Rapid Coordinate System Creation and Mapping Using Crickets

by

Roshan Bantwal Baliga

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
June 4, 2004

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

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

Rapid Coordinate System Creation and Mapping Using Crickets

by

Roshan Bantwal Baliga

Submitted to the Department of Electrical Engineering and Computer Science
on June 4, 2004, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

In this thesis, I describe a system that lays the foundation for context-aware applications. This system allows a user to set up a reference coordinate system in a room, using three Cricket *listeners* attached to a wooden frame. The system then assigns coordinates to Cricket *beacons*, which are placed on the ceiling. Finally, by using the frame in conjunction with a laser range finder, the user can generate a map of the room in the reference coordinate system, complete with features including doors, walls, and windows. This thesis also describes necessary changes we implemented that made the Cricket positioning system much more accurate.

Thesis Supervisor: Seth Teller
Title: Associate Professor

Acknowledgments

I would like to foremost thank my advisor, Seth Teller, who could always answer the questions that had been bothering me for days within three tries, or ten minutes, whichever came first. I guess that's why he's the professor and I'm the graduate student! I would also like to thank Professor Hari Balakrishnan, who kept me on track during the past three semesters, and helped me analyze engineering problems from different viewpoints.

I am grateful to Bodhi Priyantha for designing the Cricket hardware and the AFL algorithm, and to Michel Goraczko for writing the software for the new Crickets. I am also grateful to David Moore for developing the Robust Quadrilateral algorithm, as well as his numerous embedded Cricket applications. Of course, none of this would have been possible without the support of the entire Cricket Team, including Dorothy Curtis, Allen Miu, and Ken Steele.

I would like to acknowledge Kevin Chen who wrote the source code for visualizing Crickets in his "Software Flashlight" application, which I recycled many parts of for my project. I also owe much thanks to Fred Cote for helping me with all the equipment in the Edgerton machine shop.

The two people who kept me sane during the past year are my officemates, Kevin Wang and Jonathan Wolfe. Thanks to Kevin for his encouragement and help with everything Cricket related, and thanks to Jon for providing Kevin and me with great stories.

I would like to dedicate this thesis to my family for their unconditional support.

Contents

List of Figures

List of Tables

Chapter 1

Motivation and Background

1.1 Introduction

In this thesis, I describe a system that lays the foundation for context-aware applications, using the Cricket location architecture. Cricket allows users to determine their 3D coordinates within a building, much like GPS does outdoors. However, these coordinates are only useful if they come in the context of a larger framework, such as a model of the building in that coordinate system. Additionally, when initially deploying the Cricket system within a building, one must determine the 3D coordinates of all the Cricket sensors.

The goal of this system was to enable a user to generate a map of a room, using a cartesian coordinate system, which includes locations of walls, doors, and windows, as well as locations of all the Cricket sensors in the room. Specifically, this system allows a user to set up a reference coordinate system in a room, using three Cricket *listeners* attached to a wooden frame. We call this reference coordinate system the "rest frame" of the room. Figure 1-1 shows how a rigid frame determines this coordinate system for the room. By using only three distance measurements to points on the frame from an object, we can determine the object's 3D coordinate.

In this manner, the system then assigns coordinates to Cricket *beacons*, which are placed on the ceiling. Finally, by using the frame in conjunction with a laser range finder, the user can generate a map of the room in the reference coordinate system, complete with features including doors, walls, and windows. This thesis also describes necessary changes we implemented that makes the current Cricket positioning system much more accurate than its predecessors.

