An Ultrasonic Compass for Context-Aware Mobile Applications

by

Kevin John Wang

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

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2004

Author .....
Department of Electrical Engineering and Computer Science
May 7, 2004

Certified by .....
Seth Teller
Associate Professor
Thesis Supervisor

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

An Ultrasonic Compass for Context-Aware Mobile Applications

by

Kevin John Wang

Submitted to the Department of Electrical Engineering and Computer Science
on May 7, 2004, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

If we are to realize the everyday benefits promised by pervasive computing and context-aware applications, we must first develop the infrastructure to provide contextual location and orientation information through pervasive computing elements. I lay the foundations for leveraging the Cricket indoor location system to supply orientation information. I first characterize the use of ultrasound in Cricket for distance and orientation measurements. I then propose a set of methods to calculate 3-DOF orientation from an array of well placed ultrasonic sensors operating in the Cricket system. I design and implement a prototype of this Cricket Compass using a combination of hardware and software and demonstrate end-to-end functionality of the system.

Thesis Supervisor: Seth Teller
Title: Associate Professor

Acknowledgments

First, I thank Professor Seth Teller for giving me the opportunity and support to pursue this research. His enthusiasm and never-ending stream of ideas have been both educational and of great encouragement to me while under his supervision.

I also thank the rest of the Cricket Team for making this entire endeavor possible. I especially thank Bodhi Priyantha for his help with everything Cricket and Compass related.

I braved graduate student life together with my office-mates Roshan Baliga and Jonathan Wolfe. Thanks to both for making the lab fun, even during those times when it should not have been.

Finally, I am deeply grateful to my family for always being there. Without them I would have never have made it this far.

Contents

1 Introduction **13**
- 1.1 Motivation . . . 14
- 1.2 The Design Space . . . 16
- 1.3 Overview . . . 18

2 Steps Towards Orienting the Cricket Compass **19**
- 2.1 Existing Orientation Systems . . . 19
  - 2.1.1 Constellation . . . 20
  - 2.1.2 HiBall . . . 20
  - 2.1.3 Whisper . . . 21
  - 2.1.4 Commercial Systems . . . 21
- 2.2 Groundwork for the Cricket Compass . . . 22
  - 2.2.1 Calculating Planar Orientation . . . 22
- 2.3 Challenges in Orienting the Cricket Compass . . . 24
  - 2.3.1 Characterizing Cricket Ultrasound . . . 25
  - 2.3.2 The Differential Distance Problem . . . 28
  - 2.3.3 Disambiguating θ . . . 31
  - 2.3.4 Implementation Experience . . . 32

3 Theory of Operation **37**
- 3.1 Cricket Infrastructure . . . 37
- 3.2 Three-Dimensional Local Orientation . . . 38
- 3.3 Obtaining Accurate Differential Distances . . . 42
- 3.4 Registration to Global Orientation . . . 42
  - 3.4.1 Centroid Relative Coordinates . . . 43
  - 3.4.2 Rotation of the Planes . . . 44
  - 3.4.3 Rotation in the Plane . . . 45

4 The Cricket Compass Prototype **47**
- 4.1 Hardware Design Parameters . . . 48
  - 4.1.1 Sensor Array . . . 48
  - 4.1.2 Analog-to-Digital Conversion . . . 51
- 4.2 Software Design Parameters . . . 52
- 4.3 Testing and Modifications . . . 52
  - 4.3.1 Correlation . . . 53
  - 4.3.2 Filtering . . . 55
  - 4.3.3 Pulse Shaping . . . 56

4.3.4 Error Detection 59

5 Results **61**
5.1 Localizing Beacons 61
5.1.1 Setup and Procedure 61
5.1.2 Analysis 63
5.2 End-to-End Orientation 65
5.2.1 Setup and Procedure 66
5.2.2 Analysis 67
5.3 Future Work 69

6 Contributions **71**

A Compass Hardware Design **73**
A.1 Analog Ultrasound Gain Circuit 73
A.2 Sensor Array Specifications and Naming 74
A.3 Murata MA40S4R Ultrasonic Sensor Information 75

B Compass Functionality **77**
B.1 Setup 77
B.2 Demonstration 77
B.3 MATLAB functions 78

List of Figures

1-1 Block diagram overview of the Cricket Compass. 17

2-1 Determining the angle of orientation along the horizontal plane. 23
2-2 A rotated compass leads to a difference in distances between the beacon and each of the receivers. 24
2-3 An amplified ultrasound pulse on a Cricket listener. 26
2-4 A photo of the version 2 Cricket, which can function as a listener or a beacon. 27
2-5 Receivers R_1 and R_2 can measure the differential distance from a far-away beacon. 29
2-6 An observed phase Δ can actually correspond to an infinite number of possible real phases, all separated by 2π. 30
2-7 θ is ambiguous; there are two beacon positions B_1, B_2 that result in the same θ at the compass. 31
2-8 Two ultrasound receivers mounted on a precision rotating platform. 32
2-9 A comparison of ultrasound receiver compass performance. 34

3-1 Determining the angle θ between the vector to the beacon and the axis formed by the receiver pair. 38
3-2 The intersection of two circles presents two possible beacon positions. . . . 39
3-3 Determining the position of a beacon using an array of receivers. 40
3-4 The second rotation step in determining absolute orientation. 45

4-1 A block diagram of the Cricket Compass implementation. 47
4-2 A few possible ultrasound sensor array geometries. 49
4-3 A comparison of sensor pair coverage for different geometries. 50
4-4 A photo of the actual compass hardware prototype. 51
4-5 A plot of four ultrasound waveforms coming from the Compass hardware. . 53
4-6 A plot of the six inter-sensor normalized cross correlations for varying time delays. 54
4-7 A plot of error in calculated orientation using two different methods for the differential distance measurement. 55
4-8 Plots of different types of ultrasound signals and resulting normalized cross correlations. 57
4-9 The standard Cricket ultrasound pulse and the modified pulse for the Compass. 58
4-10 A visualization of the differential distances on sensor pairs. 59

5-1 Annotated photos of beacon localization experiment. 62
5-2 Illustration of Compass rotation in the beacon localization experiment. . . . 63
5-3 A plot of beacon localization using the Compass. 64

5-4 Annotated photos of Compass end-to-end demonstration. 65
5-5 Annotated photos depicting the Compass and Cricket coordinate spaces in the end-to-end demonstration. 66
5-6 Visualization of estimated Compass end-to-end orientation. 68

A-1 Ultrasound analog gain circuit. 73
A-2 Numbering scheme and coordinate axes defined for the ultrasound sensor array. 74
A-3 Murata ultrasonic sensor physical dimensions. 75
A-4 Murata ultrasonic sensor frequency response. 75
A-5 Murata ultrasonic sensor directivity in sensitivity. 75

List of Tables

5.1 Mean error and standard deviation of error in each coordinate axis. 65
5.2 Results of end-to-end orientation demonstration. 67

A.1 I mounted the sensors using the circuit board inter-hole spacing of 2.54mm. This table shows the x and y positions of the center of the sensors in grid units where 4 grid units = 1 inter-hole space = 2.54mm. 74
A.2 Sensor pair naming, orientations, and separation distances used for position calculations. 74

Chapter 1

Introduction

If we are to realize the everyday benefits promised by pervasive computing and context-aware applications, we must first develop the foundation systems that will make those applications easy to develop, to deploy, and to maintain. We must develop systems that provide the necessary contextual information with accuracy, robustness, and scalability. We must also address the technical challenges of engineering these systems for use by humans and mobile devices in indoor environments.

Researchers at the MIT Computer Science and Artificial Intelligence Laboratory are developing the Cricket indoor location system, a ubiquitous and precise location infrastructure for pervasive computing. In this thesis I lay the foundations for leveraging the Cricket infrastructure to supply contextual orientation information. I will use the term "Cricket Compass" throughout to denote the collection of hardware designs, algorithms, and software applications that enable the delivery of orientation information through Cricket.

First, I characterize the design constraints imposed by the current revision of Cricket hardware and software and discuss them in relation to the desired properties and design targets of the Cricket Compass. Specifically, I carefully review the properties and manipulation of ultrasound in Cricket used for calculating distance estimates. I then discuss methods for determining orientation by surveying current orientation systems including Priyantha et al.'s seminal work on using ultrasonic phase differentials to infer planar orientation.

I develop methods to accurately measure ultrasonic phase differentials and propose an algorithm to infer three-dimensional orientation with respect to the source of ultrasonic pulses. I design and construct a hardware prototype of the Cricket Compass and demon-

strate end-to-end functionality of the system. Finally, I characterize the performance of the Cricket Compass and outline methods for improvement.

1.1 Motivation

Imagine that you have made an appointment to meet with a professor at his office in MIT's Ray and Maria Stata Center. You have heard that the new Stata Center has a very unique design, and you anticipate that it will be difficult to find your way among the nine twisting and turning floors that house hundreds of researchers. Upon arriving, you find the posted maps to be of limited use as you struggle to orient yourself without the aid of regularities in the architecture or useful landmarks. Directions from people along your way fail to map easily into the real environment. You give up for the moment and grudgingly stop at the information desk to ask for help.

Now imagine that instead of giving you more directions, the helpful staffers supply you with a handheld navigation device. You input your destination, and the device gives you directions in context to an automatically updating map display that orients itself to your current viewpoint. Navigating suddenly becomes easy as you match features and landmarks in your view to those displayed on the device.

During your journey, you also notice a maintenance staffer using a different navigation device to locate and service a faulty floor vent. As the staffer walks along, the device projects an outline of the floor venting pipe so he can locate a malfunctioning junction. Amazed at your recent experience, you start to see how applications inside buildings, such as offices, shopping malls, airports, and homes, have the potential to fundamentally change the way we interact with our immediate environment, in which computing elements will be "ubiquitous" or "pervasive."

The scenario described above highlights the utility of context-aware applications, a compelling class of applications in emerging pervasive computing environments. These applications can adapt their behavior according to an environmental context, such as physical location. Another important environmental context is a device or user's *orientation* with respect to one or more landmarks in a region. A context-aware computing application can benefit from knowing orientation, for instance by providing the ability to adapt a user interface to the direction in which a user is facing. Priyantha et al. [7] and Teller et al. [8]

describe in detail several compelling applications that are made possible by the availability of location information supplemented with orientation information:

1. Wayfinder: This application runs on a handheld computer to aid sighted or blind people in navigation to a destination through an unfamiliar environment. For example, the application could lead building visitors from an entry lobby to an office or seminar room for a meeting.

2. Viewfinder: This application allows a user to point in a direction and specify a desired environmental scope, for example, a range and a sweep angle. Using an active map integrated with a resource discovery system, the application then returns a representation of devices and services in the desired direction and scope. This then enables the user to interact with those services and devices via representations on the map.

3. Information overlay: A user's view of an environment is overlaid with information about objects present in that environment. The view and information presented actively adapt to the orientation of the user. For example, this capability could allow users to "see" through walls by overlaying occluded objects and landmarks on the wall.

4. Virtual tagging: Location and orientation information enables a user to point at objects in a virtual representation of the users environment. The user can now quickly create or modify database information about the location of landmarks and objects in the environment. For example, surveyors and contractors can mark buried service lines and building tenants can log physical plant maintenance requests.

5. Active signage: Location and orientation-aware displays can dynamically configure themselves to display the most critical and pertinent information. For example, a sign could automatically direct users to meetings or talks on the day that they are scheduled. In an emergency, the sign could display a route directing users along an escape route to the nearest exit.

The Cricket Compass can provide the capability required to supply context-aware applications with orientation information. The Cricket Compass provides accurate knowledge of its own position and orientation and can combine with contextual maps to present the

user with contextually rich data, such as the location of resources and the means to reach those resources.

1.2 The Design Space

Obtaining location and orientation information for applications in an indoor environment in an unobtrusive and private manner is a challenging task. Multi-path effects and dead spots inside buildings present harsh conditions to radio signals in indoor environments. A traditional magnetic compass does not work well in many buildings because of electromagnetic interference from computers and monitors. In addition, user-privacy concerns are an important consideration in the successful deployment of these applications. The infrastructure must require minimal administration because there can be potentially several thousand devices in a typical building.

I use the Cricket indoor location system to obtain precise location information. The current Cricket location system uses a network of active beacons and passive listener devices to provide location information accurate to within several centimeters. The active beacons mounted in the environment use a combination of radio frequency (RF) and ultrasound technologies for communication and ranging. Passive listeners attach to mobile or static devices of interest and use the network of active beacons to determine location information.

The distributed Cricket architecture has three significant advantages. First, there is no centralized controller or database to track users and devices. Second, Cricket scales well as the number of locatable devices increases because the listeners attached to those devices are passive and do not increase signal contention. Third, Cricket's decentralized architecture simplifies the system and makes it easy to deploy.

The Cricket Compass leverages the existing Cricket indoor location system by assuming there are beacons that periodically broadcast ultrasound pulses from known locations. The Compass also assumes that these pulses can be associated with the broadcasting beacons at the Compass, which itself can be localized. These assumptions are valid for a Cricket listener; therefore, I design the Cricket Compass as a Cricket listener augmented to determine orientation information in a manner that preserves the advantages of the Cricket system. In keeping with the goals of Cricket, the Compass also attempts to solve problems in ways that are realizable in hardware with physically small dimensions.

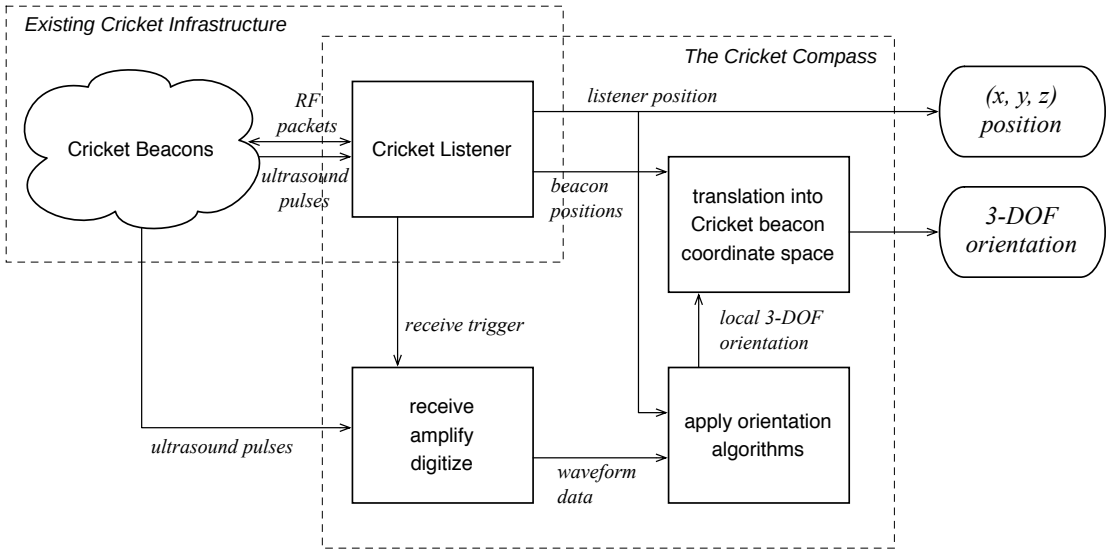


Figure 1-1: Block diagram overview of the Cricket Compass.

The orientation problem primarily reduces to accurately determining the orientation of the Compass with respect to a Cricket beacon in the Compass's own reference coordinate system. With estimates of orientation to several beacons, the end-to-end problem is then determining the orientation of the Compass in the globally consistent coordinate system maintained across all beacons and listeners.

Figure 1-1 shows the overall Cricket Compass design in a high-level block diagram. The figure distinguishes between parts of the Compass functionality that are inherited from the Cricket Listener and parts that implement the orientation capability. The figure also shows the high-level inputs and outputs of each block.

In order to accurately determine the three-dimensional orientation of the Compass with respect to a Cricket beacon, the Compass focuses on processing the ultrasound pulses emitted by the beacons. First, the Compass receives an ultrasound pulse on an array of well-placed ultrasonic receivers. The Compass amplifies and digitizes the ultrasonic waveforms output by the receivers and applies several algorithms to the waveforms to estimate its orientation to the ultrasound source. Finally, the Compass combines estimates of relative orientation to beacons with Cricket location data in order to estimate an orientation in global coordinates.

1.3 Overview

In Chapter Two, I detail my experience in approaching the orientation problem. I discuss other tracking and location systems, and present experience with Priyantha et al.'s planar orientation method for the Cricket system. In Chapter Three, I present the theory behind the operation of the Cricket Compass. I show how to localize beacons using an array of receivers, and how to determine orientation based on known beacon positions.

In Chapter Four, I describe the design and implementation of a Cricket Compass prototype, incrementally building up to Chapter Five, where I characterize the performance of the prototype and demonstrate end-to-end functionality. I then suggest areas for improvement in future implementations. I list the contributions of this thesis in Chapter Six.

In Appendix A, I present the specific hardware implementation details including circuit diagrams and sensor information. In Appendix B, I describe the setup and demonstration of the Cricket Compass and include the supporting source code.

Chapter 2

Steps Towards Orienting the Cricket Compass

The Cricket indoor location system provides pervasive computing applications with an infrastructure that addresses user privacy concerns, scales well, and is also easy to deploy and maintain. In order to extend Cricket to provide orientation data, I examine other systems that attempt to solve similar problems and review Priyantha et al.'s seminal work on orienting the Cricket Compass. I outline the specific challenges of providing three-dimensional orientation using the current Cricket location system. I then present experimental experience from attempts at solving those challenges.

2.1 Existing Orientation Systems

Virtual reality, interactive computer graphics, and mobile robotics applications have motivated the development of several systems to track the position and orientation of users and interesting objects. Welch and Foxlin [11] list the design goals of an ideal device that would satisfy the requirements of almost all position and orientation tracking applications. This device would be:

1. Small: the size of a small microchip;
2. Self-contained: have no other parts mounted in the environment or the user;
3. Complete: track all three degrees of freedom for position and all three degrees of freedom for orientation;

4. Accurate: have sub-centimeter accuracy in position and sub-degree accuracy in orientation;

5. Fast: provide high position and orientation update rates with low latency;

6. Immune to Occlusions: would not require line of sight to anything else;

7. Robust: resist performance degradation from light, sound, heat, magnetic fields, radio waves, and other environmental effects;

8. Wireless: run without wires and have an unlimited range of operation;

9. Scaleable: the above hold true no matter how many users or devices are being tracked.

Systems use a variety of technologies including electromagnetic, optical, acoustic, radio frequency, inertial, and mechanical sensing. No systems exist today that satisfy all these criteria. A few systems do quite well on a number of criteria, but because these systems are meant for virtual reality *tracking* applications, they are generally highly complex, centralized, and limited in range. These properties make them unsuitable for the types of pervasive computing applications the Cricket Compass intends to support. However, by reviewing the designs and tradeoffs made in these systems, I develop an understanding of what techniques are available and how to make the proper design choices on the Cricket Compass.

2.1.1 Constellation

The Constellation system uses a combination of accelerometers, gyroscopes, and ultrasonic sensors to estimate position and orientation [3]. Like Cricket, Constellation relies on an active set of ultrasonic beacons to determine the initial tracking position of a device. Constellation uses a Kalman filter to reject corrupted measurements and refines orientation estimates using inertial sensors. However, the precise coordination that is required between receivers and transmitters in this system makes it unsuitable for large-scale deployment.

2.1.2 HiBall

The HiBall Tracker uses synchronized infrared LEDs and precision optics to determine position with sub-millimeter accuracy with less than a millisecond latency [10]. HiBall

deploys large arrays of hundreds to thousands of LED beacons on ceiling tiles and requires a sophisticated sensor package of infrared sensors and optical lenses. Position and orientation estimates are obtained by sighting the relative angles and positions of the ceiling LEDs. Both the sensor and the LED arrays are centrally synchronized by a computer to control the LED intensity and lighting patterns required to determine position. The system requires extensive wiring, which makes it expensive and difficult to deploy. It also suffers from the need for central coordination.

2.1.3 Whisper

Whisper is an acoustic tracking system that uses a wide bandwidth signal to take advantage of low frequency sound's ability to diffract around objects [9]. Acoustic systems usually suffer from low update rates and are not very robust to environmental noise. Whisper applies spread spectrum concepts to acoustic tracking in order to overcome those problems. Whisper recursively tracks the correlation between a transmitted and received version of a pseudo-random wide-band acoustic signal. A Kalman filter is also used to reduce the computational expense of correlation calculations. The communication-intensive methods of Whisper make it hard to scale, and its use of audible frequencies make it undesirable for pervasive computing applications.

2.1.4 Commercial Systems

Commercial magnetic motion trackers have been used in virtual reality and simulation applications such as head-mounted displays and biomechanical motion capture: Ascension [1], Polhemus [6], and Northern Digital [2] all offer these motion tracking products. They provide estimates of the position and orientation of the target object by sending magnetic pulses and detecting the change of field strength along three orthogonal axes. These systems usually require a centralized coordination between the magnetic transmitters and receivers, and are limited in their range. They are susceptible to magnetic interference from the presence of metals or other conductive materials in the environment, which causes problems in many indoor environments.

2.2 Groundwork for the Cricket Compass

Having surveyed some existing orientation systems, I put forth a set of principles that should guide the design of the Cricket Compass in context to its support for pervasive computing applications. The Cricket Compass should ideally be:

Complete: able to track all three degrees of freedom for position and all three degrees of freedom for orientation;

Accurate: have sub-centimeter accuracy in position and sub-degree accuracy in orientation;

Unobtrusive: have small physical size, avoid the use of wires and visible or audible techniques;

Unrestricted: functions throughout an arbitrarily large work volume;

Robust: resist performance degradation from light, sound, heat, magnetic fields, radio waves, and other environmental effects;

Scaleable: the above hold true no matter how many users or devices are instrumented.

All six principles above apply to the approach of the Cricket location system; therefore, the Compass should supplement Cricket's capabilities while adhering to these principles.

In Section 1.2, I outlined the design space for the Cricket Compass. Recall that through the Cricket indoor location system, a set of active RF and ultrasound beacons allows me to determine the 3D position coordinates of a Cricket listener. With the Cricket system in place, Priyantha et al. [7] describe a preliminary design and implementation of the Cricket compass system consisting of a set of active Cricket beacons, passive hardware sensors, and associated software algorithms.

2.2.1 Calculating Planar Orientation

Figure 2-1 shows a beacon B, and a compass with two ultrasonic receivers, R_1 and R_2, which are located at a distance L apart from each other. The angle of rotation of the compass, θ, with respect to the beacon B, is related to the difference in distances d_1 and d_2, where d_1 and d_2 are the distances of receivers R_1 and R_2 from B. The vertical and horizontal distances from the center of the compass to B are denoted by z and x, respectively.

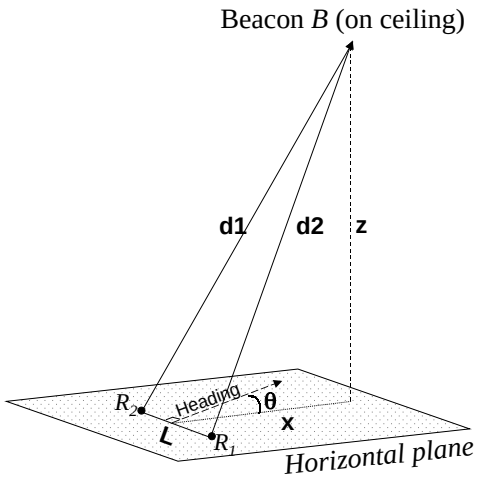


Figure 2-1: Determining the angle of orientation along the horizontal plane, θ, using distance estimates. The heading is perpendicular to the line joining the ultrasonic compass receivers, R_1 and R_2, which are placed at a distance L from each other.

Figure 2-2 shows the beacon B from Figure 2-1 projected onto the horizontal plane along which the compass is aligned. In this figure, x_1 and x_2 are the projections of distances d_1 and d_2 on to the horizontal plane. Here I must assume that I hold the compass parallel to the horizontal plane.

