at Berkeley, 2000.

[34] Andrew Howard, Maja J Mataric, and Gaurav Sukhatme. Relaxation on a Mesh: a Formalism for Generalized Localization. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Oct 2001.

[35] A. Hu and S. D. Servetto. dfsk: Distributed frequency shift keying modulation in dense sensor networks. In *Proceedings of the IEEE International Conference on Communications (ICC)*, 2004.

[36] Roland S. Johansson and Ake B. Valibo. Tactile sensory encoding in the glabrous skin of the human hand. *Trends in Neuroscience*, 6(1):27–32, 1983.

[37] Yehuda Koren. On spectral graph drawing. In *Proceedings of the 9th International Computing and Combinatorics Conference (COCOON'03)*, 2003.

[38] Jack B. Kuipers. *Quaternions and Rotation Sequences.* Princeton University Press, 1999.

[39] Anthony LaMarca, Waylon Brunnette, David Koizumi, Matthew Lease, Stefan B. Sigurdsson, Kevin Sikorski, Dieter Fox, and Gaetano Borriello. Making Sensor Networks Practical with Robots. In *Pervasive Computing First International Conference*, pages 152–166, August 2002.

[40] Koen Langendoen and Niels Reijers. Distributed localization in wireless sensor networks: a quantitative comparison. *The International Journal of Computer and Telecommunication Networking*, 43(4):499 – 518, November 2003.

[41] Joshua Lifton. Pushpin Computing: A Platform for Distributed Sensor Networks. Master's thesis, Massachusetts Institute of Technology, 2002.

[42] Joshua Lifton, Deva Seetharam, Michael Broxton, and Joseph Paradiso. Pushpin Computing System Overview: a Platform for Distributed, Embedded, Ubiquitous Sensor Networks. In *Proceedings of the International Conference on Pervasive Computing*, August 2002.

[43] Randolph L. Moses, Dushyanth Krishnamurthy, and Robert M. Patterson. A Self-Localization Method for Wireless Sensor Networks. *EURASIP Journal on Applied Signal Processing*, pages 348 – 358, 2003.

[44] Radhika Nagpal. Organizing a Global Coordinate System from Local Information on an Amorphous Computer. A.I. Memo 1666, MIT Artificial Intelligence Laboratory, 1999.

[45] Radhika Nagpal, Howard Shrobe, and Jonathan Bachrach. Organizing a Global Coordinate System from Local Information on an Ad Hoc Sensor Network. In *2nd International Workshop on Information Processing in Sensor Networks (IPSN '03)*, April 2003.

[46] F. Paillet, D. Mercier, and T. Bernard. Second Generation Programmable Artificial Retina. In *Proceedings of the IEEE ASIC/SOC Conference*, pages 304–309, 1999.

[47] Thomas V. Papakostas, Julian Lima, and Mark Lowe. A large area force sensor for smart skin applications. In *Proceedings of the IEEE Sensors 2002 Conference*, 2002.

[48] Joseph Paradiso, Joshua Lifton, and Michael Broxton. Sensate Media – multimodal electronic skins as dense sensor networks. *BT Technology Journal*, 22(4), October 2004.

[49] Neal Patwari and Alfred O. Hero III. Using Proximity and Quantized RSS for Sensor Localization in Wireless Networks. In *Proceedings of the 2nd ACM*

international conference on Wireless sensor networks and applications, pages 20 – 29, 2003.

[50] Neal Patwari and Alfred O. Hero III. Manifold learning algorithms for localization in wireless sensor networks. In *Proceedings of the 2004 IEEE Int. Conf. on Acoustics, Speech, and Signal Processing (ICASSP)*, May 2004.

[51] Christina Peraki and Sergio D. Servetto. On the maximum stable throughput problem in random networks with directional antennas. In *Proceedings of the ACM MOBIHOC 2003*, 2003.

[52] Arkady Pikovsky, Michael Rosenblum, and Jurgen Kurths. *Synchronization: A universal concept in nonlinear sciences.* Cambridge University Press, 2001.

[53] Willian H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. *Numerical Recipes in C: The Art of Scientific Computing.* Cambridge University Press, 1988.

[54] Nissanka B. Priyantha, Hari Balakrishnan, Erik Demaine, and Seth Teller. Anchor-Free Distributed Localization in Sensor Networks. Tech Report 892, MIT Laboratory for Computer Science, 2003.

[55] Nissanka B. Priyantha, Anit Chakraborty, and Hari Balakrishnan. The cricket location-support system. In *Mobile Computing and Networking*, pages 32–43, 2000.

[56] Chip Redding, Floyd A. Smith, Greg Blank, and Charles Cruzan. Cots mems flow-measurement probes. *Nanotech Briefs*, 1(1):20, January 2004.

[57] A. Savvides, W. Garber, S. Adlakha, R. Moses, and M. B. Srivastava. On the Error Characteristics of Multihop Node Localization in Ad-Hoc Sensor Networks. In *Proceedings of the 2nd International Workshop on Information Processing in Sensor Networks (IPSN'03)*, 2003.

[58] A. Savvides, H. Park, and M. Srivastava. The Bits and Flops of the N-Hop Multilateration Primitive for Node Localization Problems. In *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, pages 112–121, 2002.

[59] Andreas Savvides, Chih-Chieh Han, and Mani B. Srivastava. Dynamic fine-grained localization in ad-hoc networks of sensors. In *Mobile Computing and Networking*, pages 166 – 179, 2001.

[60] M. Sergio, N.Manaresi, M. Tartagni, R. Guerrieri, and R.Canegallo. A textile based capacitive pressure sensor. In *Proceedings of the IEEE Sensors 2002 Conference*, 2002.

[61] Yi Shang, Wheeler Ruml, Ying Zhang, and Markus P. J. Fromherz. Localization from Mere Connectivity. In *Proceedings of the 4th ACM international symposium on mobile ad hoc networking and computing*, pages 201 – 212, 2003.

[62] Xiaohong Sheng and Yu-Hen Hu. Sensor deployment for source localization in wireless sensor network system. submitted to The First ACM Conference on Embedded Networked Sensor Systems, 2003.

[63] Anthony Man-Cho So and Yinyu ye. Theory of Semidefinite Programming for Sensor Network Localization. To appear in SODA 2005, 2005.

[64] Christopher Taylor, Ali Rahimi, Jonathan Bachrach, and Howard Shrobe. Simultaneous localization and tracking in an ad hoc sensor network. Submitted to IPSN 2005, 2005.

[65] Ben L. Titzer, Daniel K. Lee, and Jens Palsberg. Avrora: Scalable sensor network simulation with precise timing. In *Proceedings of IPSN 2005, Fourth International Conference on Information Processing in Sensor Networks*, 2005.

[66] Andy Ward, Alan Jones, and Andy Hopper. A New Location Technique for the Active Office. *IEEE Personal Communications*, 4(5):42–47, October 1997.

[67] Brett Warneke, Matt Last, Brian Liebowitz, and Krisofer S. J. Pister. Smart dust:communicating with a cubic-millimeter computer. *Computer*, 32:44 – 51, 2001.

[68] Rob Weiss. *Cellular Computation and Communications using Engineered Genetic Regulatory Networks.* PhD thesis, Massachusetts Institute of Technology, 2001.

[69] Kamin Whitehouse. The Design of Calamari: an Ad-hoc Localization System for Sensor Networks. Master's thesis, University of California at Berkeley, 2002.

[70] Yong Xu, Yu-Chong Tai, Adam Huang, and Chih-Ming Ho. Ic-integrated flexible shear-stress sensor skin. Solid-State Sensor, Actuator and Microsystems Workshop, 2002.