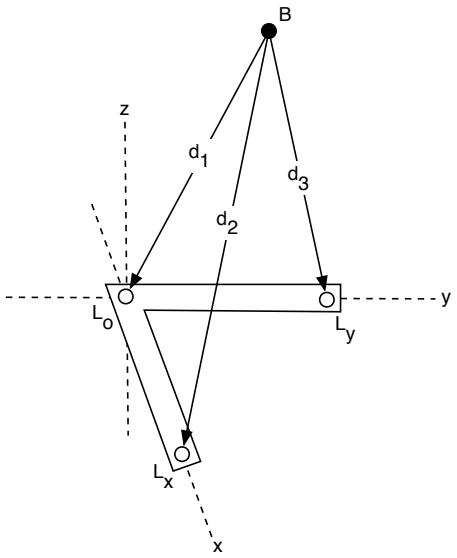

Figure 1-1: The frame can be used to assign coordinates to any object, using only distance measurements from the object to three points on the frame.

The contributions of this thesis include:

- A frame for three Cricket listeners, which creates a consistent 3D coordinate system within a room. The user may assign coordinates to anything in the room in reference to the placement of this frame.
- *WallsDemo3*, a piece of software that displays the frame, as well as the beacons, in the rest frame of the room. This software also allows the user to generate a model of the room in that coordinate system, by marking features of the room, including doors, walls, and windows.
- Many improvements to the current version of the Cricket positioning system, dramatically improving its accuracy.

1.2 Overview of the Cricket Indoor Location System

Cricket is a location architecture that allows sensors, laptops, handheld computers, and other devices to know about their current location. Location information is a necessary prerequisite for context-aware applications in pervasive computing, such as those being developed by MIT's Project Oxygen. Outdoors, Cricket uses the Department of Defense's

Global Positioning System (GPS), but indoors, where GPS signals are unreliable, Cricket uses a combination of RF and ultrasound to determine position.

One of the goals of the Cricket project is to develop a pervasive internal spatial localization system so that objects can be tagged, either physically or virtually, with data on their current location. This data may be information about the type of room the object is in, or the object's position within the building. Another goal is to provide real-time location information to all mobile devices, which enables personalization of applications based on location, and allows for applications that give users directions to real-world sites. Knowledge about location is imperative for effective pervasive computing.

The Cricket system relies on beacons and listeners. Beacons are mounted in rooms and broadcast information used by passive listeners to determine their respective locations. If the beacons know their locations relative to each other, then a listener can determine its location with respect to the beacons, as shown in Figure 1-2. However, for many applications, the listener needs to know its location with respect to a room, building, or the earth. For the listener to determine its latitude and longitude, the beacons need to know their respective latitudes and longitudes. In the same manner, when an application calls for an (x, y, z) building coordinate, the beacons must each know their (x, y, z) building coordinate.[1]

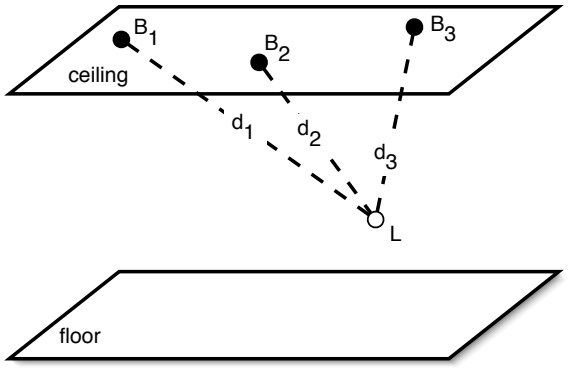

Figure 1-2: A listener (white circle) trilaterates its position using distance measurements from nearby beacons (black circles).

[1]In this thesis, when I refer to absolute position, I mean the position of a beacon or listener with respect to an external coordinate system, such as building coordinates, or "earth-relative" coordinates (latitude and longitude).

1.3 The Need for Cricket

Cricket provides the core information necessary for applications that rely on location information. If mobile devices know their current location, then applications such as navigation systems become possible. Museums can distribute handheld guides that can provide users with specific, real-time directions to works of art the user is interested in viewing. Visitors to MIT's campus could use handheld computers with navigation software to find their way through unfamiliar buildings.

Cricket-enabled sensors can provide environmental data specific to their location, which is useful for hazardous materials monitoring, or simple indoor climate management. Assets, such as oscilloscopes and laptops, can be tracked within a building, making manual inventory checks obsolete.