From Figure 2-1:

$$x_1^2 = d_1^2 - z^2 \tag{2.1}$$

$$x_2^2 = d_2^2 - z^2 \tag{2.2}$$

$$x = \sqrt{\bar{d}^2 - z^2}$$

where $\bar{d} \approx \frac{d_1+d_2}{2}$ when $d_1, d_2 \gg L$.

From Figure 2-2:

$$x_1^2 = (\frac{L}{2}\cos\theta)^2 + (x - \frac{L}{2}\sin\theta)^2$$

$$x_2^2 = (\frac{L}{2}\cos\theta)^2 + (x + \frac{L}{2}\sin\theta)^2$$

$$\Rightarrow x_2^2 - x_1^2 = 2Lx\sin\theta$$

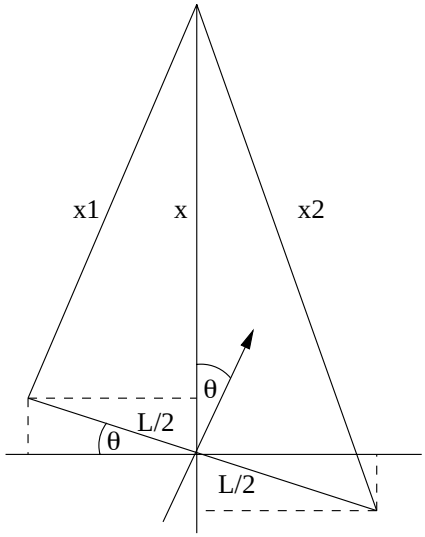


Figure 2-2: A rotated compass leads to a difference in distances between the beacon and each of the receivers. This figure is the result of projecting the beacon from Figure 2-1 onto the horizontal plane of the compass.

Substituting for x_1^2 and x_2^2 from Equations 2.1 and 2.2, I get:

$$\sin\theta = \frac{d_2 + d_1}{2Lx} \cdot (d_2 - d_1) \tag{2.3}$$

I can rewrite this as:

$$\sin\theta = \frac{d_2 - d_1}{L\sqrt{1 - (\frac{z}{\bar{d}})^2}} \tag{2.4}$$

Equation 2.4 shows that I can estimate two quantities in order to determine the orientation of the compass with respect to a beacon: (i) $(d_2 - d_1)$, the difference in distances of the two receivers from the beacon, and (ii) $z/\bar{d}$, the ratio of the beacon height over the compass plane and the distance of the beacon from the center of the compass. The location information supplied by the Cricket infrastructure allows us to compute $z/\bar{d}$ easily from the coordinates of a beacon and the coordinates of a listener. The goal then is to estimate $(d_2 - d_1)$ with high precision in order to produce an accurate estimate of θ.

2.3 Challenges in Orienting the Cricket Compass

Using Priyantha et al.'s differential distance method as a foundation to deriving orientation, I tackle several problems in order to realize the Cricket Compass. In this section, I elucidate

and characterize the primary problems in delivering accurate end-to-end results in context to the design and technology constraints of the Compass. I discuss how I attacked these problems and evaluate the effectiveness of these approaches. The experience I gained in this section forms the foundation for the solutions to the problems described in Chapter Three, where I propose the methods for delivering end-to-end Cricket Compass functionality.

2.3.1 Characterizing Cricket Ultrasound

Typical ultrasound ranging systems generate brief ultrasound pulses and use time-of-flight measurements to generate distance estimates based on the speed of sound. All versions of Cricket so far have used this simple principle to generate distance estimates for position calculation. The typical challenges and tradeoffs in using ultrasound are:

Multipath: Because walls and objects in a room reflect acoustic signals well, an ultrasound receiver can receive a signal that is the sum of a direct path signal and one or more reflected signals of longer path lengths. A great feature of pulsed ultrasound is that multipath reflections can generally be rejected by detecting the first pulse that arrives. The first pulse is guaranteed to have arrived via the direct path unless the signal is blocked. This feature comes from the fact that ultrasound travels at the speed of sound, allowing a significant time difference between direct and reflected path pulses.

Range and Directionality: A desirable property for the transmission of ultrasound is omnidirectionality, which places no restrictions on performance as transmitters and receivers vary in position and orientation. Another desirable property is long range. While transducers with smaller active surfaces help to achieve wider directionality, smaller active surfaces also translate into reduced range because the efficiency of an ultrasonic transducer is proportional to its active surface area.

Accuracy: Accurate distance measurements depend on accurate timing and accurate estimation of the speed of sound, which depends significantly on temperature, humidity, and air currents. Accurate timing depends on pulse reception. Highly resonant transducers driven by a train of cycles at the resonant frequency produce high amplitude pulses to improve range. However, the narrow bandwidth of the transducers also results in a received waveform that "rings up" gradually for several cycles, peaks,

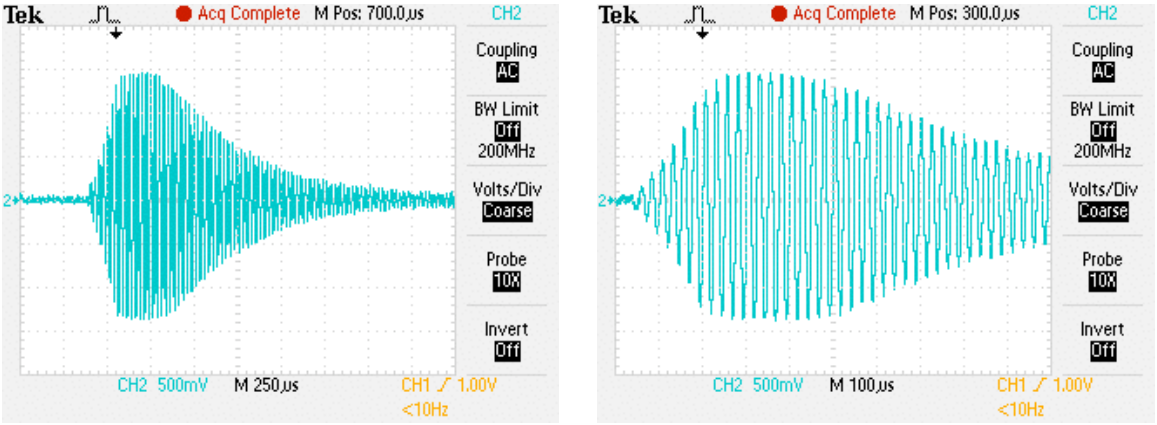


Figure 2-3: An amplified ultrasound pulse on a Cricket listener. The Cricket beacon is transmitting at a distance ≈ 1.5 m from the listener at $\approx 45°$ elevation. The received pulse rings up, peaks, and then rings down over the course of almost one hundred ultrasound periods, even though only six periods of ultrasound drive the transmitter. The horizontal scale is 500 mVolts per division. The horizontal scale is 250 μseconds per division on the left and zoomed into 100 μseconds per division on the right.

and then gradually rings down. Detection and timing of this waveform is the most important factor in achieving high accuracy.

Occlusion: Although ultrasound is more tolerant than optical methods, the acoustic nature of ultrasound still requires a general line of sight between transmitters and receivers.

Noise and Interference: Operating at the higher and inaudible ultrasound frequency range helps to reduce the ambient noise level, but jingling of keys generates high amplitude spurious ultrasound that is difficult to handle.

A fundamental step in leveraging the Cricket ultrasound for estimating orientation is to characterize Cricket's ultrasound transmit and receive behavior and to understand the underlying hardware that support ultrasound capabilities.

Cricket Ultrasound Hardware

All versions of Cricket use a standard method to generate the ultrasound pulses. The standard method is to generate a 40 kHz square wave pulse by toggling a pin on the Cricket microcontroller at the correct frequency for several cycles. The 40 kHz square wave pulse then passes through an amplification stage. The specific implementation of this amplification stage has varied across different Cricket versions, but in each version,

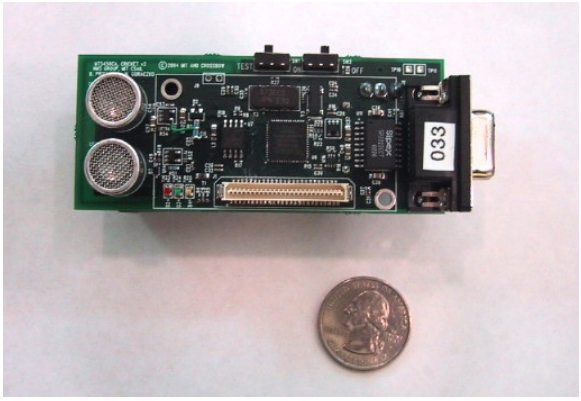


Figure 2-4: A photo of the version 2 Cricket, which can function as a listener or a beacon.

the amplification is designed to match the receive circuit in order to amplify the pulse to produce the desired range. The amplified pulse then drives the ultrasound transmitter to produce an ultrasound pulse.

All versions of Cricket use an ultrasound receiver coupled with two stages of analog gain, which affects both range and sensitivity to noise. As discussed previously, the narrow bandwidth of the transmitters and receivers results in a received waveform that "rings up" gradually for several cycles, peaks, and then gradually rings down. Figure 2-3 shows an actual oscilloscope capture of an amplified ultrasound pulse on the receive-side. From the receive-side amplified output, Cricket version-specific receive circuits use various methods to detect the presence of an ultrasound pulse.

The receive circuit on the version 1 Crickets uses a phase-lock-loop to detect the incoming ultrasound signal. Unfortunately, the phase-lock-loop takes a variable amount of time to lock on to the received pulse and limits the version 1 distance measurement accuracy to one meter. The receive circuit on the version 2 Crickets currently uses a peak detector to detect the envelope of the ultrasound pulse. The rising envelope feeds into a comparator that toggles the ultrasound receive pin on the microcontroller. The comparator is biased at a threshold that is above the ambient noise ceiling; this generally causes the ultrasound receive pin to switch high on one of the rising cycles of the received pulse. Adding the variable delay in the software layers of the Cricket, the end-to-end distance accuracy of the version 2 Crickets is on the order of several centimeters. Figure 2-4 shows a photo of the version 2 Cricket.

Cricket Ultrasound Characteristics

The unpredictable shape of the received ultrasound waveform presents a major obstacle to improving Cricket's ranging accuracy. Pulses generally decrease in ring-up rate and amplitude as distance increases. This trend, however, is difficult to discern accurately because directionality in transceivers also has a large effect on ring-up rate and amplitude—the difference in two received waveforms observed by moving a listener a fixed distance, but maintaining a fixed orientation, is oftentimes indiscernible from the difference observed by tilting or rotating a listener by a few degrees. To complicate the situation, when range is low, the high gain circuit can saturate and rail, introducing yet another variable into the received waveform characteristic. In the next section, 2.3.4, I briefly describe some work that I carried out to build up this characterization and show how the Cricket Compass circumvents the imprecise absolute ranging problem.

2.3.2 The Differential Distance Problem

The accuracy of the Compass orientation estimation depends fundamentally on the measured quantity $(d_2 - d_1)$, the differential distance. There are two main approaches to tackling this problem. The first approach is to determine d_1 and d_2 each with high accuracy; this involves obtaining precise absolute distances. The second approach is to determine the quantity $(d_2 - d_1)$ with high accuracy. I will show that obtaining precise absolute distances within the scope of the current hardware and technologies is difficult, and that obtaining precise differential distances instead is much more feasible.

Obtaining Precise Absolute Distances

Precisely measuring d_1 and d_2 separately is quite difficult. Consider, for example, a situation where $L = 5$cm, and $\theta = 10°$, with a beacon at a distance of 2 meters and a height of 1 meter from the receivers. From Equation 2.4, the value of $(d_2 - d_1)$ in this case is only ≈ 0.6cm.

Cricket's current ultrasound capability does not permit it to measure distances with sub-centimeter precision. Cricket currently uses a threshold detector on the amplified ultrasound, which results in distance accuracy on the order of several centimeters. This accuracy is insufficient to achieve precise orientation estimates. While I could increase the

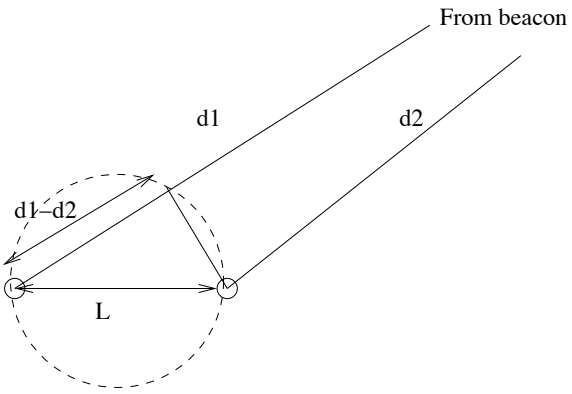


Figure 2-5: Receivers R_1 and R_2 can measure the differential distance from a far-away beacon.

value of L in order to decrease the accuracy demands on $(d_2 - d_1)$, this requires that I must space the receivers further apart. Increasing the receiver separation distance, however, runs counter to the goal of building a compass with physically small dimensions for handheld applications.

A more elaborate method than the current threshold detector is necessary to solve the absolute distance problem. Due to the ultrasound transmit and receive characteristics described in Section 2.3.1, one idea is to extrapolate the waveform start time from an equation that fits the envelope of the ultrasound wave.

To test this idea, I used exponential, first-order, and second-order equations to fit the rising peaks of the ultrasound wave using a least-squares method. Both the distance between ultrasound transmitter and receiver and the angle of reception on the receiver with respect to its vertical axis influence the shape of the ultrasound waveform envelope. However, there is no discernible predictability in those factors, and the result is that the accuracy of the waveform start time estimated by the best-fit equations, when used to calculate a distance estimate, perform worse than the current threshold detection scheme. In addition, the ultrasound waveforms are so sensitive to minute variations in those factors that this method cannot even be used to accurately determine differential distances on closely spaced receivers.

Obtaining Precise Differential Distances

Priyantha et al.'s solution to the differential distance problem uses the *phase difference* between the ultrasonic signals at two different receivers to determine differential distance

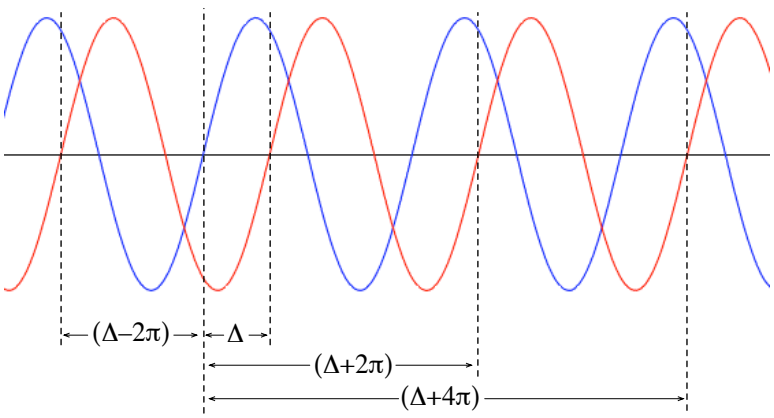


Figure 2-6: An observed phase Δ can actually correspond to an infinite number of possible real phases, all separated by 2π.

measurements with sub-centimeter precision.

Consider two ultrasonic receivers R_1 and R_2 located a distance L apart, as shown in Figure 2-5. Let d_1 and d_2 be the distances to receivers R_1 and R_2 from beacon B. Let $\delta d = d_1 - d_2$ and let W_1 and W_2 be the ultrasonic waveforms received by R_1 and R_2 from B. The phase difference between the waveforms at the two receivers, ϕ, depends on the difference in distances traversed from B to the receivers by the ultrasonic signal and the wavelength λ of the signal, and may be expressed as:

$$\phi = \frac{(\delta d)}{\lambda} \cdot 2\pi \tag{2.5}$$

ϕ denotes the *actual* phase difference between the two signals.

This approach, however, poses a phase ambiguity problem. Without knowing the start times of W_1 and W_2, I can only measure a phase difference Δ, observable from repeated low-to-high or high-to-low transitions. This observed phase difference can correspond to an infinite number of possible real phase differences, all separated by 2π. Figure 2-6 illustrates these phase ambiguities.

One way to solve this problem is to observe from Equation 2.5 that as long as $\delta d < \lambda/2$, $\phi = \Delta$, and there is no ambiguity. Since d_1, d_2, and L are three sides of a triangle, $L \geq |d_1 - d_2| = |\delta d|$, and I can therefore place the receivers at a distance $L < \lambda/2$ to unambiguously determine ϕ and therefore uniquely estimate $(d_1 - d_2)$. However, for a 40 kHz ultrasonic waveform at a temperature of 25°C and 50% humidity, $\lambda/2 = 4.35$ mm.

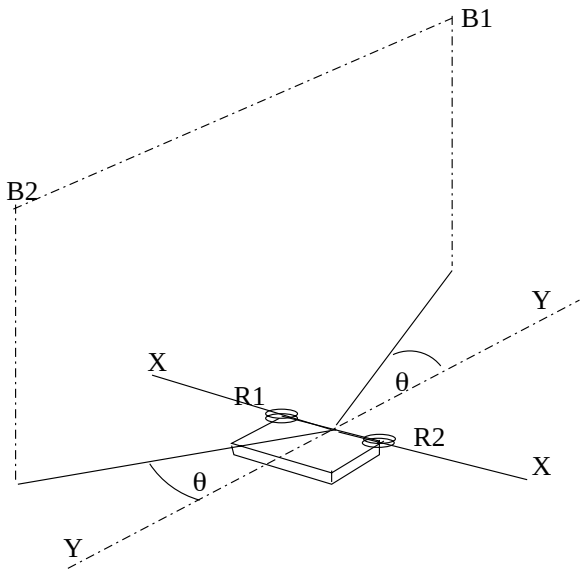


Figure 2-7: θ is ambiguous; there are two beacon positions B_1, B_2 that result in the same θ at the compass.

This is smaller than the size of most available ultrasound receivers, which are typically on the order of about 1 cm.

Priyantha et al. suggest a method of placing three receivers along a line with separation distances chosen to be relatively prime integral multiples of $\lambda/2$. Because the observed phase differences between receivers is not independent, one can disambiguate the observations to find the actual phase difference ϕ. This approach, however, requires the addition of the third receiver. In Chapter Three, I will propose an alternative method that preserves the minimal requirement of two receivers for determining ϕ.

2.3.3 Disambiguating θ

Using Equation 2.4 and the techniques discussed thus far, I can determine $\sin\theta$ between the compass and a particular beacon B. But as Figure 2-7 shows, in general, there are two positions B_1, B_2 for a beacon B that result in the same θ at the compass. This is due to symmetry of the system about the line X–X. An analytical way of understanding this is to observe that there are two values of θ in the range $[0, 2\pi)$ for a given value of $\sin\theta$. This ambiguity in the position of the beacon prevents me from determining a unique value for the heading.

Priyantha et al. solve this by using two sets of non-collinear receiver-triplets to break

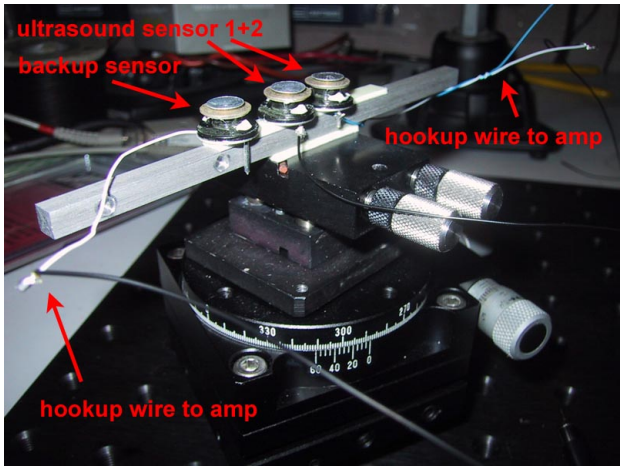


Figure 2-8: Two ultrasound receivers mounted on a precision rotating platform.

the symmetry. The two sets of receiver-triplets are perpendicularly configured using five receivers. In Chapter Three, I propose methods that allow the use of a single receiver to break the symmetry, thereby reducing the number of receivers required to three.

2.3.4 Implementation Experience

Priyantha et al. demonstrated a prototype compass device providing 2D planar orientation to within five degrees of accuracy with respect to a beacon. As a step towards realizing the Cricket Compass, I set out to improve the accuracy of planar orientation estimation using two ultrasound receivers, without dealing with the phase ambiguity or θ ambiguity problems. That is, while precisely controlling all variables, I sought to understand the limitations on obtaining highly accurate raw differential distances.

Setup

In this series of experiments, I used two beacons, two ultrasound receivers, two amplification circuits, an oscilloscope, and two pieces of precision optics mounting equipment. The optical mounting equipment consists of a heavy board and a rotating platform; the board provides a stable foundation, and the rotating platform allows precise measurements of rotation in the horizontal plane and controls for calibration. I also used a plumb line and spirit level to aid in calibration.

Figure 2-8 shows the rotating platform with the two ultrasound receivers mounted using a thin layer of double-sided sticky foam. The two receivers are placed as close to each other

as possible without touching so as to minimize the separation distance. The midpoint between the two receivers lies on the axis of rotation. The ultrasound receivers themselves only produce signals with millivolt amplitudes in response to the beacon pulses, so I need to amplify the raw signal. To provide the necessary amplification, I rebuilt two copies of the dual-stage amplification circuits that are used on Cricket listeners.

Procedure

In different trials, I explored the effect of different ultrasound receivers and waveform measurement methods on the accuracy of differential distance measurements. In these experiments I wanted to carefully control all the other variables so that I could collect accurate data representing the rotation in the horizontal plane. In each trial, I started with this calibration procedure:

1. Select a good open space to test such that multipath signals do not contribute to measurements.

2. Place a beacon directly above the rotary table. Use a plumb line hanging down from the beacon's ultrasound transmitter to locate the midpoint of the ultrasound receivers directly below.

3. Using a spirit level, level the rotating platform along two orthogonal axes to ensure that the rotation occurs purely in the horizontal plane.

4. Turn the beacon on, and level the two receivers such that when rotated, the two received waveforms are exactly in-phase.

After this calibration procedure, the receivers were completely level and rotated only in the horizontal plane. I then set up a beacon about three feet above the sensors and about four feet away horizontally. I turned on the beacon and rotated the receivers until both signals were in-phase. This was the baseline 0° heading. I then rotated the receivers in 5° increments from $-90°$ to $+90°$, recording three phase measurements from three separate pulses at each increment. Rotating in this way implicitly avoids the θ ambiguity. In addition, I disambiguated the phase measurements by using apriori knowledge of the heading.

I carried out several trials of this rotation experiment using three different ultrasound receivers manufactured by Panasonic, Murata and Kobetone. The Kobetone receivers are

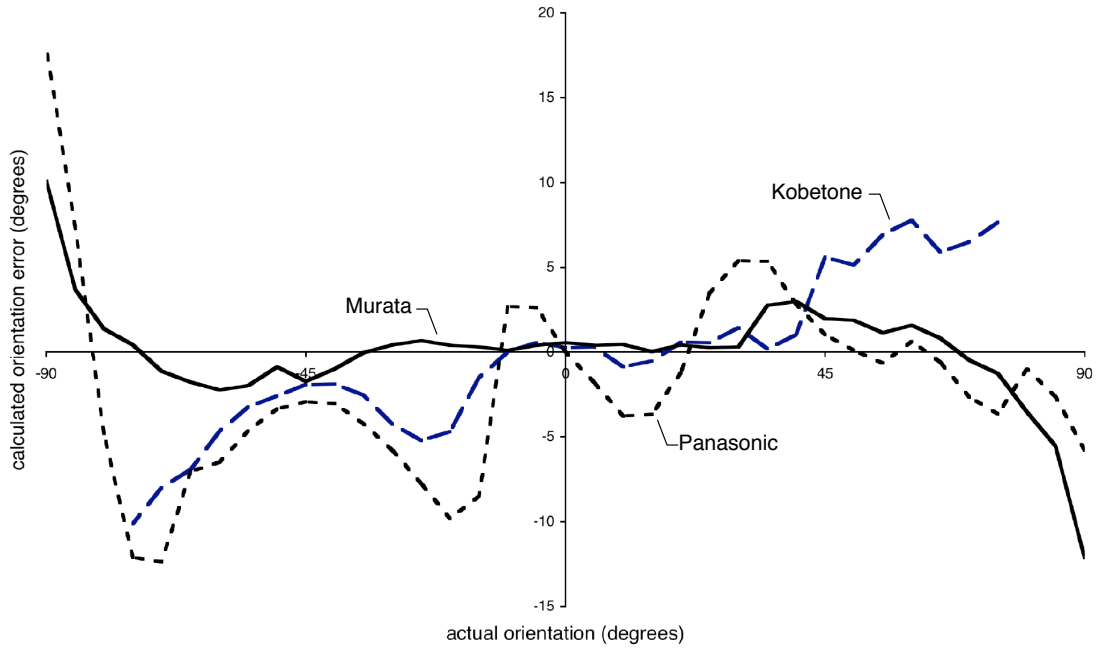


Figure 2-9: A plot of error in calculated orientation while taking measurements using three different ultrasound receiver types. The Murata sensors provide the most accurate results.

the standard ones used on Crickets. For each set of rotation and phase data, I measured the beacon height and distance and used Equation 2.4 to convert the phase measurements into estimated orientation. I plot the best set of data for each of the three sensors against each other in Figure 2-9.