Specific locations within buildings can be tagged, so queries from a handheld computer return information related only to the computer's current location. Facilities repairs can be identified by exact location, rather than vague descriptions (e.g. "the northeast wall of 32-331 has a leak").

Cricket also fulfills a prerequisite for pervasive computing; the software needs to understand spatial context. For example, a user's request to print a document should be serviced by the printer nearest to the user. The query "Where are the chips?" has quite a different meaning when the user is in a kitchen, as compared to a research lab.

In a coarse-grained system, even information about what room a device is in is helpful. A Cricket-enabled name tag could route phone calls to the phone nearest to the wearer. However, a fine-grained system, with accuracy down to a cubic centimeter, is necessary for virtual tagging, mapping, navigation, and sensor applications. Current RF identification systems fail to fill this need because they are not pervasive and do not provide accurate location information. The Cricket system fills this need nicely.

1.4 The Need for Beacon Calibration

For Cricket to be successful, the system must be pervasive. Outdoors, this is easy to accomplish, since GPS signals are already available almost everywhere. Indoors, however, Cricket requires an installed beacon infrastructure. While beacons can be installed in any existing building, it is preferable that in the future beacons be installed during a building's

construction. Regardless of when beacons are installed, the installation procedure should be quick and easy if Cricket use is to become widespread.

One can imagine a contractor installing beacons onto a room's ceiling while measuring out distances to walls using a tape measure. The contractor would then manually program each beacon with its absolute location within the building. This simple process, while providing the information necessary for beacons to function, would be tedious for anyone installing more than a few beacons.

Instead, it is preferable to have a system by which the beacons are assigned coordinates rapidly. This system must also generate a context for this coordinate system, either by assigning coordinates to the beacons that are consistent with building coordinates given on a CAD drawing, or by generating a map of the building as beacons are installed.

The system I describe in this thesis uses the latter method, and allows the end user to add building features to the map, including the locations of doors, walls, and windows. A beacon calibration system must also create a coordinate system that is not only consistent among all the beacons, but also among the building coordinates or building plan. By using a coordinate system with a known real-world scale, it becomes possible to register the generated map with a CAD drawing of the building.

1.5 Related Work

Active Bats, a project at AT&T Cambridge, also uses RF and ultrasound to determine locational coordinates within a building [9]. Due to the design choice of having mobile beacons and fixed listeners (on the ceiling), the Bat system requires listeners to be mounted in a very precise grid pattern. This requirement does not facilitate easy installation; if self-calibration in the Cricket system works as expected, the installation of Cricket beacons should be much easier than the installation of Active Bat listeners.

Many projects attempt to use RF propagation characteristics to determine location. One such system is RADAR, developed at Microsoft Research in 2000 [1]. The RADAR system compares received signal strength (expressed in dBm) and the signal-to-noise ratio (expressed in dB) from three 802.11 transmitters to triangulate position. Since RF propagation is difficult to predict, prior to using RADAR the area being covered must be "mapped-out" by measuring the received signal at various locations within the area. This

offline calibration requires significant human effort and is unreliable over time. RF propagation characteristics within a room change constantly due to human interference, as well as interference from electrical, motor, or RF systems in the area.

Recent work has improved the accuracy of RF localization systems; 802.11-based systems now claim accuracy to 7 feet [10]. However, these advances have not changed the offline mapping prerequisite for using these systems.

Chapter 2

Approach

2.1 Accuracy

Traditional handheld GPS receivers can pinpoint locations to an accuracy of ±20 meters [8]. While this level of accuracy is acceptable for most outdoor applications, an indoor location system should be much finer-grained. With Cricket, we have attempted to provide location information with an approximate uncertainty of a cubic centimeter. The first version of the Cricket system could measure distances only with an accuracy of ±15 cm. Since this level of error was unsuitable for most of our intended applications, we first worked to bring subsequent revisions of the hardware and software up to an acceptable performance level.