In the rotation experiments, I measured phase differences by looking at repeated low-high zero-crossing points. However, within any pulse there can be well over twenty periods. I made most measurements in the highest amplitude parts of the pulse, but this method is still somewhat arbitrary. Therefore, after carrying out the rotation experiments, I went back to analyze the captured ultrasound waveforms to check for consistency in phase difference throughout the pulse duration.

Results

There were three main conclusions from this preliminary experience:

Choice of Ultrasound Receiver: The best sets of data in Figure 2-9 are representative of the general receiver performance. The Murata sensors produced the best results. The Murata sensors were of higher manufacture quality and were also physically the

smallest.

Accurate Range: Regardless of which ultrasound receiver I used, the experimental results are always most accurate from $-45°$ to $+45°$. Intuitively, as I rotate the receivers past $45°$, the differential distance between the receivers with respect to the beacon changes much less than when rotating at smaller angles. Based on Equation 2.4, we can analytically look at the arcsine function and observe that for $-45° \leq \arcsin(x) \leq 45°$ the function is quite linear, but that outside this range, the slope begins to increase. My precision in phase measurement is bounded, corresponding to a bounded precision on x. For a bounded error in x, the error in $|\arcsin(x)| > 45°$ is greater than error in $|\arcsin(x)| < 45°$.

Measurement Method: Measuring phase by looking at repeated zero-crossings introduces error because the consistency of zero-crossings between two pulses vary from noise and analog-to-digital conversion quantization error. I address this issue in Chapter Three by proposing an accurate measurement method that captures more than just the information contained in one period of the waveforms.

Chapter 3

Theory of Operation

In this chapter, I propose an end-to-end solution to determine three-dimensional orientation. I build upon the Cricket indoor location system and the initial compass work described in Chapter Two to propose a set of methods enabling the Compass orientation capability described in Section 1.2. I first describe the specific capabilities provided by the Cricket infrastructure. Then, I break down the orientation problem into two parts: (i) determining orientation to beacons in the Compass local coordinates and (ii) using multiple estimates of orientation to beacons to determine the unique Compass orientation consistent with the deployed Cricket infrastructure. While proposing methods to solve these problems, I also propose a solution to the lower-level differential distance problem.

3.1 Cricket Infrastructure

The current revision of the Cricket system requires the placement of four active beacons to enable delivery of location information. A listener deployed in this infrastructure specifically provides the capability to localize the (x, y, z) coordinates of itself and all the beacons and to associate each received ultrasound pulse with the particular beacon sending that pulse. Recall that in this version of the Cricket system, distance measurements are only accurate to within several centimeters.

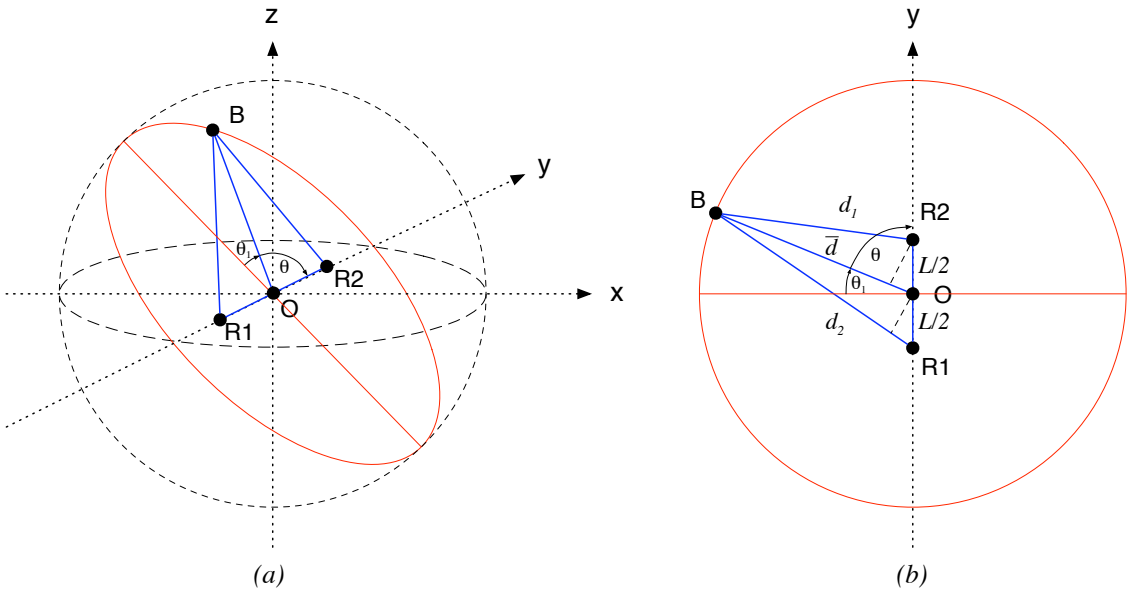


Figure 3-1: Determining the angle θ between the vector to the beacon and the axis formed by the receiver pair using distance estimates. (a) shows a 3D view and (b) shows the 2D view of the plane formed by R1, R2 and B.

3.2 Three-Dimensional Local Orientation

Figure 3-1a shows a beacon B and two ultrasonic receivers, R_1 and R_2, which are separated by a distance L. First, let me define a coordinate system where the origin O is the mid-point of the segment from R_1 to R_2. Let the vector from the origin to the beacon be $\overrightarrow{OB}$. For convenience, let us define the y-axis to run through R_1 and R_2. The basic building block of solving the three-dimensional orientation problem is the ability to determine the angle θ between the two vectors $\overrightarrow{OB}$ and $\hat{y}$.

From Figure 3-1b:

$$d_1^2 = \left(\frac{L}{2}\cos\theta_1\right)^2 + \left(\bar{d} - \frac{L}{2}\sin\theta_1\right)^2$$

$$d_2^2 = \left(\frac{L}{2}\cos\theta_1\right)^2 + \left(\bar{d} + \frac{L}{2}\sin\theta_1\right)^2$$

$$\Rightarrow d_2^2 - d_1^2 = 2\bar{d}L\sin\theta_1 \tag{3.1}$$

where $\bar{d} \approx \frac{d_1+d_2}{2}$ when $d_1, d_2 \gg L$.

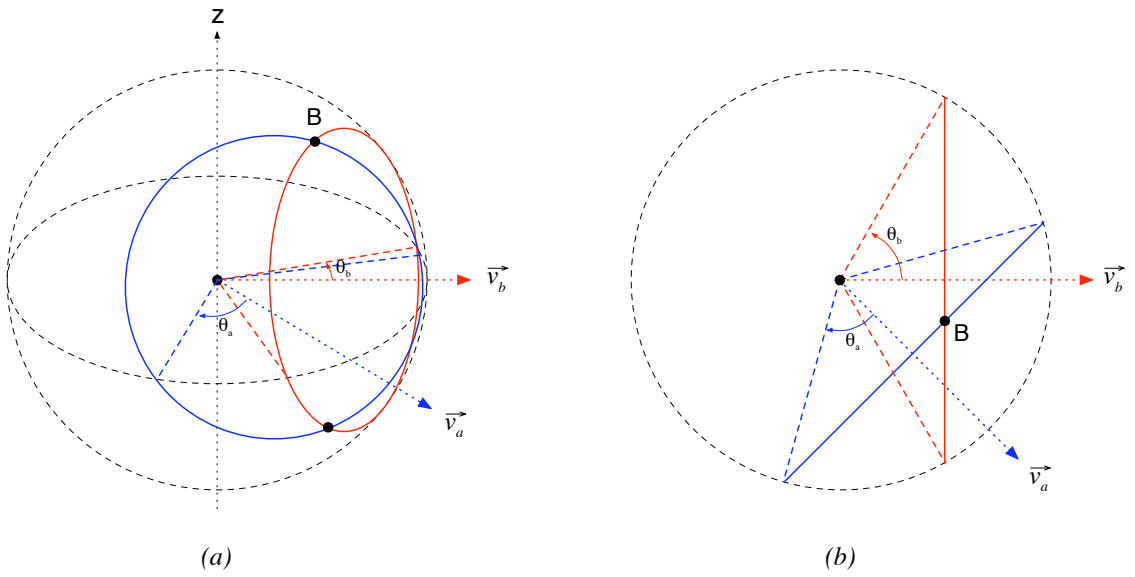


Figure 3-2: A sphere bounds the possible beacon positions based on a distance estimate. $\overrightarrow{v_a}$ and $\overrightarrow{v_b}$ represent axes through receiver pairs. The "opening angles" θ_a and θ_b relative to the axes determine two cones that intersect with the sphere to form two circles. The intersection of these circles are the two possible beacon positions. (a) shows a 3D view and (b) shows the view from above.

Equation 3.1 reduces as follows:

$$\sin\theta_1 = \left(\frac{d_2 - d_1}{L}\right)\left(\frac{d_2 + d_1}{2\bar{d}}\right) \quad \Rightarrow \quad \sin\theta_1 = \frac{d_2 - d_1}{L}$$

From Figure 3-1b I can then relate θ_1 and θ because $\theta = \pi/2 - \theta_1$:

$$\theta = \frac{\pi}{2} - \arcsin\left(\frac{d_2 - d_1}{L}\right) \tag{3.2}$$

Equation 3.2 now allows me to determine the angle θ between the two vectors $\overrightarrow{OB}$ and $\hat{y}$ by measuring the differential distance $(d_2 - d_1)$ between the two ultrasound receivers. Recall that $\hat{y}$ is really just the vector that runs through the two receivers. Therefore, I can find θ between the vector to a beacon and any vector running through a pair of receivers.

Given a pair of receivers and θ, I know that the position of the beacon must be on the surface of a cone. The cone shape arises because the vector through the receivers is an axis of symmetry and rotation about that axis produces the cone. The point of this cone is the midpoint of the two receivers, and the "opening angle" of this cone is θ.

Now, if I use a distance estimate provided by a Cricket listener, this further limits the

position of the beacon to one circle on that cone. In order to further constrain the position of the beacon, I introduce another pair of receivers. Using Equation 3.2 and a beacon distance estimate, I can again use this pair of receivers to locate the beacon position on a circle. As long as this pair of receivers is not collinear, the two receiver pairs will allow us to localize the beacon position to two points, the intersection of the two circles. These two circles lie on the surface of the sphere that has its center at the receivers and a radius equal to the beacon distance estimate.

Figure 3-2 illustrates the idea of using two receiver pairs. Observe that the beacon position is ambiguous because it can be located either above or below the plane formed by the receivers. Also note that the two receiver pairs in the figure are formed by only three receivers. If I use four receivers, I then have more than two pairs of receivers. And, if the fourth receiver is not coplanar with the other three, I can then unambiguously locate the beacon at a single point.

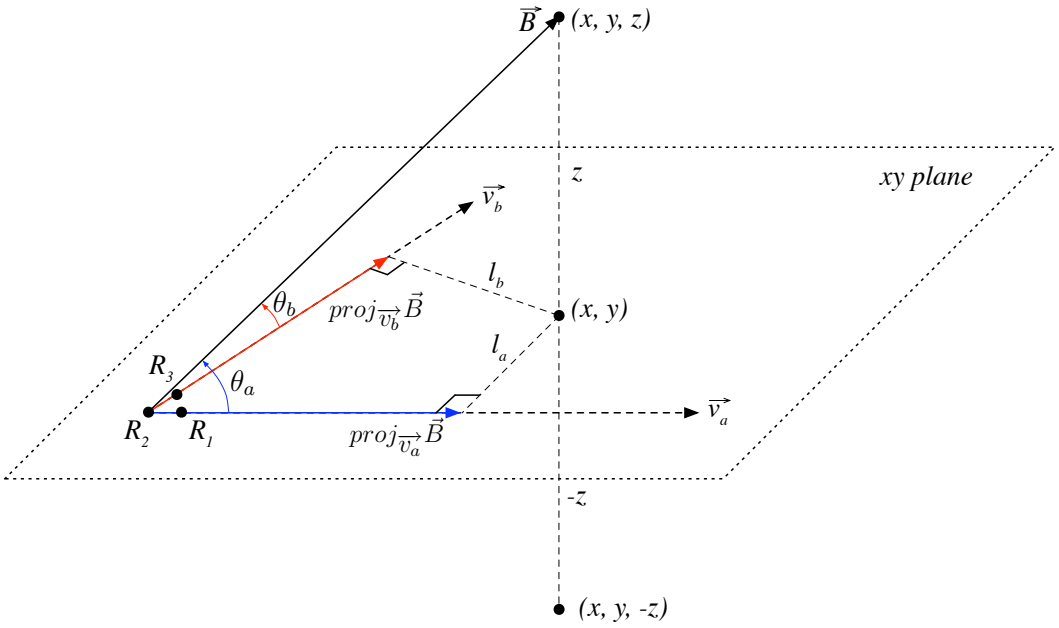


Figure 3-3: Determining the position of a beacon using an array of receivers. Vector methods pinpoint (x, y) coordinates. A $\pm z$ ambiguity arises because the receivers are coplanar.

Returning back to the case of three receivers, I can calculate the (x, y, z) coordinates of the beacon as shown in Figure 3-3. Let $\overrightarrow{v_a}$ be a vector starting from R_2 and passing through R_1. Let $\overrightarrow{v_b}$ be the vector starting from R_2 and passing through R_3. Let $\overrightarrow{v_a}$ and $\overrightarrow{v_b}$ together define the xy plane containing the three receivers. Let $\vec{B}$ be the vector from R_2 to

the beacon. When $\|\vec{B}\| \gg$ the separation distance between the receivers, we can calculate as follows:

1. Calculate θ_a and θ_b using Equation 3.2. θ_a is the angle between $\overrightarrow{v_a}$ and $\vec{B}$. θ_b is the angle between $\overrightarrow{v_b}$ and $\vec{B}$:

$$\theta_a = \frac{\pi}{2} - \arcsin\left(\frac{d_2 - d_1}{L}\right)_a \qquad \theta_b = \frac{\pi}{2} - \arcsin\left(\frac{d_2 - d_1}{L}\right)_b$$

2. Project $\vec{B}$ onto $\overrightarrow{v_a}$, and project $\vec{B}$ onto $\overrightarrow{v_b}$:

$$proj_{\overrightarrow{v_a}}\vec{B} = \|\vec{B}\|\cos(\theta_a)\frac{\overrightarrow{v_a}}{\|\overrightarrow{v_a}\|} \qquad proj_{\overrightarrow{v_b}}\vec{B} = \|\vec{B}\|\cos(\theta_b)\frac{\overrightarrow{v_b}}{\|\overrightarrow{v_b}\|}$$

3. Define the line l_a such that $l_a \perp proj_{\overrightarrow{v_a}}\vec{B}$ and passes through the endpoint of $proj_{\overrightarrow{v_a}}\vec{B}$. Similarly define the line l_b such that $l_b \perp proj_{\overrightarrow{v_b}}\vec{B}$ and passes through the endpoint of $proj_{\overrightarrow{v_b}}\vec{B}$. The intersection of l_a and l_b determine the x and y coordinates of the beacon position.

4. Calculate the two possible beacon z coordinates:

$$z = \pm\sqrt{\|\vec{B}\|^2 - x^2 - y^2} \tag{3.3}$$

I now have the $(x, y, \pm z)$ coordinates of a beacon in the local coordinate space defined by the receivers. The position of receiver R_2 determines the exact origin of this coordinate space. The coordinates I assign to R_1 and R_3 implicitly define the axes of this space because they determine the vectors $\overrightarrow{v_a}$ and $\overrightarrow{v_b}$.

Note that I can combine steps 1 and 2 of the above beacon position calculation to eliminate any trigonometric functions. Taking the receivers associated with $\overrightarrow{v_a}$ as an example, I have:

$$\theta_a = \frac{\pi}{2} - \arcsin\left(\frac{d_2 - d_1}{L}\right)_a \quad \text{and} \quad proj_{\overrightarrow{v_a}}\vec{B} = \|\vec{B}\|\cos(\theta_a)\frac{\overrightarrow{v_a}}{\|\overrightarrow{v_a}\|}$$

Combining the two equations:

$$proj_{\overrightarrow{v_a}}\vec{B} \;=\; \|\vec{B}\|\cos\left(\frac{\pi}{2} - \arcsin\left(\frac{d_2 - d_1}{L}\right)_a\right)\left(\frac{\overrightarrow{v_a}}{\|\overrightarrow{v_a}\|}\right)$$

$$
\begin{aligned}
&= \|\vec{B}\| \sin\left(\arcsin\left(\frac{d_2 - d_1}{L}\right)_a\right)\left(\frac{\overrightarrow{v_a}}{\|\overrightarrow{v_a}\|}\right) \\
&= \|\vec{B}\| \left(\frac{d_2 - d_1}{L}\right)_a \left(\frac{\overrightarrow{v_a}}{\|\overrightarrow{v_a}\|}\right) \qquad (3.4)
\end{aligned}
$$

3.3 Obtaining Accurate Differential Distances

In Section 2.3.2 I discussed the problem of obtaining accurate differential distances. This problem applies to the beacon position calculation in the Section 3.2 because I want to precisely measure $(d_2 - d_1)$, which appears in Equation 3.4.

As discussed in Section 2.3.2, I can measure $(d_2 - d_1)$ by looking at the phase differential between two received ultrasound waveforms. As shown in figure 2-6 there is a phase ambiguity problem when looking at phase differentials. Instead of looking at repeated low-high or high-low crossings, however, I can measure the differential distance by looking at the normalized cross-correlation between the waveforms received at R_1 and R_2 for a varying delay k. I express this as:

$$
n[k] = \frac{\sum_i s_1[i]s_2[i+k]}{\sqrt{(\sum_i s_1^2[i])(\sum_i s_2^2[i+k])}} \qquad (3.5)
$$

$s_1[i]$ and $s_2[i]$ are sample values of the ultrasound waveforms received on R_1 and R_2 respectively. $n[k]$ reaches a maximum when s_2 is precisely k_{max} samples ahead of s_1, so I can use k_{max} as an accurate estimate of the phase differential between the two waveforms. Using the waveform sample rate and the speed of sound, I can convert k_{max} into $(d_2 - d_1)$:

$$
\begin{aligned}
(d_2 - d_1) &= \left(\frac{\text{speed of sound}}{\text{sample rate}}\right) k_{max} \\
&= \left(\frac{\text{speed of sound}}{\text{sample rate}}\right) (\arg\max_k n[k]) \qquad (3.6)
\end{aligned}
$$

3.4 Registration to Global Orientation

In Section 3.2, I proposed how to determine the Compass orientation with respect to a beacon. This orientation, however, is based on a set of coordinate axes defined by the array of receivers on the Compass. In order to integrate the Compass orientation into the Cricket system, what I require is the end-to-end orientation of the Compass in the Cricket coordinate

system defined by the network of beacons. In particular, I wish to find the 3-DOF rotation that optimally registers the Compass coordinate space with Cricket coordinate space. This registration yields the absolute orientation of the Compass in the globally consistent Cricket coordinate space.

From the existing Cricket infrastructure, I can determine the (x, y, z) positions of the Compass and beacons in Cricket coordinates. Using the methods proposed in Section 3.2, I can also determine the positions of the beacons in a coordinate space where the Compass defines the origin and coordinate axes. I wish to find the relationship between the two coordinate systems using corresponding position estimates of the beacons and the Compass in the two coordinate systems. Berthold Horn has presented an elegant closed-form least-squares solution to this problem using unit quaternions [5].

If I take the case where I have two beacons and one Compass, I then have three points in the Compass and Cricket coordinate systems. Horn's general solution for 6-DOF absolute orientation simplifies greatly when both sets of measurements are exactly coplanar, as always happens when there are only three measurements. Horn breaks the 3-DOF rotation problem into two parts: (i) rotating the planes formed by each set of measurements into coincidence and then (ii) rotating about the normal of the plane so as to minimize the sum of squares of distances between corresponding measurements. The overall rotation is the combination of these two rotations.

Horn's method also includes the further steps for finding the optimal 3-DOF translation, but for the Compass I am only interested in pure 3-DOF rotation. Here I will apply Horn's method to a set of positions in Cricket space and a set of positions in Compass space. Each set includes the (x, y, z) position of two beacons B_1 and B_2 and the Compass C.

3.4.1 Centroid Relative Coordinates

Let the positions in the *compass* and *cricket* coordinate systems be:

$$\{\mathbf{r}_{compass,i}\} \quad \text{and} \quad \{\mathbf{r}_{cricket,i}\}$$

where $i = \{B_1, B_2, C\}$ and $\mathbf{r}$ is the (x, y, z) vector measurement of a beacon or the Compass. Start by referring all measurements to the centroids defined by:

$$\bar{\mathbf{r}}_{compass} = \frac{1}{3}\sum_i \mathbf{r}_{compass,i} \quad \text{and} \quad \bar{\mathbf{r}}_{cricket} = \frac{1}{3}\sum_i \mathbf{r}_{cricket,i}$$

Denote the new centroid-relative measurements by:

$$\mathbf{r}'_{compass,i} = \mathbf{r}_{compass,i} - \bar{\mathbf{r}}_{compass} \quad \text{and} \quad \mathbf{r}'_{cricket,i} = \mathbf{r}_{cricket,i} - \bar{\mathbf{r}}_{cricket}$$

3.4.2 Rotation of the Planes

There are two beacon positions B_1 and B_2 and the Compass position C forming a plane in each coordinate system. Start by finding the normals to the two planes using cross products:

$$\mathbf{n}_{compass} = \mathbf{r}'_{compass,B_2} \times \mathbf{r}'_{compass,B_1}, \qquad \mathbf{n}_{cricket} = \mathbf{r}'_{cricket,B_2} \times \mathbf{r}'_{cricket,B_1}$$

Also find the unit normals:

$$\hat{n}_{compass} = \frac{\mathbf{n}_{compass}}{\|\mathbf{n}_{compass}\|}, \qquad \hat{n}_{cricket} = \frac{\mathbf{n}_{cricket}}{\|\mathbf{n}_{cricket}\|}$$

The line of intersection of the two planes lies in both planes, so it is perpendicular to both normals and therefore, parallel to the cross product of the two normals:

$$\mathbf{a} = \mathbf{n}_{compass} \times \mathbf{n}_{cricket}, \qquad \hat{a} = \frac{\mathbf{a}}{\|\mathbf{a}\|}$$

The angle ϕ between the normals is the angle I must rotate:

$$\cos\phi = \hat{n}_{compass} \cdot \hat{n}_{cricket}, \qquad \sin\phi = \|\hat{n}_{compass} \times \hat{n}_{cricket}\|$$

To rotate the Compass measurements into the plane containing the Cricket measurements, use Rodrigues' formula [4], the unit quaternion:

$$\mathbf{q}_a = \left(\cos\frac{\phi}{2},\ \hat{a}\sin\frac{\phi}{2}\right)$$

Let $\mathbf{r}''_{compass,i}$ be the rotated version of $\mathbf{r}'_{compass,i}$ by:

$$\mathbf{R''}_{compass,i} = \mathbf{q}_a * \mathbf{R'}_{compass,i} * \mathbf{q}_a^{-1}$$

where $*$ denotes quaternion multiplication and $\mathbf{R}$ is the quaternion representation of a point $\mathbf{r}$ given by $\mathbf{R} = (0, \mathbf{r})$.

3.4.3 Rotation in the Plane

Now find the rotation in the plane of the Cricket measurements that minimizes the sum of squares of distances between corresponding measurements. Let α_i be the angle between corresponding measurements. Let $r'_{cricket,i} = \|\mathbf{r'}_{cricket,i}\|$ and $r''_{compass,i} = \|\mathbf{r''}_{compass,i}\| = \|\mathbf{r'}_{compass,i}\|$. Figure 3-4 illustrates how to apply the cosine rule for triangles to find the square of the distance between corresponding measurements:

$$d_i^2 = \left(r'_{cricket,i}\right)^2 + \left(r''_{compass,i}\right)^2 - 2\left(r'_{cricket,i}\right)\left(r''_{compass,i}\right)\cos\alpha_i$$

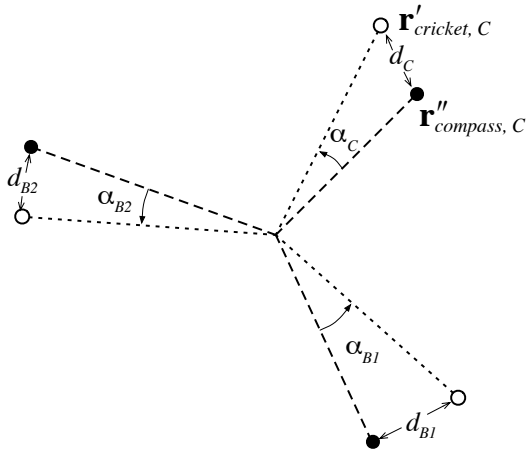


Figure 3-4: The second rotation in the plane minimizes the sum of squares of distances between corresponding points.

When I rotate the Compass measurements in the plane through an angle θ, the angles α_i are reduced by θ. So, to minimize the sum of squares of distances I need to maximize:

$$\sum_i \left(r'_{cricket,i}\right)\left(r''_{compass,i}\right)\cos\left(\alpha_i - \theta\right)$$

Because $\cos(\alpha+\beta) = \cos\alpha\cos\beta - \sin\alpha\sin\beta$, I can equivalently maximize:

$$C\cos\theta + S\sin\theta,$$

where

$$C = \sum_i \left(r'_{cricket,i}\right)\left(r''_{compass,i}\right)\cos\alpha_i = \sum_i \left(\mathbf{r}'_{cricket,i}\cdot\mathbf{r}''_{compass,i}\right)$$

$$S = \sum_i \left(r'_{cricket,i}\right)\left(r''_{compass,i}\right)\sin\alpha_i = \left(\sum_i \mathbf{r}'_{cricket,i}\times\mathbf{r}''_{compass,i}\right)\cdot\hat{n}_{cricket}$$

This expression has extrema where $C\sin\theta = S\cos\theta$. Using the identity $\sin^2\theta + \cos^2\theta = 1$, I have:

$$\sin\theta = \pm\frac{S}{\sqrt{S^2+C^2}}\,, \qquad \cos\theta = \pm\frac{C}{\sqrt{S^2+C^2}}$$

The extreme values of the expression are $\pm\sqrt{S^2+C^2}$. For a maximum, choose the values:

$$\sin\theta = +\frac{S}{\sqrt{S^2+C^2}}\,, \qquad \cos\theta = +\frac{C}{\sqrt{S^2+C^2}}$$

Note that S and C can be negative. The second rotation is about the axis $\hat{n}_{cricket}$ by an angle θ. Using Rodrigues' formula again, I can represent this by the unit quaternion:

$$\mathbf{q}_p = \left(\cos\frac{\theta}{2},\ \hat{n}_{cricket}\sin\frac{\theta}{2}\right)$$

Now, the overall rotation is the composition of the two rotations: $\mathbf{q} = \mathbf{q}_p * \mathbf{q}_a$, which is the quaternion representing the rotation of the Cricket Compass into the globally consistent Cricket coordinate space. The inverse of $\mathbf{q}$ is the end-to-end orientation of the Compass in the Cricket coordinate space.

Note that I can avoid using any trigonometric functions by using the half-angle formulas:

$$\cos\frac{\theta}{2} = \sqrt{\frac{1+\cos\theta}{2}}\,, \qquad \sin\frac{\theta}{2} = \frac{\sin\theta}{\sqrt{2(1+\cos\theta)}}$$

for $-\pi \le \theta \le \pi$.

Chapter 4

The Cricket Compass Prototype

In the previous chapter, I proposed methods to determine the end-to-end orientation of the Cricket Compass operating within the Cricket location infrastructure. In this chapter I implement a prototype of the Cricket Compass; the design of this implementation is summarized in figure 4-1. I discuss several practical challenges that arise when designing and implementing the Cricket Compass. I characterize the performance of this prototype at each layer of implementation and in end-to-end performance.

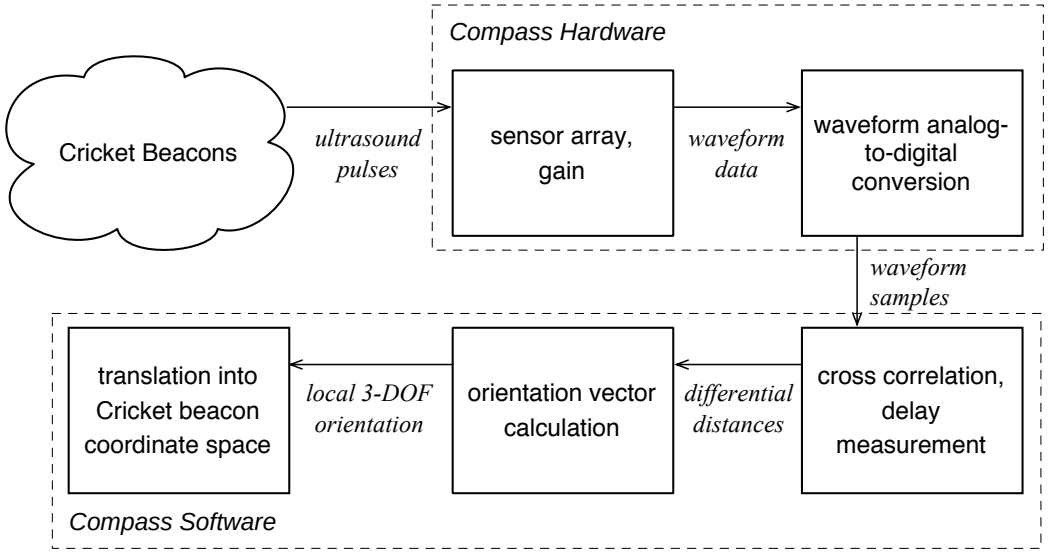


Figure 4-1: A block diagram of the Cricket Compass implementation. Ultrasound receive, amplification and digitization are accomplished in hardware while the orientation algorithms are implemented in software.

4.1 Hardware Design Parameters

The Cricket Compass hardware handles low-level reception, amplification and digitization of ultrasound pulses. The overall performance and capabilities of the Compass depend on several parameters including the choice of sensors, sensor array geometry, and waveform sampling rate. I accumulated experience through work described in Section 2.3.4, and I complement this experience with additional analysis here to guide my implementation and design choices.

4.1.1 Sensor Array

I start at the hardware level by building the sensor array and amplification circuity to produce the analog ultrasound waveforms. In Section 2.3.4, I performed preliminary experiments with three types of ultrasound sensors and a two-stage amplification circuit similar to the one in the current revision of Cricket hardware.

For the Cricket Compass implementation, I use the Murata ultrasound sensors, which performed the best in previous experiments. I also use the same well-tested and proven amplification circuit; for the circuit design refer to Appendix A. The response of the ultrasound sensors to Cricket ultrasound pulses is only a few millivolts, so this is an extremely high gain circuit that is prone to oscillation. Because I build multiple channels of this high gain circuit operating from the same supply and ground, I exercise extreme care to minimize noise to a level not exceeding 200mV peak-to-peak on the output. This is on par with the performance of current Cricket hardware.

Sensor Array Geometry

Having chosen a sensor and gain circuit, the first design parameter for the sensor array is how many sensors to use. The orientation method proposed in Section 3.2 requires at least two sensor pairs, or three non-collinear receivers. In addition, a practical issue is that a receiver pair is only accurate to within angles of $|\theta| < 45°$ as discussed in Section 2.3.4. I require that the Compass have the capability of accurately determining orientation in all directions. Therefore, the choice of sensor array geometry is loosely coupled with the choice of number of sensors. That is, for a given sensor array geometry, for each possible Compass orientation there must be at least two sensor pairs capable of generating accurate

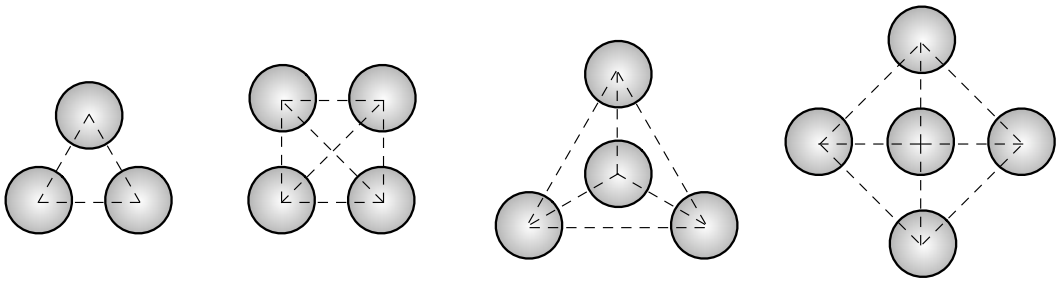


Figure 4-2: A few possible ultrasound sensor array geometries. The dotted lines denote sensor pairs.

measurements.

Before I discuss specific sensor array geometries, I wish to briefly mention that the methods proposed in Section 3.2 allow for the use of sensors in arbitrary configurations. Although this makes it possible to implement three-dimensional sensor arrays, I restrict my design choice to planar configurations of sensors. All the ultrasound sensors I experimented with are directional in nature, and all have a top and bottom. The ultrasound sensors detect almost no signal from sources that are located below the sensor; this makes the implementation of an omnidirectional three-dimensional array quite difficult.

In this Compass implementation, I adhere to the usual notion of beacons mounted on ceilings with listening devices generally facing up while in use. Therefore, for simplicity I restrict the space of possible array geometries to planar configurations. Given that the sensor array is planar and should be omnidirectional, the possible array geometries should look symmetric. For example, some combination of equilateral triangles, squares, and crosses could be possible candidates. Figure 4-2 illustrates a few possible geometries.

In selecting a configuration to implement, there are a few guiding principles. First, as discussed before, there must be at least two sensor pairs capable of generating measurements in order to calculate a beacon position. Remember also that a given sensor pair is only accurate for opening angles up to 45°. Finally, most oscilloscopes and multi-channel analog-to-digital converters offer only four signal channels.

Figure 4-3 shows four configurations of four sensors or less and illustrates the accurate sensor "coverage" of each configuration. (a) illustrates that one sensor pair oriented on the 0° axis alone provides accurate coverage for the range $[45°, 135°]$ and $[225°, 315°]$. (b) shows that with three sensors, there are ranges without accurate coverage by two pairs. (c) is a square configuration of four sensors. Although there are six pairs of sensors, two of the pairs

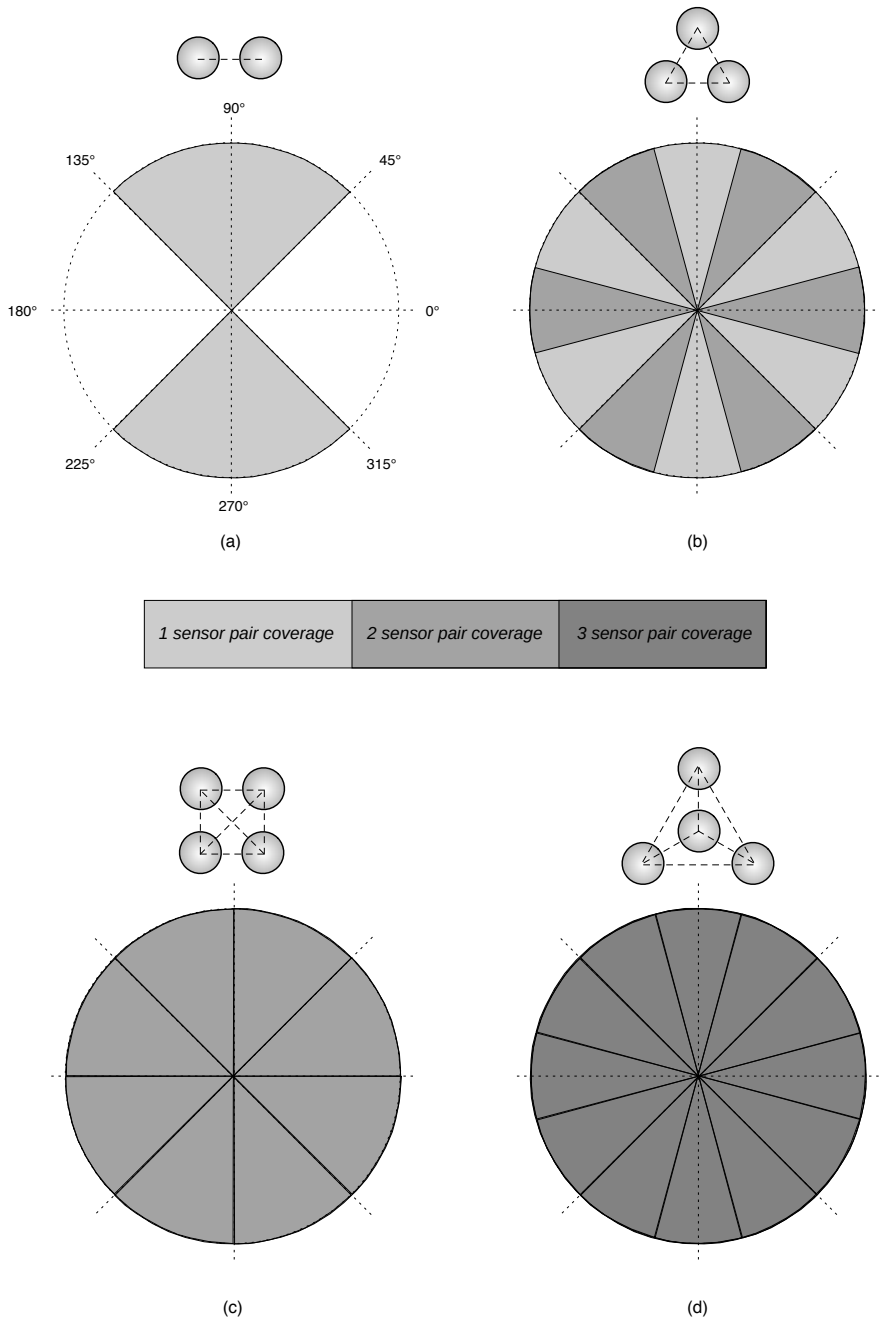


Figure 4-3: A comparison of sensor pair coverage for different geometries.

are not unique in their orientation and do not provide more information. This geometry exhibits accurate coverage by two sensor pairs in every direction. (d) represents the four sensor equilateral triangle. I select this geometry because it exhibits accurate coverage by three sensor pairs in every direction. In each of the eight 30° sectors, I can use three of the six total sensor pairs to calculate position.

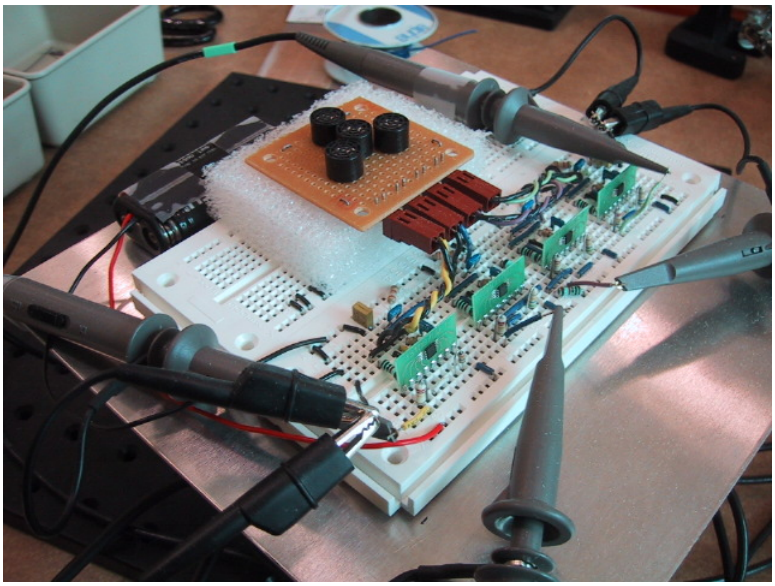


Figure 4-4: A photo of the actual compass hardware prototype. The four ultrasound sensors are configured in the equilateral triangle configuration. Oscilloscope probes connect to the output of four channels of high-gain battery-powered amplifiers. A sheet of metal provides a good ground plane.

Separation Distance

Having chosen a sensor array geometry, I can choose how far apart to set the sensors. For several reasons, I choose to place the sensors as close to each other as possible without touching. First, from experience a smaller separation distance increases accuracy. Second, a small separation physically constrains the search space for the cross correlation delay measurement; the waveforms cannot have a differential distance greater than the physical distance between receivers. Finally, a minimal separation distance creates a physically small sensor package.

4.1.2 Analog-to-Digital Conversion

I digitize the ultrasound waveforms in order to process them in software. The analog-to-digital conversion resolution and sample rate can both affect the differential distance measurement performed by the normalized cross correlation. The conversion sample rate limits the precision of the measurement because the units of the inter-waveform time delay are a discrete number of samples. The conversion resolution, however, has not shown any effect on the measurement. Most analog-to-digital converters provide at least eight bits of resolution, which is sufficient for the Compass application.

For the Cricket Compass, I use the conversion capabilities of a Tektronix TDS2024 digitizing oscilloscope. This oscilloscope provides serial connectivity to a PC, four channels of input, and sample rates as high as 2 gigasamples per second at eight-bit resolution. The oscilloscope is limited to 2500 samples per captured waveform.

4.2 Software Design Parameters

I program the cross correlation, relative beacon position calculation, and coordinate translation methods described in Chapter Three, implementing these methods in several hundred lines of MATLAB code. I also automate the acquisition of waveform data from the oscilloscope through the MATLAB serial programming interface. These implementations provide functions to acquire the ultrasound waveform data in the MATLAB environment, measure the inter-sensor differential distances, generate relative beacon position estimates, and to visualize the calculation methods.

After measuring the six inter-sensor differential distances, but before estimating a beacon position, I must choose the three differential distances that can support accurate calculations. Figure 4-3d shows twelve 30° sectors; in each sector, three sensor pairs are accurate. I bootstrap from the six differential distances by looking at the sign of the differential distances. By taking the sign of a distance corresponding to a sensor pair, I know which sensor of that pair received the ultrasound pulse first. I can use this information to localize a beacon to a 180° sector. By doing this for each of the six pairs, I can then localize a beacon to a single 30° sector and decide which pairs to use for calculation.

When generating beacon position estimates, I always take the $(x, y, +z)$ coordinate because the ultrasound array cannot receive ultrasound from below the xy plane and because in general, beacons will be positioned above on walls or the ceiling. After the compass functions have been applied to estimate the position of two beacons, I then apply the translation method to determine absolute orientation.

4.3 Testing and Modifications

The primary challenge for the Compass is obtaining accurate and precise inter-sensor differential distance measurements. The accuracy of the higher-level calculations depends heavily on these measurements. In preliminary testing of the hardware and software, a number of

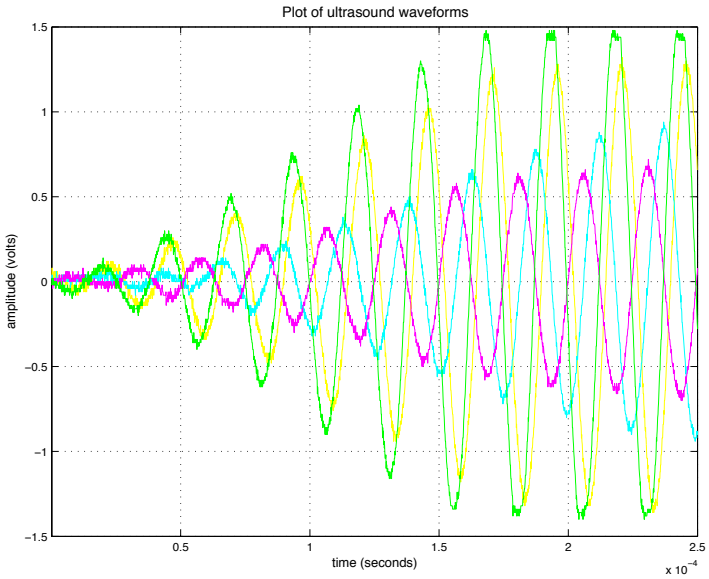


Figure 4-5: A plot of four ultrasound waveforms coming from the Compass hardware. Observe that even though the sensors are physically close together, the envelope of the ultrasound can vary greatly.

issues arise that make this difficult. First I will discuss the details of the correlation method and its typical performance. Then I will discuss modifications to improve this performance and to handle inaccurate measurements.

4.3.1 Correlation

The normalized cross correlation method for measuring differential distance works as described in Section 3.3. To visualize the principle behind this method, you can imagine taking two waveforms and sliding them relatively in time until the two look the most alike. The amount you have to slide to achieve this is the precise time delay. In my implementation, each waveform consists of 2500 eight-bit voltage samples. As I shift the waveforms, my implementation zeros the values that fall outside the window where the two waveforms overlap.

As shown in Figure 4-6, the correlation between two ultrasound waves has a frequency of 40 kHz. The peaks of the correlation are candidates for the actual phase difference between two waveforms. This method allows the *precise* measurement of the phase difference. This measurement, however, is not necessarily *accurate*. The peaks of the correlation are sometimes very close in magnitude and the wrong peak can easily be selected as the

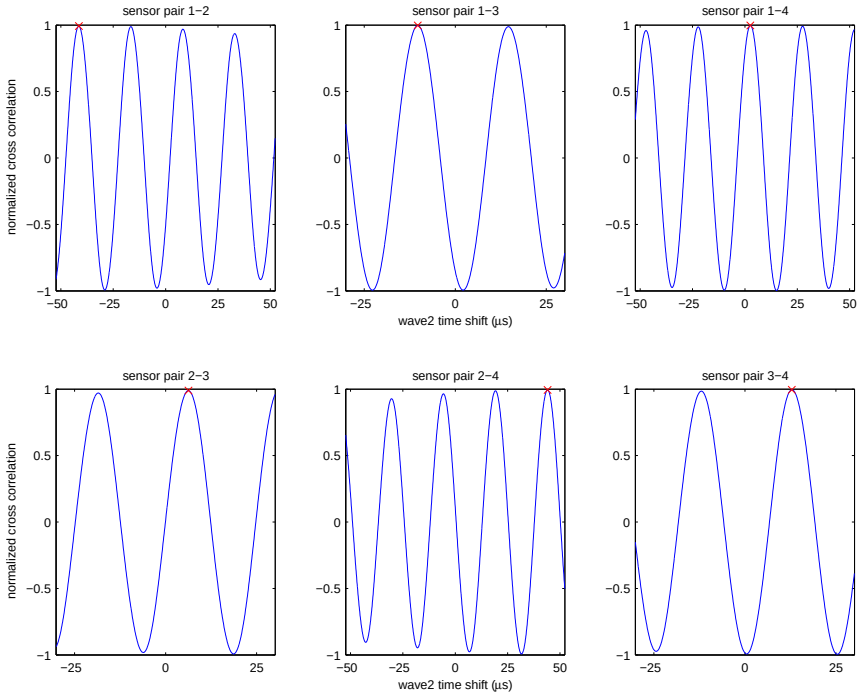


Figure 4-6: A plot of six inter-sensor normalized cross correlations for varying time delays. The input waveforms are those shown in Figure 4-5. Each correlation has a 40 kHz frequency as the two waveforms move in and out of phase. The marker indicates the peak of maximum correlation.

measurement.

In order to verify the *precision* of this method I carried out an experiment, duplicating the experiment in Section 2.3.4. Instead of using repeated zero-crossings to measure differential distance, however, I used the correlation method applied to the single rotating sensor pair. Figure 4-7 compares the performance of the two methods. The correlation method performs as well as the best set of data from the zero-crossing method and approaches the limit on precision imposed by the quantization error in the digitized waveforms.

In order to quantify the *accuracy* of the correlation method, I started testing the Compass prototype by again setting up an experiment similar to the one described in Section 2.3.4. I mounted the compass on the precision rotating platform and collected measurements for two 360° rotations at 10° increments. The measurements at each 10° increment include the four ultrasound waveforms generated by the sensor array in response to an ultrasound pulse sent by a beacon of known position. From these four waveforms, the correlation method measures the time delays on the six sensor pairs.

The experimental data includes over 400 measurements of time delays on sensor pairs.

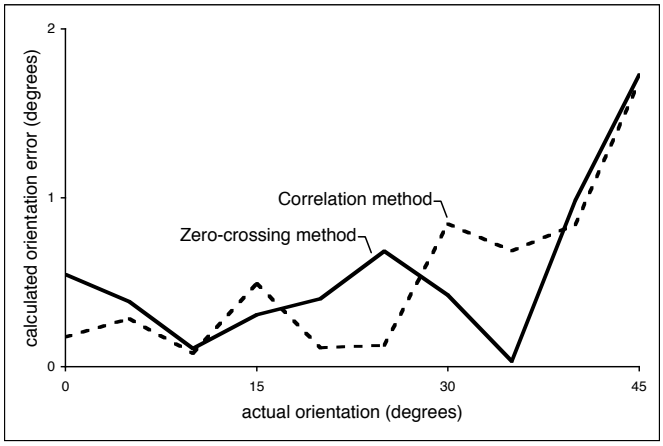


Figure 4-7: A plot of error in calculated orientation using two different methods for the differential distance measurement.

I compared these experimental measurements against the expected values I calculated in simulation, and found an overall error rate of 18.5%. The errors occur when the correlation method selects a peak that is either one period before or after the correct peak. From analysis of the data, the distribution of these errors looks random. That is, the errors are not clustered in any discernible way.

At any single orientation, I generate six measurements from one ultrasound pulse. For all six measurements to be correct at that orientation, each of the measurements corresponding to the six sensor pairs must be correct. These sets of six measurements were correct 33% of the time in the experimental data.

4.3.2 Filtering

I wish to improve the accuracy of the correlation measurement. At the lowest level, there is always noise on the amplified waveforms, and as discussed in Section 2.3.1 the envelope of the ultrasound waveforms is highly variable and unpredictable. For this reason, I briefly explored the use of filters to improve the quality of the waveform data.

Ideally, the filter should pass only the 40 kHz ultrasound waveforms. In hardware, I built several sharp band-pass filters centered around 40 kHz. However, I abandoned this approach very quickly because it is difficult to build hardware filters that are precisely matched. In particular, I observed that the filters did not have matching phase delays at the 40 kHz ultrasound frequency.

Having implemented functions for working with the waveforms in MATLAB, I also

tried building several types of software filters. Doing the filtering in software allows me to run each of the waveforms through one single filter and avoids the mismatch of hardware filters. However, the phase delay characteristics of standard filters fundamentally make them unsuitable for the Compass application. In particular, narrow and sharp filters exhibit extremely non-linear phase delays that cause the waveform envelope to be shifted several periods in the 2500 sample window. This corrupts the waveform information and reduces correlation accuracy. From this experience, it seems better to focus on building a good noise and oscillation-free circuit than to introduce variability from filtering the waveforms.

4.3.3 Pulse Shaping

The principle of the correlation measurement suggests that when the waveforms are more irregular, the correlation will have a more prominent peak. To demonstrate this, I applied the correlation method to waveforms generated by the standard Cricket ultrasound pulse and ultrasound noise that I generated by banging a pair of scissors against some metal. Compare the correlation plots in figure 4-8a with the plots in figure 4-8c. (a) shows the usual response to the standard Cricket pulse while (c) shows the response to the generated noise.

Waveforms that exhibit impulse-like behavior are easier to correlate. Operating on this principle, I want to generate a pulse that can correlate better in order to improve the accuracy of the correlation measurement. In Section 2.3.1, I discussed the Cricket beacon's capability to generate ultrasound pulses. To summarize, the version 2 Crickets generate a six-period 40 kHz square wave by toggling a pin on the Cricket microcontroller at 40 kHz for six cycles. This six-period square wave drives the ultrasound transmitter. The difficulty with the standard Cricket pulse is that it rings-up slowly. The current Cricket hardware has very limited capability to generate other ultrasound pulses because (i) the microcontroller can only generate signals containing low and high amplitude and (ii) the ultrasound transmitters are narrow-bandwidth.

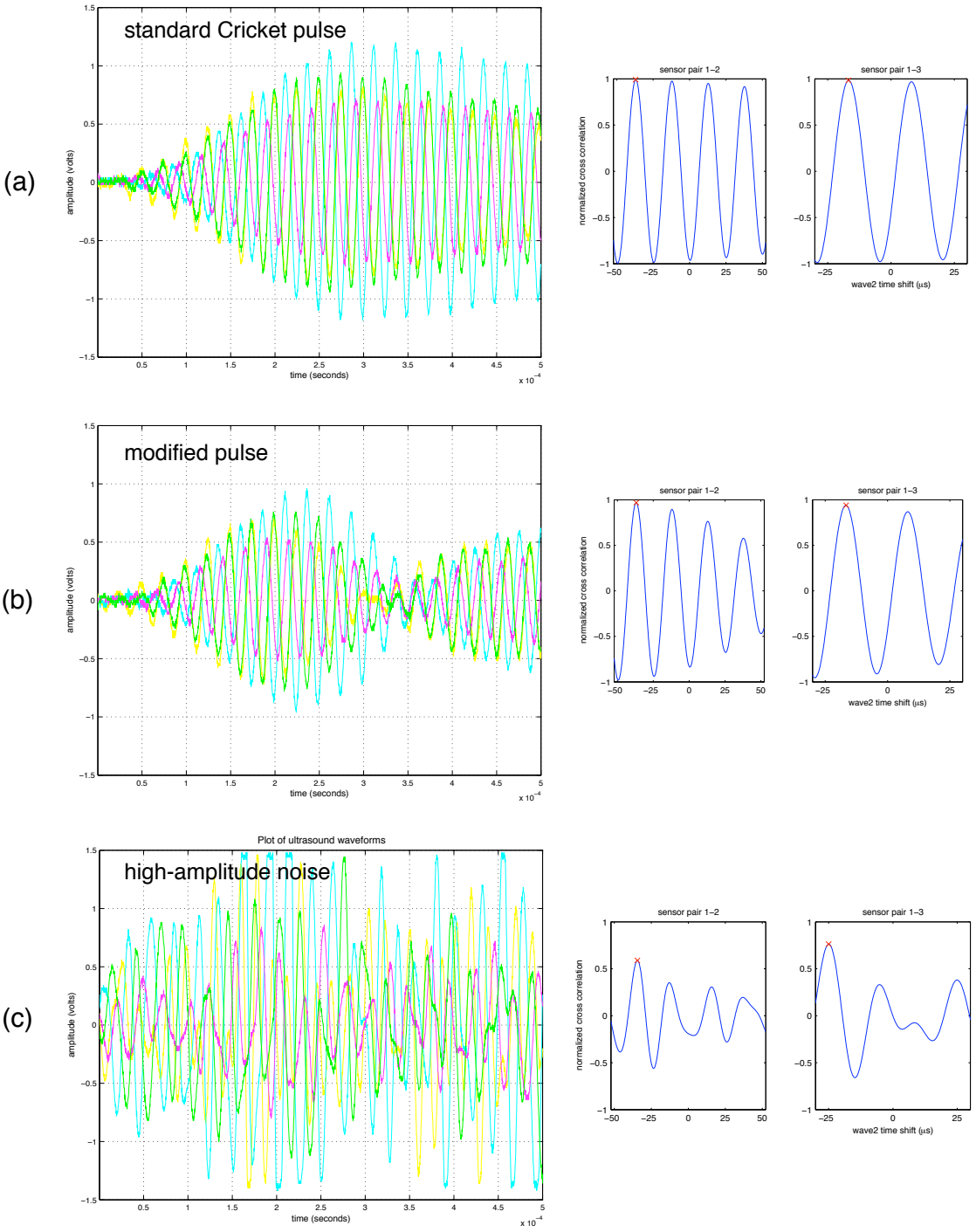


Figure 4-8: A plot of ultrasound signal and resulting normalized cross correlations for varying time delays on two sensor pairs. (a) is the response to the standard six-period Cricket ultrasound pulse. (b) is the response to the modified pulse. (c) is the response to ultrasound noise generated by banging a pair of scissors against metal. Observe that the noisy signal has prominent peaks of correlation. The modified pulse has a better correlation than the standard Cricket pulse.

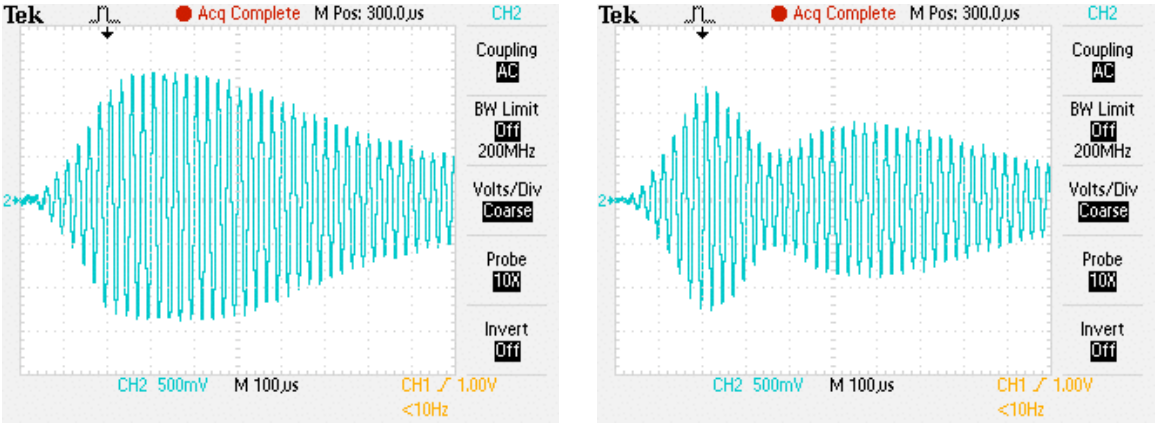


Figure 4-9: The standard Cricket ultrasound pulse on the left, and the modified pulse for the Compass on the right.

The current Cricket hardware can drive the ultrasound transmitter with square waves of varying frequency and phase. The transmitter's narrow-bandwidth prevents any useful use of square waves of varying frequency, but we can achieve some benefit by varying the phase of the driving signal. Figure 4-9 shows two received ultrasound waveforms, the left in response to the standard Cricket pulse and the right in response to a modified driving signal. This modified driving signal consists of the standard six-period 40 kHz square immediately followed by seven periods of 40 kHz square that are exactly π radians out-of-phase.

The modified beacon pulse takes advantage of the highly resonant nature of the ultrasound transmitter. The first few periods of the out-of-phase driving signal counteract the ringing on the transmitter. Then, the following periods of the out-of-phase signal generate a secondary out-of-phase ringing. The resulting ultrasound pulse has more features to aid the correlation method. Figure 4-8b shows the modified pulse and some corresponding correlation curves. The transmitting beacon position and compass orientation are held constant between Figure 4-8a and Figure 4-8b.

I conducted the accuracy experiments described in Section 4.3.1 with the beacon transmitting the modified pulse. I again collected over 400 measurements of time delays on sensor pairs. The use of the modified pulse reduced the measurement error rate on sensor pairs from 18.5% to 11.6%. Sets of six measurements were now correct 47.2% of the time in this collection of experimental data.

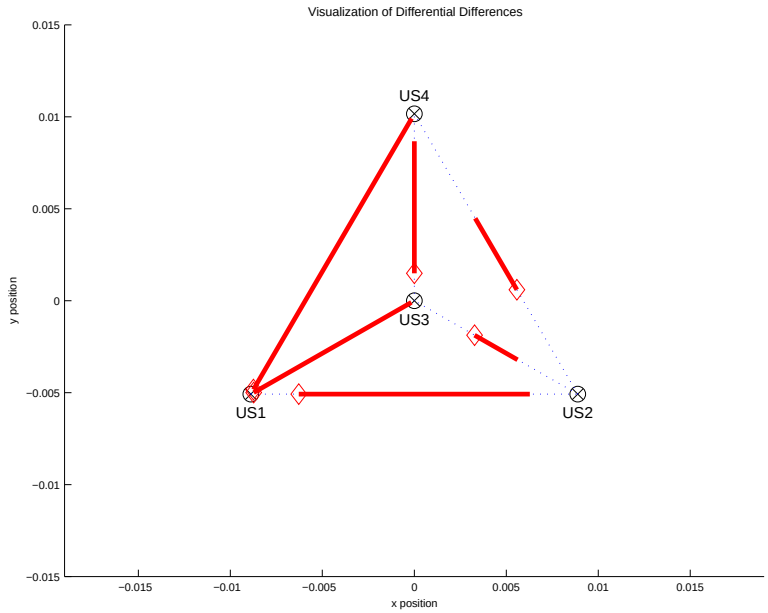


Figure 4-10: A visualization of the differential distances on sensor pairs. Lines representing the differential distances are displayed between each corresponding sensor pair. The length of the lines represent the differential distance magnitude. The diamond markers mark the end of the segment, which can be interpreted as an arrowhead indicating the sign of the differential distance. In this simulated example, the beacon is positioned at $(10, 10)$. This beacon position generates small differential distances between sensors 2 and 3 and sensors 2 and 4, and a large differential distance between sensors 1 and 4.

4.3.4 Error Detection

For each set of measurements produced by the Compass sensor array, I visualize the differential distances as shown in Figure 4-10. Relationships exist between these differential distances that allow me to perform error detection on the measurements. Imagine the differential distances forming a graph between the sensor pairs. For each cycle in this graph, which are triangles, the sum of the differential distance of two edges must equal the differential distance of the third edge. By looking at this relationship on the four cycles of the array geometry, I can detect and sometimes correct measurement errors.

These relationships permit error correction when there is only one error in the six measurements. From the collected data, however, there are often sets of measurements where there are two errors. To handle these errors, I can instead perform error detection and reject sets of measurements containing errors. With sufficient beacon density, I wait until I receive two error-free measurements from beacons. I then localize those beacons and apply the global translation step.

Chapter 5

Results

In the previous chapter, I presented a number of implementation techniques that solve problems in lower layers of the Compass. In this chapter I characterize the performance of the implemented Cricket Compass prototype. I present experimental setup and procedures and analyze the collected data, demonstrating end-to-end functionality of the Compass. I then suggest areas for future research based on these results.

5.1 Localizing Beacons

After receiving an ultrasound pulse from a beacon, the Compass attempts to estimate the position of that beacon in its own coordinate system. In this section, I characterize the Compass localization accuracy by setting up a single beacon at a fixed position, capturing an ultrasound pulse, and applying my implementation of the localization methods.

5.1.1 Setup and Procedure

In this experiment, I used two beacons: one for calibration and one as a target beacon for localization. I mounted the Compass prototype on the precision optics equipment described in Section 2.3.4 to perform precise rotation in the horizontal plane. I performed the experiment using the following setup and experimental procedure:

1. I selected an open lab area where multipath signals do not contribute to measurements. The lab area has a double-height ceiling, and beacons mounted on this ceiling do not always have sufficient range to reach ground level. Therefore, I mounted beacons on

(a) (b)

Figure 5-1: Photos of the beacon localization experiment. (a) shows the Compass prototype mounted on the rotating platform with an overlay of the axes in the Compass' coordinate space. (b) shows the lab space with the positions of the calibration beacon, target beacon, and Compass.

tripods and pillars in the lab space to achieve beacon placement at more moderate heights.

2. I placed one beacon above the rotating platform to assist in calibration. I located this calibration beacon precisely above the center of the Compass sensor array by using a plumb line hanging down from the beacon's ultrasound transmitter. The plumb bob rested directly above the central ultrasound receiver on the array.

3. Using a spirit level, I leveled the rotating platform along two orthogonal axes to ensure that the rotation occurs purely in the horizontal plane.

4. I turned the calibration beacon on, and leveled the sensor array such that when rotated, the waveforms received by the sensor array were minimally out of phase. I was unable to find a calibration such that all waveforms were precisely in-phase. After this calibration procedure, the sensor array was optimally level and rotated only in the horizontal plane.

5. I placed the target beacon on a pillar at a fixed distance of 1.4m and height of 1.0m with respect to the beacon as shown in Figure 5-1b. Also note that this target beacon has been modified to transmit the modified Compass pulse.

6. I rotated the Compass in 10° increments, from 0° to −180°. These degree headings are the headings of the beacon with respect to the Compass coordinate system. I

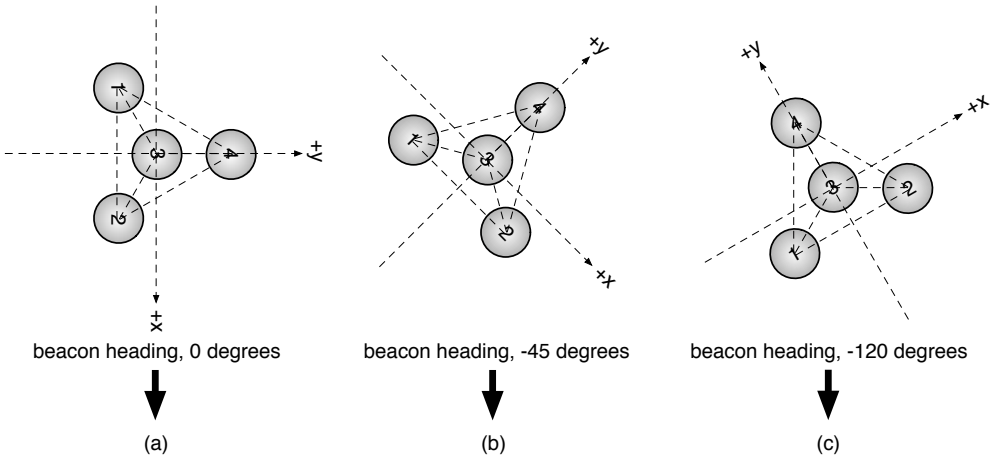


Figure 5-2: Illustration of the Compass rotation in the beacon localization experiment. The compass rotates from a beacon heading of (a) 0° through (b) −45° and (c) −120° on its way to −180°.

defined the Compass coordinate system using the sensor array geometry as shown in Figure 5-1a. I have also illustrated this rotation in Figure 5-2 with an overlay of the Compass x and y axes. This rotation is the same as if I held the Compass stationary and rotated the beacon about the Compass. However, by rotating the Compass on the precision mount, I am able to precisely fix the distances and amount of rotation.

7. At each 10° increment, I localize the beacon position. Specifically, I capture the ultrasound waveforms generated by the beacon pulse on the Compass sensor array. I use the correlation method to measure the six inter-sensor differential distances. I check these six distances for consistency and if there are no errors, I estimate the beacon's position. When there were errors, I attempted to localize up to five times at that heading before giving up.

5.1.2 Analysis

Figure 5-3 shows the results of this experiment and gives an idea of the Compass accuracy in localizing beacon positions. Note that at -180° and -170° I was unable to localize the beacon because of consistent measurement errors.

I have separated the plots in Figure 5-3 into the xy and xz planes to illustrate the different error in localizing the x, y, and z coordinates. Recall that the z coordinate calculation depends on the both the x and y coordinate estimates, as described by Equation 3.3. The

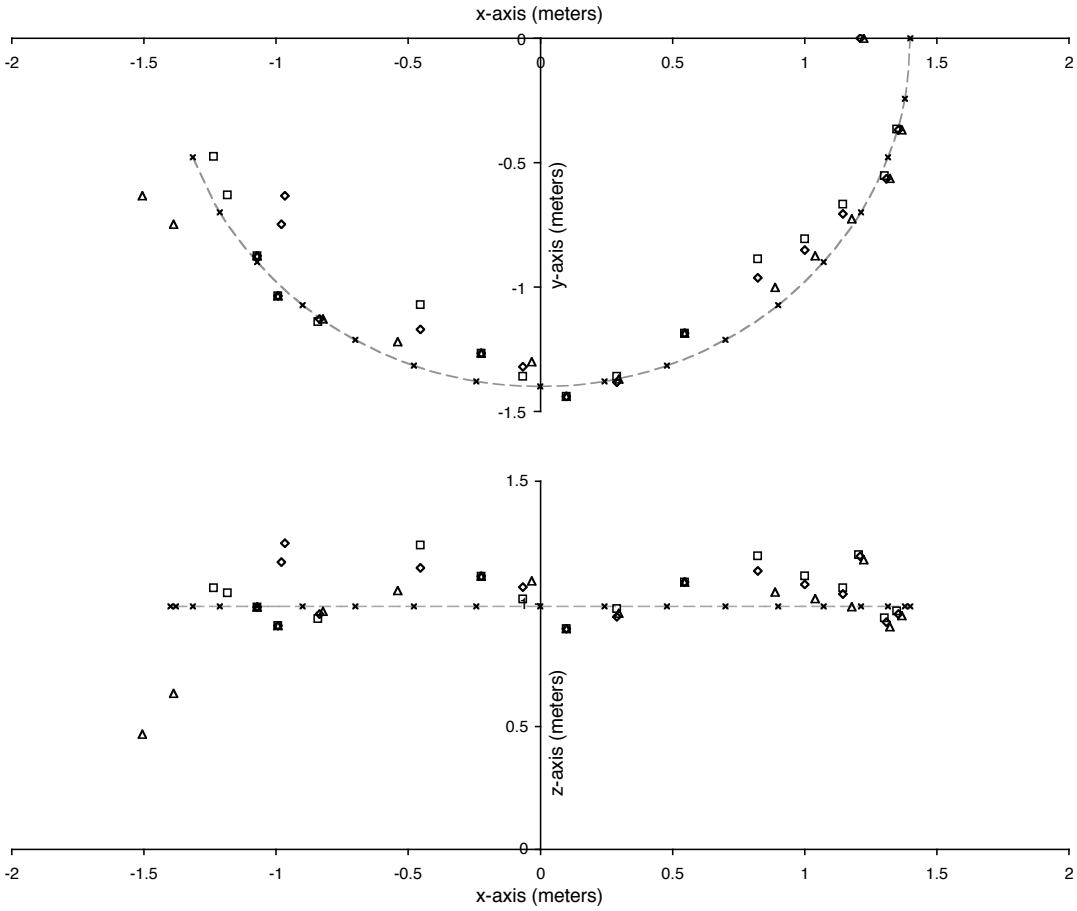


Figure 5-3: A plot of a beacon localized using the Compass. I rotated the Compass at 10° increments at a fixed distance of 1.4m and height of 1m with respect to the beacon. Both plots show the localized beacon position with respect to the Compass, which lies at the origin. The top plot shows positions in the xy plane while the bottom plot shows positions in the xz plane. The dotted lines represent the beacon's actual physical path, with Compass readings taken at the cross marks. The diamond, box, and triangle markers represent three different position estimates for each real beacon position.

plots show how the z error is directly related to the x and y error. Table 5.1 summarizes the error in the position estimates broken down by coordinate axis.

Looking more closely at figure 5-3, the xy plot shows that there is some grouping of positions that can explain a source of error. As shown in Figure 4-3, in each 30° sector, there are three sensor pairs on the Compass that can provide accurate differential distances. As the Compass rotates through 30° sectors in Figure 5-3, the sensor pairs used to make measurements changes. This effect is most prominent around the trio of readings at $-30°$, $-40°$, and $-50°$. The estimates at these three angles come from the same three sensor pairs and exhibit similar accuracy.

Coordinate Axis	Mean Error	Error Std. Dev.
x	9.20 cm	7.66 cm
y	7.05 cm	5.09 cm
z	10.10 cm	9.52 cm

Table 5.1: Mean error and standard deviation of error in each coordinate axis.

The change in accuracy as different sensor pairs are selected for use in localization is a manifestation of the difficulty of calibrating the sensor array. For just a single pair of rigidly mounted sensors, it is simple to make minute adjustments to cancel out any unevenness in their mounting. However, for a rigidly mounted array of four sensors, it is difficult to completely level the array to remove any unevenness.

5.2 End-to-End Orientation

The end-to-end functionality of the Compass is estimating its 3-DOF orientation in the Cricket coordinate system. In this section, I build upon the experiment of the previous section. I characterize and demonstrate the end-to-end functionality of the Compass using position estimates to two beacons and translation between Compass and Cricket coordinates.

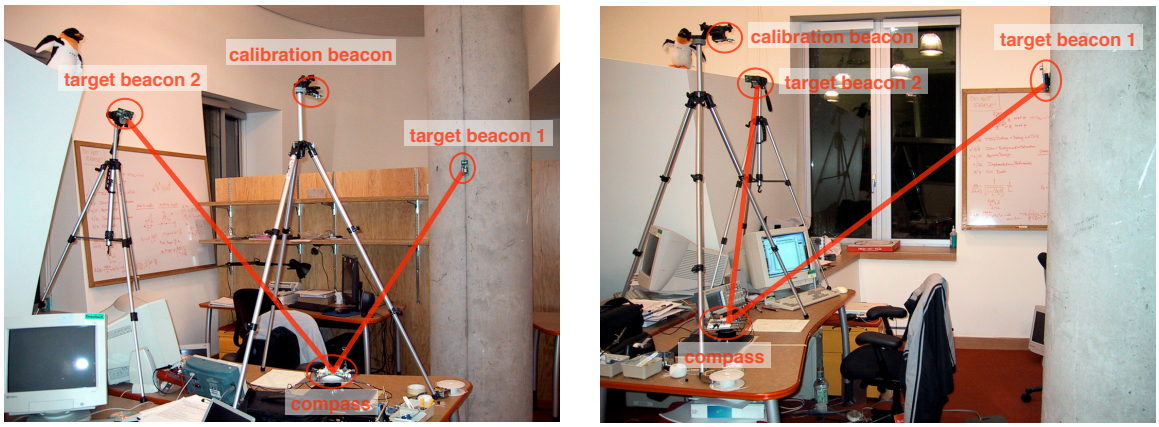
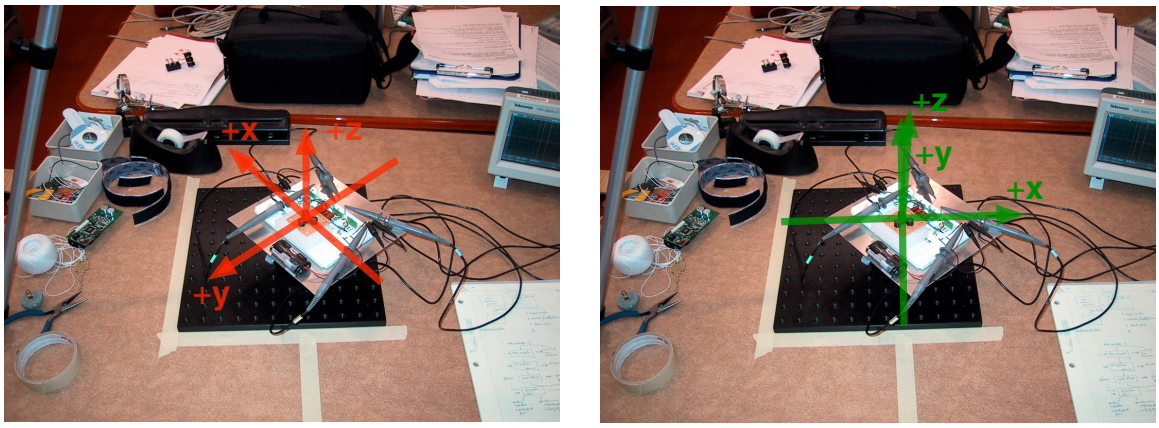


Figure 5-4: Photos of the Compass end-to-end demonstration. (a) and (b) show two views of the lab space with the positions of the calibration beacon, two target beacons, and Compass.



(a) (b)

Figure 5-5: Photos depicting the Compass and Cricket coordinate spaces in the end-to-end demonstration. (a) shows the Compass coordinate axes, which are determined by the orientation of the Compass. (b) shows the Cricket coordinate axes, which I defined when measuring the beacons and the Compass position.

5.2.1 Setup and Procedure

In this experiment, I used three beacons: one beacon for calibration and two target beacons for localization. I again mounted the Compass prototype on the precision optics equipment. I performed the experiment using the following setup and experimental procedure:

1. I followed the setup and calibration procedure in the previous section to optimally level the Compass sensor array.

2. I placed two beacons in the lab area as shown in Figure 5-4. Target beacon one is at a distance of 1.4m and height of 1.0m while target beacon two is at a distance of 1.5m and height of 1.2m.

3. I established an accurate and precise Cricket coordinate system by using a laser rangefinder and trilateration methods to calculate the two exact beacon positions and Compass position. Later, I input these coordinates and distance measurements into the localization and registration algorithms.

4. I then rotated the Compass in 10° increments for a full 360° rotation. At each increment, I localized each of the target beacons as described in the previous section. From the setup, I have the coordinates of three points in Cricket space, and from the Compass localization I have two points in Compass space. The third point in Compass

space is the origin, representing the Compass. I then applied the registration method to find the optimal rotation from the Cricket to the Compass coordinate systems.

Angle	Roll	Pitch	Yaw
0	2.3	1.1	0.2
7	no estimate		
17	-1.8	8.7	17.5
27	-4.9	3.6	26.4
37	-5.7	2.7	34.1
47	0.9	1.6	39.5
57	1.6	-3.8	52.6
67	no estimate		
77	no estimate		
87	no estimate		
97	-5.3	-5.9	100.1
107	no estimate		
117	no estimate		
127	no estimate		
137	no estimate		
147	no estimate		
157	no estimate		
167	no estimate		
177	4.3	-0.2	177

Angle	Roll	Pitch	Yaw
187	1.8	1.5	188.2
197	no estimate		
207	no estimate		
217	1.6	-3.3	213.7
227	-1.3	1.9	223.3
237	no estimate		
247	no estimate		
257	10.1	3.9	263.6
267	12.1	-2.6	271.9
277	no estimate		
287	no estimate		
297	no estimate		
307	4.3	1.4	309.2
317	no estimate		
327	1	1.9	327.7
337	3.4	-1.6	337.4
347	no estimate		
357	no estimate		

Table 5.2: Results of end-to-end orientation demonstration.

5.2.2 Analysis

Table 5.2 summarizes the results of this demonstration. For each rotation increment in the horizontal plane, I present the Compass estimate for its orientation in Cricket coordinates. I have converted the quaternion orientation output into the more intuitive roll-pitch-yaw notation. In this demonstration, I only rotated the Compass about the z-axis, so the roll and pitch should be zero, and the yaw should follow the angle rotated about the z-axis. The average orientation error was 2.6° in roll, 3.9° in pitch and 2.9° in yaw. The maximum observed error was 7.5° roll, 12.1° in pitch and 8.7° in yaw.

In 21 of the 37 orientations I tested, the Compass was unable to receive error-free measurements from both of the beacons and was therefore unable to estimate its orientation. I introduced a third beacon to help alleviate this problem. At the current Compass error rate, however, a beacon density of three was still insufficient to provide a reasonable guarantee of two error-free measurements at the Compass.

Figure 5-6 shows a visualization of the rotating Compass based on its estimated orientation. 12 of the 16 successful orientations are shown. This demonstration successfully proves the concept for the Cricket Compass.

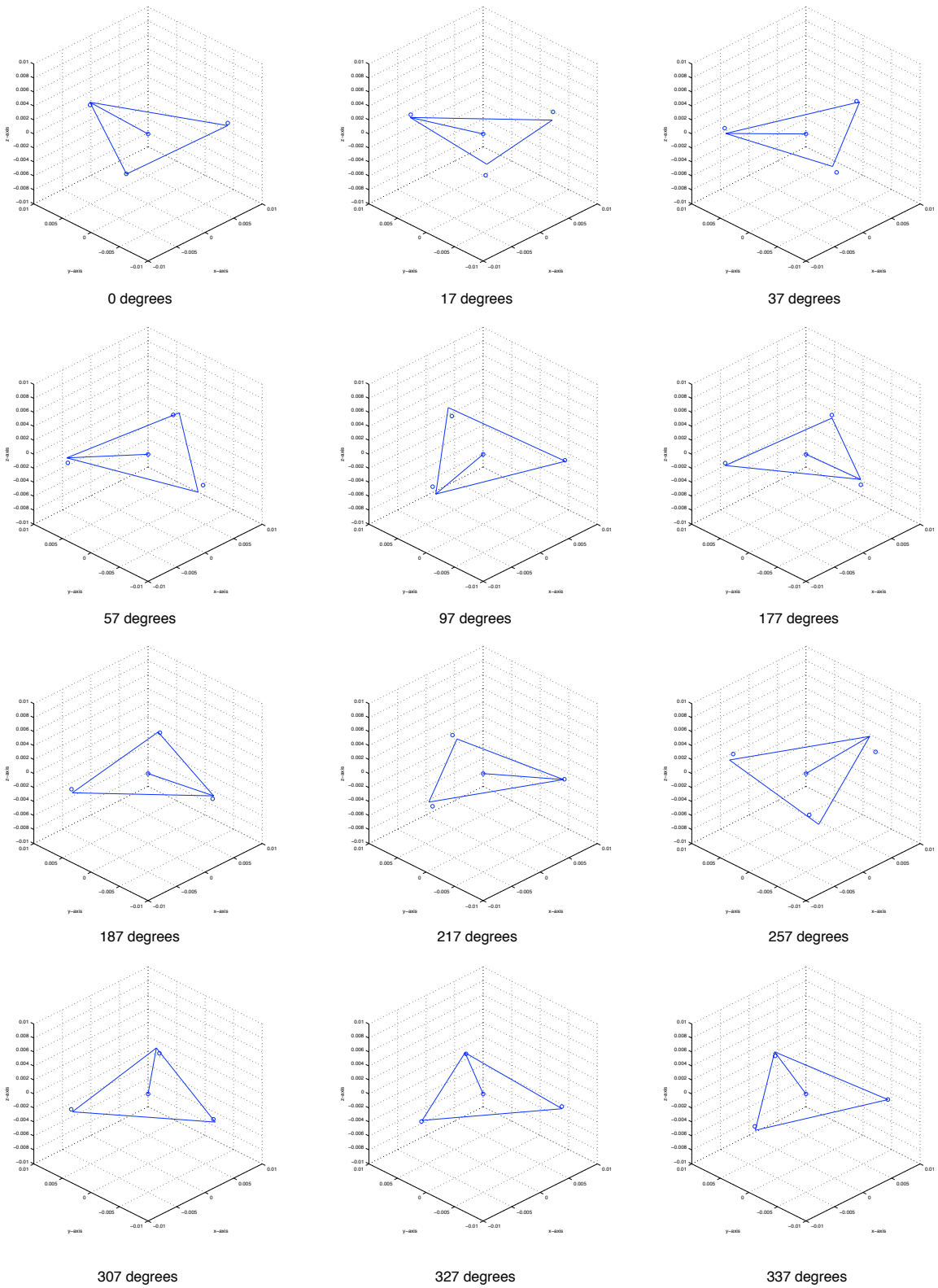


Figure 5-6: Estimated Compass end-to-end orientation. Each subfigure is a plot of the estimated Compass orientation for captioned degrees of rotation. The triangle outlines the three outer sensors; a line runs along the Compass $+y$-axis originating from the central sensor. The circles represent the actual locations of the sensors.

5.3 Future Work

Future research on the Compass could take the form of improvements or new methods in any layer of the Compass. At the lowest level, the primary area for improvement is the Compass measurement of the differential distances. Increasing the accuracy of the measurements would produce much more accurate position and orientation estimates. Increasing the integrity of the measurements would also permit the Compass to operate with fewer restrictions and would reduce the beacon density necessary for orientation. On a higher level, there may be new methods to correct errors or improve accuracy by leveraging new algorithms or Cricket infrastructure. Finally, the Compass hardware and software can be tightly integrated with Cricket to enable orientation-aware applications.

Improvements could take the following forms:

- Using more sensors and different sensor geometries for measurements.
- Designing and fabricating a precision hardware prototype, taking care to minimize noise and to ensure precise placement of ultrasound sensors.
- Experimenting with a more flexible high-speed analog-to-digital converter to explore new methods of measuring phase difference or to improve the correlation phase measurement method.
- Designing and characterizing linear phase filters for ultrasound waveforms.
- Exploring the use of higher-bandwidth ultrasound transceivers to improve the precision of distance measurements.
- Implementing a Cricket daughterboard to interface the Compass in real-time with Cricket. The daughterboard would require a dedicated processor to carry out the correlations. An application programming interface could also be specified.

Chapter 6

Contributions

In this thesis, my primary contributions are:

- building upon Priyantha et al.'s planar orientation method to enable orientation in the Cricket location system. I present experimental experience and results using this method, which motivate the design choices in the Cricket Compass.

- proposing a set of methods to calculate end-to-end orientation from an array of well placed sensors operating within the Cricket system. I use the normalized cross correlation between two waveforms to precisely measure the relative phase difference, from which I calculate a differential distance. I apply vector methods to differential distances measured between sensors in order to localize the source of an ultrasound pulse. I then register those localized sources with known positions in a reference coordinate system to determine end-to-end orientation. Implementing these methods does not require any trigonometric functions.

- designing a prototype of the Cricket Compass. I present a sensor array geometry that enables omnidirectional sensor coverage and discuss the effects of analog-to-digital conversion parameters. I also show how to bootstrap from a set of sensor array measurements to select a subset of accurate measurements to use in calculations.

- implementing a prototype of the Cricket Compass and demonstrating end-to-end functionality. I build supporting hardware and interface this hardware with an oscilloscope and software methods implemented in MATLAB. I describe an error detection method and modification to Cricket ultrasound pulses, which improve the Compass robustness.

Finally, I characterize the performance of this implementation while demonstrating end-to-end functionality.

In this thesis I also:

- characterize the use of Cricket ultrasound for estimating distance and orientation. I identify the varying ramp-up of ultrasound on narrow-bandwidth transceivers as the chief obstacle to timing ultrasound precisely. I outline the capabilities of Cricket ultrasound hardware and describe one method to modify Cricket ultrasound pulse characteristics.

- identify specific areas for improvement. Accurate differential distance measurements are critical to improving accuracy. New hardware can permit new techniques to improve or supersede those I have presented. Real-time integration with Cricket can permit better characterizations and can enable the development of context-aware applications.

Appendix A

Compass Hardware Design

A.1 Analog Ultrasound Gain Circuit

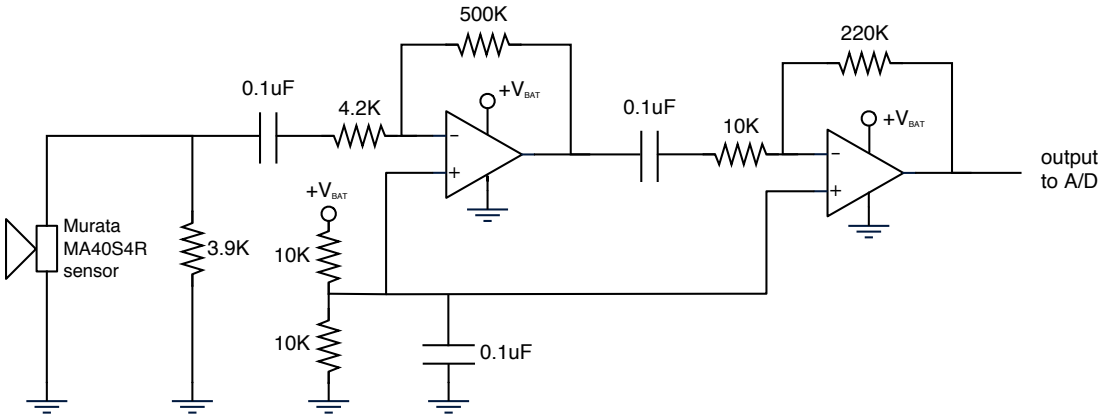


Figure A-1: The Compass ultrasound analog gain circuit. Four copies of this circuit provide gain on the four channels of ultrasound. This circuit operates from a supply of 2AA batteries.

A.2 Sensor Array Specifications and Naming

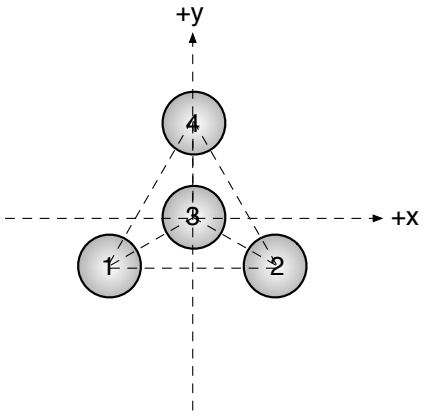


Figure A-2: Numbering scheme and coordinate axes defined for the ultrasound sensor array. Sensors are not drawn to scale.

Sensor	x	y
1	-14	-8
2	14	-8
3	0	0
4	0	16

Table A.1: I mounted the sensors using the circuit board inter-hole spacing of 2.54mm. This table shows the x and y positions of the center of the sensors in grid units where 4 grid units = 1 inter-hole space = 2.54mm.

Sensor 1	Sensor 2	Pair Number	Angle	Separation
1	2	1	0.0°	17.8 mm
1	3	2	29.7°	10.2 mm
1	4	3	59.7°	17.6 mm
2	3	4	120.3°	10.2 mm
2	4	5	150.3°	17.6 mm
3	4	6	180.0°	10.2 mm

Table A.2: Sensor pair naming, orientations, and separation distances used for position calculations.

A.3 Murata MA40S4R Ultrasonic Sensor Information

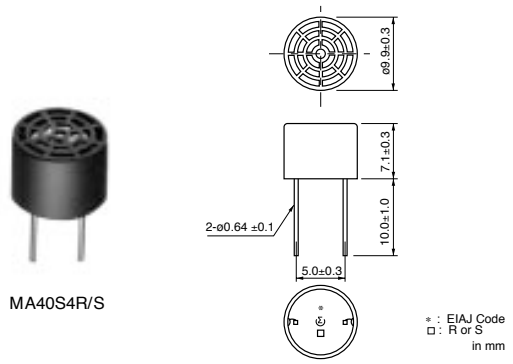


Figure A-3: Murata ultrasonic sensor physical dimensions.

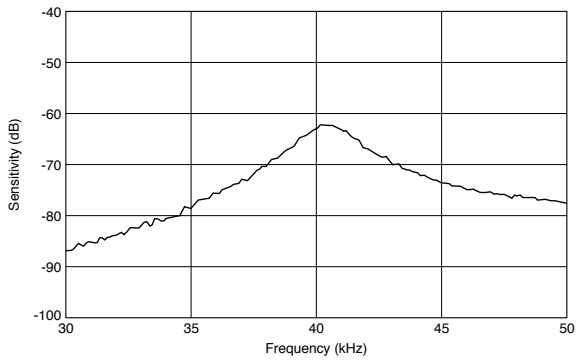


Figure A-4: Murata ultrasonic sensor frequency response.

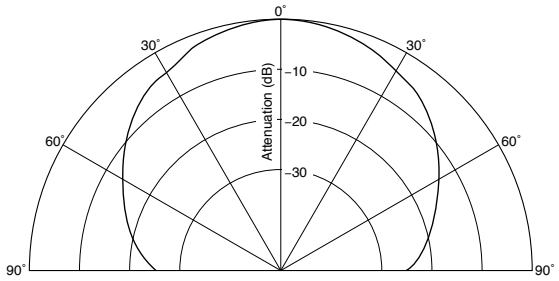


Figure A-5: Murata ultrasonic sensor directivity in sensitivity.

Appendix B

Compass Functionality

This appendix describes the setup and use of the Compass hardware and software.

B.1 Setup

In addition to the sensor array and ultrasound gain circuits, I use a Tektronix TDS2024 oscilloscope with the TDS2CMAX communication module installed. On the host computer, I run MATLAB v6.5 with the Instrument Control Toolbox installed. Additionally, I use the Quaternion Toolbox package v1.2.2 by Jay St. Pierre and 3D Rotations package by Giampiero Campa.[1]

Connect the four channels of amplified ultrasound output to the four input channels of the oscilloscope. In the options menu of the scope, set the RS-232 baud rate to 19200 and set flow control to hardware flagging. Connect the scope via RS-232 serial cable to the host computer through serial port 2. Serial port 1 can be used by editing the `tds_sample.m` file.

B.2 Demonstration

To demonstrate localization of a single beacon:

1. Place a beacon in a location where the Compass can hear the ultrasound pulses. You can check for this by manually triggering the oscilloscope and looking at the waveforms while the beacon is transmitting ultrasound.

2. Use the function `compass_single` to start the beacon localization method. The function will prompt you for the distance to the beacon in meters. The function will then capture the next set of ultrasound waveforms that trigger the scope.

3. The function will perform error detection. If the measurements pass the error detection, the function returns the (x, y, z) coordinates of the beacon in Compass coordinate space as defined in Appendix A by the sensor array. If an error is detected, the function returns the coordinates $(0, 0, 0)$. The coordinates are in units of meters.

To demonstrate end-to-end orientation using two beacons:

[1] These files are available at MATLAB Central. http://www.mathworks.com/matlabcentral/

1. Setup the Cricket coordinate space, either using Crickets or other methods. Find the location of the Compass and two beacons in this space. Place the beacons in locations where the Compass can hear the ultrasound pulses. You can check for this by manually triggering the oscilloscope and looking at the waveforms while the beacons are transmitting ultrasound.

2. Use the function `cricket_compass` to start the process. The function will prompt you for the (x, y, z) coordinates of beacon 1, beacon 2, and the Compass in the Cricket coordinate space from step 1.

3. The function will prompt you to turn off all beacons except beacon 1. Once this is done, strike any key and `cricket_compass` will attempt to localize the beacon. If the measurement fails to pass error detection, you can try again or abort.

4. After beacon 1 has been localized, `cricket_compass` will repeat step 3 for beacon 2.

5. If `cricket_compass` localizes both beacon 1 and 2, the function will calculate and return the end-to-end orientation in unit quaternion notation. It will also display the Euler x-y-z (roll-pitch-yaw) notation.

B.3 MATLAB functions

The Compass software methods are implemented in MATLAB m-files. The Compass m-files are available in the Cricket CVS repository. The following is the source for each MATLAB function, including the self-documentation comments for each function.

```
function array_graph_plot(phases)

% ARRAY_GRAPH_PLOT Cricket Compass utility function
%
%  ARRAY_GRAPH_PLOT(PHASES) plots the measured differential distances
%  between sensors on the compass ultrasound sensor array.
%
%  PHASES is a 4x4 matrix of measured phase differentials of the form:
%  PHASES = [    0   t12   t13   t14; ...
%              -t12     0   t23   t24; ...                                  10
%              -t13 -t23     0   t34; ...
%              -t14 -t24 -t34     0 ];
%  where txy is the phase difference from sensor x to sensor y.
%
%  This function converts the phases into distances using predefined speed
%  of sound and plots them on a representation of the sensor array.
%
%  This function has a predefined sensor array geometry. All distances are
%  in meters, and all times are in seconds.
%                                                                          20
% See also US_CORRELATE, EXPECTED_PHASE.
%
% Package: Cricket Compass MATLAB
% Author:  Kevin J Wang, kevin1@graphics.csail.mit.edu
% Date:    2004/05/23

speed_sound = 344; % speed of sound 344 m/s at approx 22 Celsius

% set sensor array geometry. each row corresponds to x,y,z position of
% sensor# where #=row. sensor3 lies at the origin. conversion from circuit  30
% board layout inter-hole spacing is 4 units = 2.54mm
```

```
sensor= [−14 −8 0; 14 −8 0; 0 0 0; 0 16 0] .∗ (2.54e−3 / 4);

% visualize signed phase differences as vectors between points
hold on;
title('Visualization of Differential Differences', 'FontSize', 8);
set(gca, 'FontSize', 7);
axis([−1.5 1.5 −1.5 1.5].∗1e−2);
Xlabel('x position');
Ylabel('y position');

% define what the plotted sensors look like
sensor_point_style = 'blackx';
sensor_point_size = 10;
sensor_style = 'blacko';
sensor_size = 10;

% plot points representing ultrasound sensors on sensor array
plot(sensor(:,1), sensor(:,2), sensor_point_style, 'MarkerSize', sensor_point_size);
plot(sensor(:,1), sensor(:,2), sensor_style, 'MarkerSize', sensor_size);

% label ultrasound sensors
text(sensor(1,1), (sensor(1,2)−1e−3), 'US1', 'HorizontalAlignment', 'center');
text(sensor(2,1), (sensor(2,2)−1e−3), 'US2', 'HorizontalAlignment', 'center');
text(sensor(3,1), (sensor(3,2)−1e−3), 'US3', 'HorizontalAlignment', 'center');
text(sensor(4,1), (sensor(4,2)+1e−3), 'US4', 'HorizontalAlignment', 'center');

% plot dotted lines between sensor pairs
US_line_style = 'b:';
for j = 2:4
    plot([sensor(1,1) sensor(j,1)], [sensor(1,2) sensor(j,2)], US_line_style);        % pairs: 1-2, 1-3, 1-4
end
for j = 3:4
    plot([sensor(2,1) sensor(j,1)], [sensor(2,2) sensor(j,2)], US_line_style);        % pairs: 2-3, 2-4
end
plot([sensor(3,1) sensor(4,1)], [sensor(3,2) sensor(4,2)], US_line_style);            % pair 3-4

% calculate direction vectors for each sensor pair
dir_vector = repmat(NaN, 6, 3);
for i = 1:3
    dir_vector(i,:) = sensor((i+1),:) − sensor(1,:);        % pairs: 1-2, 1-3, 1-4
end
for i = 4:5
    dir_vector(i,:) = sensor((i−1),:) − sensor(2,:);        % pairs: 2-3, 2-4
end
dir_vector(6,:) = sensor(4,:) − sensor(3,:);                % pair:  3-4

% find midpoint between each sensor pair, this is where the differential
% distance is centered
start_point = [ sensor(1,:); sensor(1,:); sensor(1,:); sensor(2,:); sensor(2,:); sensor(3,:)];
mid_point = start_point + dir_vector./2;

% normalize direction vectors, also calculate separation distances
for i = 1:6
    dir_vector_norm(i,:) = dir_vector(i,:) ./ norm(dir_vector(i,:));
    separation(i,1) = norm(dir_vector(i,:));
end

% prepare to plot phases (converted to distance) in correct direction
% repackage phase information into a vector
phases_vector = [ phases(1,2); phases(1,3); phases(1,4); phases(2,3); phases(2,4); phases(3,4) ];
distances = phases_vector .∗ speed_sound;

end_point = [ sensor(2,:); sensor(3,:); sensor(4,:); sensor(3,:); sensor(4,:); sensor(4,:)];

to_end = repmat(NaN, 6, 3); to_start = repmat(NaN, 6, 3);

% plot each of the six differential distances, centered on the midpoints
for i = 1:6
```

```
    to_end(i,:) = dir_vector_norm(i,:);
    to_start(i,:) = dir_vector_norm(i,:) .* (-1);
    to_end_point(i,:) = mid_point(i,:) + to_end(i,:) .* (distances(i,:)/2);
    to_start_point(i,:) = mid_point(i,:) + to_start(i,:) . *(distances(i,:)/2);
    plot([to_start_point(i,1) to_end_point(i,1)], [to_start_point(i,2) to_end_point(i,2)], ...
        'r-', 'LineWidth', 3);
    plot(to_end_point(i,1), to_end_point(i,2), 'rd', 'MarkerSize', 10);
end
axis equal;
```

```
function [position_3d] = compass_single()

% COMPASS_SINGLE Localize a single beacon in Compass coordinates.
%
%  [POSITION_3D] = COMPASS_SINGLE localizes the position of a single
%  beacon. The function prompts the user for the distance to the Cricket
%  beacon and captures measurements on the Compass sensor array via the
%  oscilloscope. It then detects errors on the measurements. If
%  measurements pass error detection, it then estimates the beacon
%  location.
%
%  POSITION_3D [x y z], the estimate of beacon location in Compass
%  coordinates. Units are in meters. Value is [0 0 0] if measurements
%  failed to pass error detection.
%
% Package: Cricket Compass MATLAB
% Author:  Kevin J Wang, kevin1@graphics.csail.mit.edu
% Date:    2004/05/23

fprintf(1, '\n::: Starting compass routines :::\n');

d_measured = input('Enter distance to beacon (meters): ');

%Acquire waveform from tektronix scope, transfer data via serial into matlab
[w1, w2, w3, w4, increment] = tds_sample;

%get phase measurements from correlation and shift method
phases = US_correlate(w1, w2, w3, w4, increment, false);

% prevent divide by zeros by adding tiny values to the phase matrix
phases = phases + (ones(4).*1e-12);

%detect errors on this phase measurement
error = error_detect(phases);
if ~error
    position_3D = position_solver(phases, d_measured);
else
    fprintf(1, 'Bypassing beacon1 position estimation.\n');
    position_3D = [0 0 0];
end
```

```
function [ORIENTATION] = cricket_compass()

% CRICKET_COMPASS Estimate the orientation of the Compass.
%
%  [ORIENTATION] = CRICKET_COMPASS determines the orientation of the Compass.
%  The function interactively steps the user through the steps:
%      1. Asking for the location of:
%          a. beacon1 in *Cricket* coordinates
%          b. beacon2 in *Cricket* coordinates
%          c. the Compass in *Cricket* coordinates
%      2. Localizing beacon1 in *Compass* coordinates
%      3. Localizing beacon2 in *Compass* coordinates
%  It then outputs the estimate of orientation.
%
```

```
%  ORIENTATION is an estimate of the quaternion that converts the Cricket
%  coordinate system into the Compass coordinate system, which is the
%  end-to-end orientation of the physical compass in the Cricket
%  coordinate space.
%
% See also COMPASS_SINGLE, ROTATION_REGISTRATION.
%
% Package: Cricket Compass MATLAB
% Author:  Kevin J Wang, kevin1@graphics.csail.mit.edu
% Date:    2004/05/23

fprintf(1, '\n::: Starting compass routines :::\n');

%%% first get beacon1, beacon2 and listener/compass coordinates:
fprintf(1, '\n*** Asking for beacon1, beacon2 and compass positions in Cricket coordinates ***\n');
fprintf(1, 'Input position coordinate values in meters.\n');

b1x = input('Enter beacon1 x: ');
b1y = input('Enter beacon1 y: ');
b1z = input('Enter beacon1 z: ');
b1 = [b1x b1y b1z];
fprintf(1, '=> Beacon1 position: (%2.2f, %2.2f, %2.2f)\n\n', b1);

b2x = input('Enter beacon2 x: ');
b2y = input('Enter beacon2 y: ');
b2z = input('Enter beacon2 z: ');
b2 = [b2x b2y b2z];
fprintf(1, '=> Beacon2 position: (%2.2f, %2.2f, %2.2f)\n\n', b2);

listener_x = input('Enter compass x: ');
listener_y = input('Enter compass y: ');
listener_z = input('Enter compass z: ');
listener = [listener_x listener_y listener_z];
fprintf(1, '=> Listener position: (%2.2f, %2.2f, %2.2f)\n\n', listener);

distance1 = norm(b1 - listener);
distance2 = norm(b2 - listener);
fprintf(1, '=> Beacon1 distance: %2.2f, beacon2 distance: %2.2f meters\n', distance1, distance2);

%%% get ultrasound input from the two beacons
got_one = false;
got_two = false;

fprintf(1, '\n\n\n::: Attempting to estimate beacon1 position :::\n');
fprintf(1, 'Turn BEACON1 on. Make sure all other beacons are OFF.\n');
fprintf(1, 'Press any key when ready. ^C to abort.\n');
pause;
while (~got_one)
    %Acquire waveform from tektronix scope, transfer data via serial into matlab
    [w1, w2, w3, w4, increment] = tds_sample;

    %get phase measurements from correlation and shift method
    phases1 = US_correlate(w1, w2, w3, w4, increment, false);

    % prevent divide by zeros by adding tiny values to the phase matrix
    phases1 = phases1 + (ones(4).*1e-12);

    %detect errors on this phase measurement
    error1 = error_detect(phases1);
    if ~error1
        b1_compass = position_solver(phases1, distance1);
        got_one = true;
        fprintf(1, '\n=> Got estimate for beacon1 position.\n')
    else
        fprintf(1, 'Cannot estimate beacon1 position:\nTo try again, hit any key. ^C to give up.');
        pause;
    end
end
```

```
% repeat procedure for beacon2
fprintf(1, '\n\n\n::: Attempting to estimate beacon2 position :::\n');
fprintf(1, 'Turn BEACON2 on. Make sure all other beacons are OFF.\n');
fprintf(1, 'Press any key when ready. ^C to abort.\n');
pause;
while (~got_two)
    [w1, w2, w3, w4, increment] = tds_sample;
    phases2 = US_correlate(w1, w2, w3, w4, increment, false);
    phases2 = phases2 + (ones(4).*1e-12);
    error2 = error_detect(phases2);
    if ~error2
        b2_compass = position_solver(phases2, distance2);
        got_two = true;
        fprintf(1, '\n=> Got estimate for beacon2 position.\n')
    else
        fprintf(1, 'Cannot estimate beacon2 position:\nTo try again, hit any key. ^C to give up.');
    pause;
    end
end

% output to command window
fprintf(1, '\n\n\n::: Registering end-to-end orientation :::\n');
fprintf(1, 'Cricket coordinates: \n');
fprintf(1, 'Beacon1:  (%2.2f, %2.2f, %2.2f)\n', b1);
fprintf(1, 'Beacon2:  (%2.2f, %2.2f, %2.2f)\n', b2);
fprintf(1, 'Listener: (%2.2f, %2.2f, %2.2f)\n\n', listener);

fprintf(1, 'Compass estimates: \n');
fprintf(1, 'Beacon1:  (%2.2f, %2.2f, %2.2f)\n', b1_compass);
fprintf(1, 'Beacon2:  (%2.2f, %2.2f, %2.2f)\n', b2_compass);
fprintf(1, 'Compass:  (0.00, 0.00, 0.00)\n\n');

Compass_to_Cricket = rotation_registration(b1_compass, b2_compass, listener, b1, b2);

% Compass_to_Cricket is the quaternion converting Compass coordinate space
% into Cricket coordinate space. Its inverse converts in the other
% direction, and represents the orientation of the Compass in Cricket
% coordinate space. The inverse of a unit quaternion is also its conjugate;
% I pass the quaternion in physics notation [(x y z), w].
Cricket_to_Compass = qconj([Compass_to_Cricket(2:4) Compass_to_Cricket(1)]);
orientation = [Cricket_to_Compass(4) Cricket_to_Compass(1:3)];

% also convert the orientation into the roll-pitch-yaw angles for reference
transformation_matrix = x2t([0; 0; 0; 1; Cricket_to_Compass'],'qua');
rotation_parameters = t2x(transformation_matrix,'rpy');
rotation_parameters = rotation_parameters.*(180/pi);
fprintf(1, 'Rotation in R-P-Y angles (euler x-y-z convention):\n');
fprintf(1, 'Roll: %3.1f Pitch: %3.1f Yaw: %3.1f\n', ...
        rotation_parameters(5), rotation_parameters(6), rotation_parameters(7));
```

```
function [error] = error_detect(phases)

% ERROR_DETECT Detect errors in Compass phase measurements.
%
%  [ERROR] = ERROR_DETECT(PHASES) checks the phases for cycle consistency
%  based on Compass sensor array geometry. These consistency checks can
%  detect when the phase measurements deviate by a whole period, which can
%  happen when using the US_CORRELATE phase measurement function.
%
%  PHASES is a 4x4 matrix of measured phase differentials of the form:
%  PHASES = [    0   t12   t13   t14; ...
%              -t12     0   t23   t24; ...
%              -t13  -t23     0   t34; ...
%              -t14  -t24  -t34     0 ];
%  where txy is the phase difference from sensor x to sensor y.
%
```

```
%  ERROR is true when the measurements fail the consistency checks.
%
% See also US_CORRELATE, EXPECTED_PHASE.
%
% Package: Cricket Compass MATLAB
% Author:  Kevin J Wang, kevin1@graphics.csail.mit.edu
% Date:    2004/05/23

fprintf(1, '\n*** Detecting measurement errors ***\n');

% initialize
parityA = true;
parityB = true;
parityC = true;
parityD = true;

% allow for some measurement error, but not more than the error caused by
% the phase measurement deviating a whole period from the true value
parity_threshold = 10e-6;  % 10 microseconds is less than half a period

% check each cycle for consistency
if ( ((phases(1,3) + phases(3,4)) >= (phases(1,4) - parity_threshold)) & ...
     ((phases(1,3) + phases(3,4)) <= (phases(1,4) + parity_threshold)) )
    parityA = false; fprintf(1, 'test A pass. ');
end
if ( ((phases(2,3) + phases(3,4)) >= (phases(2,4) - parity_threshold)) & ...
     ((phases(2,3) + phases(3,4)) <= (phases(2,4) + parity_threshold)) )
    parityB = false; fprintf(1, 'test B pass. ');
end
if ( ((phases(1,3) + phases(3,2)) >= (phases(1,2) - parity_threshold)) & ...
     ((phases(1,3) + phases(3,2)) <= (phases(1,2) + parity_threshold)) )
    parityC = false; fprintf(1, 'test C pass. ');
end
if ( ((phases(1,2) + phases(2,4)) >= (phases(1,4) - parity_threshold)) & ...
     ((phases(1,2) + phases(2,4)) <= (phases(1,4) + parity_threshold)) )
    parityD = false; fprintf(1, 'test D pass. ');
end

% report errors, and guess which measurement might be inconsistent (guess
% is true if there is only one error among the six measurements)
stop = false;
error = false;
if ( (~parityA) & (~parityB) & (~parityC) & (~parityD))
    fprintf(1, '\nAll measurements consistent.\n');
    stop = true;
end
if ( (parityC & parityD) & ~stop & (~parityA & ~parityB) )
    fprintf(1, '\nPair 1 might be inconsistent.\n');
    stop = true; error = true;
end
if ( (parityA & parityC) & ~stop & (~parityB & ~parityD) )
    fprintf(1, '\nPair 2 might be inconsistent.\n');
    stop = true; error = true;
end
if ( (parityA & parityD) & ~stop & (~parityB & ~parityC) )
    fprintf(1, '\nPair 3 might be inconsistent.\n');
    stop = true; error = true;
end
if ( (parityB & parityC) & ~stop & (~parityA & ~parityD) )
    fprintf(1, '\nPair 4 might be inconsistent.\n');
    stop = true; error = true;
end
if ( (parityB & parityD) & ~stop & (~parityA & ~parityC) )
    fprintf(1, '\nPair 5 might be inconsistent.\n');
    stop = true; error = true;
end
if ( (parityA & parityB) & ~stop & (~parityC & ~parityD) )
    fprintf(1, '\nPair 6 might be inconsistent.\n');
```

```
    stop = true; error = true;
end

if error
    fprintf(1, 'Error detected!\n');
end
if ~stop
    fprintf(1, 'Inconsistent State!\n');
    error = true;
end
```

```
function [phases] = expected_phase(beacon)

% EXPECTED_PHASE Cricket Compass utility function
%
%  [PHASES] = EXPECTED_PHASE(BEACON) generates the expected phase
%  measurements on the Compass sensor array.
%
%  BEACON is [x y z] coordinate vector of a beacon using the Compass
%  coordinate space.
%
%  PHASES is a 4x4 matrix of measured phase differentials of the form:
%  PHASES = [    0   t12   t13   t14; ...
%              -t12     0   t23   t24; ...
%              -t13 -t23     0   t34; ...
%              -t14 -t24 -t34     0 ];
%  where txy is the phase difference from sensor x to sensor y.
%
%  This function converts distances into phases using predefined speed of
%  sound.
%  This function has a predefined sensor array geometry.
%  All distances are in meters, and all times are in seconds.
%
% See also ARRAY_GRAPH_PLOT, POSITION_SOLVER.
%
% Package: Cricket Compass MATLAB
% Author:  Kevin J Wang, kevin1@graphics.csail.mit.edu
% Date:    2004/05/23

speed_sound = 344; % speed of sound 344 m/s at approx 22C

% define sensor array geometry. each row corresponds to x,y,z position of
% sensor# where #=row. sensor3 lies at the origin. conversion from circuit
% board layout inter-hole spacing is 4 units = 2.54mm
sensor= [-14 -8 0; 14 -8 0; 0 0 0; 0 16 0] .* (2.54e-3 / 4);

% calculate exact distance of each sensor to beacon
d1 = norm(beacon - sensor(1,:));
d2 = norm(beacon - sensor(2,:));
d3 = norm(beacon - sensor(3,:));
d4 = norm(beacon - sensor(4,:));

% convert differential distance into phase differences
t12 = (d2 - d1) / speed_sound;
t13 = (d3 - d1) / speed_sound;
t14 = (d4 - d1) / speed_sound;
t23 = (d3 - d2) / speed_sound;
t24 = (d4 - d2) / speed_sound;
t34 = (d4 - d3) / speed_sound;

% package phase information into phase matrix
phases = [ 0 t12 t13 t14; -t12 0 t23 t24; -t13 -t23 0 t34; -t14 -t24 -t34 0 ];
```

```
function [position_3d] = position_solver(phases, d_measured)
```

```
% POSITION_SOLVER Localize a beacon from Compass measurements.
%
%  [POSITION_3D] = POSITION_SOLVER(PHASES, D_MEASURED) generates estimate
%  for the position of a beacon based on the measured phase differences
%  received on the Compass ultrasound sensor array.
%
%  PHASES is a 4x4 matrix of measured phase differentials of the form:
%  PHASES = [    0   t12   t13   t14; ...                                  10
%              -t12     0   t23   t24; ...
%              -t13  -t23     0   t34; ...
%              -t14  -t24  -t34     0 ];
%  where txy is the phase difference from sensor x to sensor y.
%  D_MEASURED is the measured distance to the beacon.
%
%  POSITION_3D is the estimated [x y z].
%
%  This function converts phases into distances using predefined speed of
%  sound.                                                                  20
%  This function has a predefined sensor array geometry.
%  All distances are in meters, and all times are in seconds.
%
% See also US_CORRELATE, EXPECTED_PHASE.
%
% Package: Cricket Compass MATLAB
% Author:  Kevin J Wang, kevin1@graphics.csail.mit.edu
% Date:    2004/05/23

fprintf(1, '\n*** Estimating beacon position ***\n');                      30

%use voting algorithm to bootstrap which pairs to use for position solving
[pairA, pairB, pairC] = voting(phases, false);

%solve for position estimates using each combination of sensor pairs
pos1 = position_solve_1(phases, d_measured, pairA, pairB);
pos2 = position_solve_1(phases, d_measured, pairB, pairC);
pos3 = position_solve_1(phases, d_measured, pairA, pairC);

% display estimates in MATLAB command window                              40
fprintf(1, 'xyz est from pairs %g+%g: (%2.3f, %2.3f, %2.3f) meters, ', pairA, pairB, pos1);
fprintf(1, '%3.1f degrees in compass xy plane\n', atan2(pos1(2),pos1(1))*180/pi);
fprintf(1, 'xyz est from pairs %g+%g: (%2.3f, %2.3f, %2.3f) meters, ', pairB, pairC, pos2);
fprintf(1, '%3.1f degrees in compass xy plane\n', atan2(pos2(2),pos2(1))*180/pi);
fprintf(1, 'xyz est from pairs %g+%g: (%2.3f, %2.3f, %2.3f) meters, ', pairA, pairC, pos3);
fprintf(1, '%3.1f degrees in compass xy plane\n', atan2(pos3(2),pos3(1))*180/pi);

% average position estimates. in the future, a heuristic might be used to
% pick the best position estimate instead of averaging
position_3d = (pos1+pos2+pos3)./3;                                         50

%%% Subfunction: postion_solve_1 %%%
% subfunction for getting the (x,y,z) position vector of a beacon given the
% phase measurements, a distance estimate, and the two sensor pairs to use
function [orientation_vector] = position_solve_1(phases, distance, pair0, pair1)

speed_sound = 344;

% define sensor array geometry. each row corresponds to x,y,z position of
% sensor# where #=row. sensor3 lies at the origin. conversion from circuit  60
% board layout inter-hole spacing is 4 units = 2.54mm
sensor= [-14 -8 0; 14 -8 0; 0 0 0; 0 16 0] .* (2.54e-3 / 4);

% create direction vectors for each sensor pair
for i = 1:3
    dir_vector(i,:) = sensor((i+1),:) - sensor(1,:);       % 1-2, 1-3, 1-4
end
for i = 4:5
    dir_vector(i,:) = sensor((i-1),:) - sensor(2,:);       % 2-3, 2-4
end                                                                        70
```

```
dir_vector(6,:) = sensor(4,:) - sensor(3,:);                % 3-4

% calculate separation distances and directio unit vectors
for i = 1:6
    dir_vector_norm(i,:) = dir_vector(i,:) ./ norm(dir_vector(i,:));
    separation(i,1) = norm(dir_vector(i,:));
end

% calculate beacon position using two pairs of sensors:

% 0. get the actual sensor numbers that constitute each pair
[i0, j0] = select_sensors(pair0);
[i1, j1] = select_sensors(pair1);

% 1. find the projection of the vector to the beacon onto the direction
% vectors of the sensor pairs (using the differential distance):
proj0 = -dir_vector_norm(pair0,:) .* (distance * (phases(i0, j0) * ...
        speed_sound / separation(pair0, 1)));

proj1 = -dir_vector_norm(pair1,:) .* (distance * (phases(i1, j1) * ...
        speed_sound / separation(pair1, 1)));

% 2. find the intersection of the lines normal to the projections, passing
% through the endpoint of the projection.
p1 = proj0;                         % endpoint of proj is one point on line
p2 = [ -proj0(2) proj0(1) 0];       % vec(-y, x) perpendicular to vec(x, y)
p2 = p2 + p1;                        % find other point on line

p3 = proj1;
p4 = [ -proj1(2) proj1(1) 0];
p4 = p4 + p3;

% points determine vectors; find u1, the scaling factor to intersect them
u1 = ( ( (p4(1)-p3(1))*(p1(2)-p3(2)) - (p4(2)-p3(2))*(p1(1)-p3(1)) ) / ...
       ( (p4(2)-p3(2))*(p2(1)-p1(1)) - (p4(1)-p3(1))*(p2(2)-p1(2)) ) );

x_calc = p1(1) + u1*(p2(1) - p1(1));
y_calc = p1(2) + u1*(p2(2) - p1(2));
z_calc = sqrt( distance^2 - (x_calc)^2 - (y_calc)^2  );

orientation_vector = [x_calc y_calc z_calc];
%%% End of subfunction: position_solve_1 %%%

%%% Subfunction: select_sensors %%%
% subfunction for giving back the sensor numbers that correspond to the named pairs
function [i, j] = select_sensors(pair);
lookup_table = [ 1 2; 1 3; 1 4; 2 3; 2 4; 3 4];
i = lookup_table(pair, 1);
j = lookup_table(pair, 2);
%%% End of subfunction: select_sensors %%%
```

```
function [rotation_quat] = rotation_registration(pos1, pos2, listener, beacon1, beacon2)

% ROTATION_REGISTRATION Find quaternion converting Compass coordinates to
% Cricket coordinates.
%
%   [ROTATION_QUAT] = ROTATION_REGISTRATION(POS1, POS2, LISTENER, BEACON1, BEACON2)
%   applies Berthold Horn's algorithm to three points in Compass and
%   Cricket coordinates to find the rotation from Compass coordinates into
%   Cricket coordinates. The three points are two beacons and the Compass.
%
%   POS1 [x y z], beacon1 position in *compass* coordinates
%   POS2 [x y z], beacon2 position in *compass* coordinates
%   LISTENER [x y z], listener position in *cricket* coordinates
%   BEACON1 [x y z], beacon1 position in *cricket* coordinates
%   BEACON2 [x y z], beacon2 position in *cricket* coordinates
```

```
%
%  ROTATION_QUAT [w x y z], quaternion that represents rotation from
%  Compass coordinate system into Cricket coordinate system.
%
% Package: Cricket Compass MATLAB
% Author:  Kevin J Wang, kevin1@graphics.csail.mit.edu
% Date:    2004/05/23

% by definition, compass/listener position in compass coords is (0,0,0)
compass_pos = [0 0 0];

% remove the centroids from the measurements
compass_centroid = (compass_pos + pos1 + pos2)./3;
compass_pos = compass_pos - compass_centroid;
pos1 = pos1 - compass_centroid;
pos2 = pos2 - compass_centroid;

cricket_centroid = (listener + beacon1 + beacon2)./3;
listener = listener - cricket_centroid;
beacon1 = beacon1 - cricket_centroid;
beacon2 = beacon2 - cricket_centroid;

%%% calculate rotation to register the two planes

%1. find normal vectors, and unit normals:
n_compass = cross(pos2, pos1);
n_cricket = cross(beacon2, beacon1);
n_compass_hat = n_compass ./ norm(n_compass);
n_cricket_hat = n_cricket ./ norm(n_cricket);

%2. find line of intersection between both planes
a = cross(n_compass, n_cricket);
a_hat = a ./ (norm(a)+1e-12);

%3. calculate phi, the angle to rotate the normals into alignment
cos_phi = dot(n_compass_hat, n_cricket_hat);
sin_phi = norm(cross(n_compass_hat, n_cricket_hat));

% use half-angle formulas to avoid trig functions
cos_phi_2 = sqrt( (1 + cos_phi)/2 );
sin_phi_2 = sin_phi / sqrt(2*(1+cos_phi));

quat_a = [ cos_phi_2 sin_phi_2*a_hat ];
% NOTE: 'quat_a' is in standard [w x y z] quaternion notation

% convert to physics notation [x y z w] to use in quaternion package:
quat_a_working = [ sin_phi_2*a_hat cos_phi_2 ];

% 4. apply first rotation to compass coordinates;
% this brings the plane of compass coordinates into the plane of cricket
% coordinates
compass_pos_rotate1 = qvrot(quat_a_working, compass_pos);
pos1_rotate1 = qvrot(quat_a_working, pos1);
pos2_rotate1 = qvrot(quat_a_working, pos2);

%%% calculate rotation vector within the plane to minimize square of
%%% distances between points

% maximize Ccos(theta) + Ssin(theta) by computing maximums for sin(theta)
% based on S and C

C = dot(compass_pos_rotate1, listener) + dot(pos1_rotate1, beacon1) + dot(pos2_rotate1, beacon2);
S = dot((cross(listener, compass_pos_rotate1) + cross(beacon1, pos1_rotate1) + ...
        cross(beacon2, pos2_rotate1)), n_cricket_hat);

sin_theta = S / sqrt(S^2 + C^2);
cos_theta = C / sqrt(S^2 + C^2);
```

```
cos_theta_2 = sqrt( (1 + cos_theta)/2 );
sin_theta_2 = sin_theta / sqrt(2*(1+cos_theta));

quat_p = [ cos_theta_2 -sin_theta_2*n_cricket_hat ];
quat_p_working = [ -sin_theta_2*n_cricket_hat cos_theta_2 ];

% find the overall rotation by multiplying the two quaternions
quat_working = qmult(quat_p_working, quat_a_working);
rotation_quat = [ quat_working(4) quat_working(1:3) ];
fprintf(1, 'Compass to Cricket quaternion: (%2.2f, %2.2fi + %2.2fj + %2.2fk)\n', ...
        quat_working(4), quat_working(1), quat_working(2), quat_working(3));
```

```
function [wave1, wave2, wave3, wave4, time_increment] = tds_sample()

% TDS_SAMPLE Acquire TDS2024 oscilloscope waveforms.
%
%  [WAVE1, WAVE2, WAVE3, WAVE4, TIME_INCREMENT] = TDS_SAMPLE
%  puts the oscilloscope into waveform single sequence acquire mode and
%  captures the waveforms on the four channels as soon as a signal
%  triggers the scope.
%
%  WAVE1, WAVE2, WAVE3, WAVE4 are 2500 sample vectors containing the
%  voltage values of the captured waveforms.
%
%  TIME_INCREMENT is the time increment in seconds between each sample.
%
%  This function requires the MATLAB Instrument Control Toolbox.
%  This function assumes that each channel is set at the same
%  voltage/division settings. It uses the channel 1 volt/div setting.
%
%  Oscilloscope:    Tektronix TDS2024 with TDS2CMAX comm module
%  Connection:      RS-232 cable to host computer on COM2
%  Scope Settings: Baud Rate:     19,200
%                  Flow Control:  Hardware Flagging
%                  EOL String:    LF
%                  Parity:        None
%
%  For scope programming reference, see the TDS2000-Series Programmer
%  Manual.
%
% Package: Cricket Compass MATLAB
% Author:  Kevin J Wang, kevin1@graphics.csail.mit.edu
% Date:    2004/05/23

tic;
fprintf(1, '\n*** Sampling waveforms on TDS2024 ***\n');

% initialize scope object
scope = serial('COM2');
set(scope, 'BaudRate', 19200);
set(scope, 'InputBufferSize', 100000);
set(scope, 'FlowControl', 'hardware');

fopen(scope);
check_errors(scope);

% set waveform capture parameters
fprintf(scope, 'DAT:WID 1');    % set data width to 1
fprintf(scope, 'DAT:ENC RIB'); % set data encoding: sign int, big Endian

% put the scope into single acquire mode
fprintf(scope, 'ACQ:MOD SAM');
fprintf(scope, 'ACQ:STOPA SEQ');
fprintf(scope, 'ACQ:STATE RUN');
fprintf(scope, '*WAI');
```

```
% get time increment in seconds
fprintf(scope, 'WFMP:XIN?');
time_increment = fscanf(scope, '%E');

% dump binary waveform data from each channel
fprintf(scope, 'DAT:SOU CH1');          %select ch1 as data source                    60
fprintf(scope, 'CURV?');                %get dump from scope
fscanf(scope, '%c', 2);                 %discard '#X' in curve preamble
binsize = fscanf(scope, '%d', 4);       %find out data size
wave1 = fread(scope, binsize, 'int8'); %get binary waveform data
fgets(scope);                           %clear any other data

fprintf(scope, 'DAT:SOU CH2');
fprintf(scope, 'CURV?');
fscanf(scope, '%c', 2);
binsize = fscanf(scope, '%d', 4);                                                     70
wave2 = fread(scope, binsize, 'int8');
fgets(scope);

fprintf(scope, 'DAT:SOU CH3');
fprintf(scope, 'CURV?');
fscanf(scope, '%c', 2);
binsize = fscanf(scope, '%d', 4);
wave3 = fread(scope, binsize, 'int8');
fgets(scope);
                                                                                      80
fprintf(scope, 'DAT:SOU CH4');
fprintf(scope, 'CURV?');
fscanf(scope, '%c', 2);
binsize = fscanf(scope, '%d', 4);
wave4 = fread(scope, binsize, 'int8');
fgets(scope);

% get waveform capture parameters to convert Y bit values into voltages
Ymultiple = query(scope, 'WFMP:CH1:YMU?', '%s\n', '%E');
Yoffset = query(scope, 'WFMP:CH1:YOF?', '%s\n', '%E');                                90
Yzero = query(scope, 'WFMP:CH1:YZE?', '%s\n', '%E');

% convert the bit values into voltage values
wave1 = ((wave1 - Yoffset) .* Ymultiple) + Yzero;
wave2 = ((wave2 - Yoffset) .* Ymultiple) + Yzero;
wave3 = ((wave3 - Yoffset) .* Ymultiple) + Yzero;
wave4 = ((wave4 - Yoffset) .* Ymultiple) + Yzero;

% cleanup
check_errors(scope);                                                                  100
fclose(scope);
delete(scope);
clear scope;

% output messages to matlab command window
fprintf(1, 'time increment is: %g microseconds\n', time_increment*1e6);
toc_sample = toc;
fprintf(1, 'sample and transfer completed in: %g seconds\n', toc_sample);

% check_error(s) sub-function checks for msgs then clears them from queue              110
function check_errors(s)
    fprintf(s, '*esr?');
    esr = fscanf(s);
    if (esr ~= '0')     % if the esr register indicates a message,
        fprintf(s, 'allev?');
        allev = fscanf(s);
        fprintf(1, 'scope says: %s', allev);
    end

%%% utility functions %%%                                                            120
% synchronization utility
```

```
    %fprintf(scope, '*OPC?');
    %fscanf(scope);

% verify scope connectivity:
    %fprintf(scope, 'ID?');
    %scopeID = fscanf(scope)
```

```
function [phases] = US_correlate(w1, w2, w3, w4, increment, corr_plot_on);

% US_CORRELATE Find the phase difference between four ultrasound waves.
%
%  [PHASES] = US_CORRELATE(W1, W2, W3, W4, INCREMENT, CORR_PLOT_ON)
%  finds the relative phase differences between four
%
%  W1, W2, W3, W4 are vectors containing the sample values of waveforms.
%  INCREMENT is the time increment in seconds between samples.
%  CORR_PLOT_ON is a boolean turning plots of the cross correlation
%      curves on and off. Use true to turn plotting on.
%
%  PHASES is a 4x4 matrix of measured phase differentials of the form:
%  PHASES = [    0   t12   t13   t14; ...
%              -t12     0   t23   t24; ...
%              -t13  -t23     0   t34; ...
%              -t14  -t24  -t34     0 ];
%  where txy is the phase difference from sensor x to sensor y.
%
% See also ARRAY_GRAPH_PLOT, US_TIME_SHIFT.
%
% Package: Cricket Compass MATLAB
% Author:  Kevin J Wang, kevin1@graphics.csail.mit.edu
% Date:    2004/05/23

fprintf(1, '\n*** Measuring phase differentials ***\n');

% set correlation/plot parameters
speed_sound = 344;

periods = 10.239e-3/ (speed_sound/40e3) * 2; % convert physical sensor
xmin = -25e-6 * periods / 2;                 % spacing to constrain
xmax = -xmin;                                % time delay search space
ymin = -1;
ymax = 1;

periods2 = 17.78e-3/ (speed_sound/40e3) * 2;
xmin2 = -25e-6 * periods2 / 2;
xmax2 = -xmin2;

% if display is on, plot the six correlation curves
if corr_plot_on
    corr_plot_fig = figure;
    set(corr_plot_fig, 'Name', 'Shift+Correlate');
    max_corr_marker = 'rx';
end

fprintf(1, 'Correlating: ');
[T12, R12, t12, c12] = US_time_shift(w1, w2, increment, periods2);
if corr_plot_on
    subplot(2,3,1), plot(T12, R12), title('sensor pair 1-2', 'FontSize', 7);
    format_corr_plot(xmin2, xmax2, ymin, ymax);
    ylabel('normalized cross correlation');
    hold on; plot(t12, c12, max_corr_marker);
end
fprintf(1, '1-2 '); % indicate in command window that search finished

[T13, R13, t13, c13] = US_time_shift(w1, w3, increment, periods);
if corr_plot_on
```

```
    subplot(2,3,2), plot(T13, R13), title('sensor pair 1-3', 'FontSize', 7);
    format_corr_plot(xmin, xmax, ymin, ymax);
    xlabel('wave2 time shift (\mus)');
    hold on; plot(t13, c13, max_corr_marker);
end
fprintf(1, '1-3 ');

[T14, R14, t14, c14] = US_time_shift(w1, w4, increment, periods2);
if corr_plot_on
    subplot(2,3,3), plot(T14, R14), title('sensor pair 1-4', 'FontSize', 7);
    format_corr_plot(xmin2, xmax2, ymin, ymax);
    hold on; plot(t14, c14, max_corr_marker);
end
fprintf(1, '1-4 ');

[T23, R23, t23, c23] = US_time_shift(w2, w3, increment, periods);
if corr_plot_on
    subplot(2,3,4); plot(T23, R23), title('sensor pair 2-3', 'FontSize', 7);
    format_corr_plot(xmin, xmax, ymin, ymax);
    ylabel('normalized cross correlation');
    hold on; plot(t23, c23, max_corr_marker);
end
fprintf(1, '2-3 ');

[T24, R24, t24, c24] = US_time_shift(w2, w4, increment, periods2);
if corr_plot_on
    subplot(2,3,5), plot(T24, R24), title('sensor pair 2-4', 'FontSize', 7);
    format_corr_plot(xmin2, xmax2, ymin, ymax);
    xlabel('wave2 time shift (\mus)');
    hold on; plot(t24, c24, max_corr_marker);
end
fprintf(1, '2-4 ');

[T34, R34, t34, c34] = US_time_shift(w3, w4, increment, periods);
if corr_plot_on
    subplot(2,3,6), plot(T34, R34), title('sensor pair 3-4', 'FontSize', 7);
    format_corr_plot(xmin, xmax, ymin, ymax);
    hold on; plot(t34, c34, max_corr_marker);
end
fprintf(1, '3-4\n');

% output recorded phase information
phases  = [    0   t12   t13   t14; ...
             −t12     0   t23   t24; ...
             −t13  −t23     0   t34; ...
             −t14  −t24  −t34     0 ];
phases = phases'; % flip-flop because of change in notation

% sub-function for setting up correlation curve plot formatting options
function format_corr_plot(xmin, xmax, ymin, ymax)
axis([xmin xmax ymin ymax]);
set(gca, 'FontSize', 7);
set(gca, 'XTick', [−50 −25 0 25 50].*1e−6);
set(gca, 'XTickLabel', '-50|-25|0|25|50');
set(gca, 'YTick', [−1 −0.5 0 0.5 1]);
```

```
function [T, R, at_t, at_c] = US_time_shift(wave1, wave2, increment, periods)

% US_TIME_SHIFT Find the time delay between two ultrasound waveforms.
%
%  [T, R, AT_T, AT_C] = US_TIME_SHIFT(WAVE1, WAVE2, INCREMENT, PERIODS)
%  finds the time shift of WAVE2 that maximally correlates WAVE2 with
%  WAVE1.
%
%  WAVE1, WAVE2 are vectors containing the sample values of two waveforms.
%  INCREMENT is the time increment in seconds between samples.
```

```
%  PERIODS specifies the maximum sample delay to shift the waveforms in
%       ultrasound periods. i.e. if periods=2, I shift WAVE2 over a range
%       [-one period, +one period].
%
%  T is the time vector representing the range of time shifts searched.
%  R is the correlation vector representing the correlation of the two
%       waveforms at each time shift. A plot of R vs. T produces the
%       correlation curve.
%  AT_T is the time shift of WAVE2 that maximally correlates WAVE2 with
%       WAVE1.                                                              20
%  AT_C is the maximum normalized cross correlation between WAVE1 and
%       WAVE1, which occurs for the time shift AT_T applied to WAVE2.
%
% Package: Cricket Compass MATLAB
% Author:  Kevin J Wang, kevin1@graphics.csail.mit.edu
% Date:    2004/05/23

% shift forward by half of the desired # of periods; shift backward by the
% other half
shifts = floor(periods*25e-6/increment)+1;                                  30
half_shifts = shifts/2;

% do correlation for no shift
temp_R = corrcoef(wave1, wave2);
R = temp_R(2, 1);

% find correlation for each sample shift *forward* of wave2
R_fwd = [];
for i = 1:(half_shifts)
    temp_R = corrcoef(US_zero_fwd(wave1, i), US_shift_fwd(wave2, i));       40
    R_fwd = [ R_fwd; temp_R(2, 1) ];
end

% find correlation for each sample shift *backward* of wave2
R_bwd = [];
for i = 1:(half_shifts)
    temp_R = corrcoef(US_zero_bwd(wave1, i), US_shift_bwd(wave2, i));
    R_bwd = [ R_bwd; temp_R(2, 1) ];
end
                                                                            50
R = [flipud(R_bwd); R; R_fwd];
temp_t = create_tvector(half_shifts, increment);
[at_t, at_c] = max_shift_index(temp_t, R);
T = temp_t;

% subfunction: shifts a US wave by INCR samples forward
function a = US_shift_fwd(input, incr)
z = zeros(incr);
z = z(1:incr, 1);
a = [ z; input(1:(length(input)-incr), 1) ];                                60

% subfunction: shifts a US wave by INCR samples backward
function b = US_shift_bwd(input, incr)
z = zeros(incr);
z = z(1:incr, 1);
b = [ input((1+incr):length(input), 1); z ];

% subfunction: zero out a US wave by INCR samples forward
function a = US_zero_fwd(input, incr)
input(1:incr, 1) = 0;                                                       70
a = input;

% subfunction: zero out a US wave by INCR samples backward
function b = US_zero_bwd(input, incr)
input((length(input)-incr+1):length(input), 1) = 0;
b = input;

% subfunction: creates the time index vector
```

```
function t = create_tvector(half_shifts, incr)
temp_t1 = [1:half_shifts];
temp_t2 = fliplr(temp_t1).*(-1);
t = [temp_t2 0 temp_t1].*incr;

% subfunction: outputs maximum correlated time shift using inputs
% of index time vector and correlations
function [t_cmax, cmax] = max_shift_index(t_vector, c_vector)
[cmax, I] = max(c_vector);
t_cmax = t_vector(1, I);
```

```
function [pairA, pairB, pairC] = voting(phases, plot_on);

% VOTING Pick 3 accurate sensor pairs.
%
%  [PAIRA, PAIRB, PAIRC] = VOTING(PHASES, PLOT_ON) uses the sign of the
%  phase measurements between sensors to localize a beacon to a 30 degree
%  sector. It then returns the three sensor pairs that are accurate for
%  localizing the beacon precisely in that sector.
%
%  PAIRA, PAIRB, PAIRC are the 3 accurate sensor pairs. There are six
%  sensor pairs on the Compass array, numbered 1, 2, 3, 4, 5, 6.
%
%  PHASES is a 4x4 matrix of measured phase differentials of the form:
%  PHASES = [    0   t12   t13   t14; ...
%              -t12     0   t23   t24; ...
%              -t13 -t23     0   t34; ...
%              -t14 -t24 -t34     0 ];
%  where txy is the phase difference from sensor x to sensor y.
%  PLOT_ON is a boolean turning the compass plot of the sensor voting
%       on and off. Use true to turn plotting on.
%
%  This function uses the predefined sensor array geometry.
%
% See also POSITION_SOLVER, US_CORRELATE, EXPECTED_PHASE.
%
% Package: Cricket Compass MATLAB
% Author:  Kevin J Wang, kevin1@graphics.csail.mit.edu
% Date:    2004/05/23

votes = repmat([0], 1, 360);    % a vector to hold the votes

% For each pair of sensors, use the sign to localize the beacon position to
% one side of the array, which is a 180 degree range.
% The voting is messy because the degree ranges must wrap around at 0=360
% degrees.
if (phases(1,2) < 0)
    votes(1,   1: 90) = votes(1,   1: 90) + 1;
    votes(1, 272:360) = votes(1, 272:360) + 1;
else
    votes(1,  92:270) = votes(1,  92:270) + 1;
end

if (phases(1,3) < 0)
    votes(1,   1:120) = votes(1,   1:120) + 1;
    votes(1, 302:360) = votes(1, 302:360) + 1;
else
    votes(1, 122:300) = votes(1, 122:300) + 1;
end

if (phases(1,4) < 0)
    votes(1,   1:150) = votes(1,   1:150) + 1;
    votes(1, 332:360) = votes(1, 332:360) + 1;
else
    votes(1, 152:330) = votes(1, 152:330) + 1;
end
```

```
if (phases(3,4) < 0)
    votes(1,   2:180) = votes(1,   2:180) + 1;
else
    votes(1, 182:360) = votes(1, 182:360) + 1;
end

if (phases(2,4) < 0)
    votes(1,  32:210) = votes(1,  32:210) + 1;
else
    votes(1, 212:360) = votes(1, 212:360) + 1;
    votes(1,   1: 30) = votes(1,   1: 30) + 1;
end

if (phases(2,3) < 0)
    votes(1,  62:240) = votes(1,  62:240) + 1;
else
    votes(1, 242:360) = votes(1, 242:360) + 1;
    votes(1,   1: 60) = votes(1,   1: 60) + 1;
end

if plot_on
    radians = [ 0:1:359 ].*(pi/180);
    [x, y] = pol2cart(radians, votes);
    compass(x, y); % use a compass plot to generate the radial shape
end

% I want to avoid edge conditions where the 30 degree sectors overlap. I
% look at increments of every 5 degrees and use the first increment off the
% center of each sector to label that sector. For example, the sector [345,
% 15] is labeled as 5 and [75, 105] is labeled as 95, etc.
degrees = [ 0:1:359 ];
shift_degrees = degrees(1, 5:360);          % take every fifth degree
shift_votes = votes(1, 5:360);              %

select_degrees = shift_degrees(2:30:356);   % extract the degree label for
select_votes = shift_votes(2:30:356);       % the thirty degree sectors

[C, I] = max(select_votes);                 % find the sector with the
max_degree = select_degrees(1, I);          % largest number of votes

% now that I have localized the 30 degree sector, I essentially do a table
% lookup to return the three sensor pairs which are accurate in that sector
switch (max_degree)
    case {5, 185}
        pairA = 3; pairB = 5; pairC = 6;
    case {35, 215}
        pairA = 4; pairB = 5; pairC = 6;
    case {65, 245}
        pairA = 1; pairB = 4; pairC = 5;
    case {95, 275}
        pairA = 1; pairB = 2; pairC = 4;
    case {125, 305}
        pairA = 1; pairB = 2; pairC = 3;
    case {155, 335}
        pairA = 2; pairB = 3; pairC = 6;
    otherwise
        fprintf(1, 'Voting algorithm failed!\n');
end
fprintf(1, 'pairs selected by vote: %g %g %g \n', pairA, pairB, pairC);
```

Bibliography

[1] Ascension Technology. http://www.ascension-tech.com/, 2004.

[2] Northern Digital Inc. - Aurora. http://www.ndigital.com/aurora.html, 2004.

[3] E. Foxlin, M. Harrington, and G. Pfeiffer. Constellation: A Wide-Range Wireless Motion-Tracking System for Augmented Reality and Virtual Set Applications. In *Proc. ACM SIGGRAPH*, Orlando, FL, July 1998.

[4] J. Gray. Olinde Rodrigues' paper of 1840 on transformation groups. *Archive for History of Exact Sciences*, 21:375–385, 1980.

[5] Berthold K. P. Horn. Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America A*, 4, April 1987.

[6] Polhemus - Products. http://www.polhemus.com/Products.htm, 2004.

[7] Nissanka B. Priyantha, Allen K. L. Miu, Hari Balakrishnan, and Seth J. Teller. The Cricket Compass for Context-Aware Mobile Applications. In *Mobile Computing and Networking*, pages 1–14, 2001.

[8] Seth Teller, Jiawen Chen, and Hari Balakrishnan. Pervasive pose-aware applications and infrastructure. In *IEEE Computer Graphics and Applications*, pages 14–18, July/August 2003.

[9] Nicholas Vallidis. *WHISPER: A Spread Spectrum Approach to Occlusion in Acoustic Tracking*. PhD thesis, University of North Carolina at Chapel Hill, Department of Computer Science, 2002.

[10] Greg Welch, Gary Bishop, Leandra Vicci, Stephen Brumback, Kurtis Keller, and D'nardo Colucci. The HiBall tracker: High-performance wide-area tracking for vir-

tual and augmented environments. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, December 1999.

[11] Greg Welch and Eric Foxlin. Motion Tracking: No Silver Bullet, but a Respectable Arsenal. In *IEEE Computer Graphics and Applications, special issue on "Tracking"*, pages 34–38, November/December 2002.