The basic theory behind the Cricket system has remained unchanged for the last few years. Cricket works using trilateration, in a manner similar to GPS. After a listener determines the distances to three fixed beacons, and receives the beacons' relative coordinates in a consistent coordinate system, the listener can use trilateration to determine its own coordinates in that system.

To measure the distance to GPS satellites, a GPS receiver measures the time it takes for RF signals to get to the receiver from each satellite. Using a known value for the speed of RF signal propagation (generally assumed to be the speed of light), the receiver can calculate the distance to each satellite.[1] Since the satellites orbit approximately 20,000 km above the earth's surface, the RF signals take at least 66μs to reach the receiver, which is not difficult for a receiver to time accurately [6]. The method used in GPS requires the use

[1]This explanation of GPS is oversimplified. In reality, GPS receivers must use clever tricks to work around their lack of atomic clocks. For a more detailed explanation of how GPS works, please see Parkinson, B. W., et al, *Global Positioning System: Theory and Applications* [6].

of very precise atomic clocks onboard the orbiting satellites.

Since Cricket works indoors over much shorter distances, it is not feasible to measure the RF time-of-flight. Instead, Cricket uses a combination of RF and ultrasound to determine the distances between a beacons and listeners. Analagous to GPS, beacons play the role of the satellites, and listeners play the role of the receivers. The crux of the Cricket system is that RF signals take a negligible amount of time to cover the small distance between beacons and listeners (generally less than 10 meters). By exploiting the immense speed difference between sound and RF signals in air, a listener can measure the time-of-flight for an ultrasound pulse to reach it. Assuming a value for the local speed of sound, the listener can then calculate the distance to the beacon which sent the ultrasonic pulse, and then use trilateration to determine its position. A more detailed description of the timing system is given in Figure 2-1.

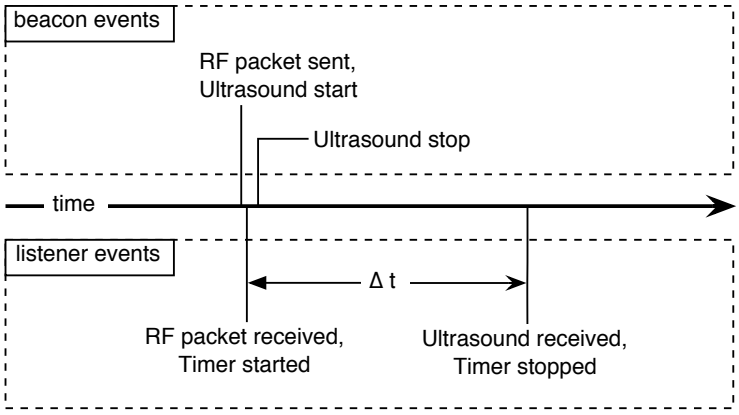

Figure 2-1: Overview of distance measurements in the Cricket system. First, the beacon sends an RF packet with its absolute location (x, y, z coordinate), along with an announcement that it is sending an ultrasonic [US] pulse. When the listener receives the RF packet, it starts an onboard timer. The listener stops the timer when it receives the US pulse, and therefore measures the time-of-flight of the ultrasound pulse. The time-of-flight, along with the local speed of sound, is then used to calculate the distance separating the beacon and listener.

2.1.1 Cricket Version 1 Design

The first version of Cricket was designed at MIT's Laboratory for Computer Science in 1999 by N. Bodhi Priyantha and Anit Chakraborty. It used a radio transmitting in the 418 MHz unlicensed band, and used ultrasound transmitters and receivers made by the

Kobitone audio company. The platform had three limitations that were addressed in subsequent revisions. First, the accuracy was not good enough for the intended applications. Second, it was difficult to develop software to run on the Crickets, largely due to the chosen microprocessor. Finally, the system used different circuit boards for the beacons and listeners, resulting in beacons being strictly limited to generating ultrasonic pulses, and listeners limited to receiving ultrasonic pulses. A Version 1 listener and beacon are shown in Figure 2-2.

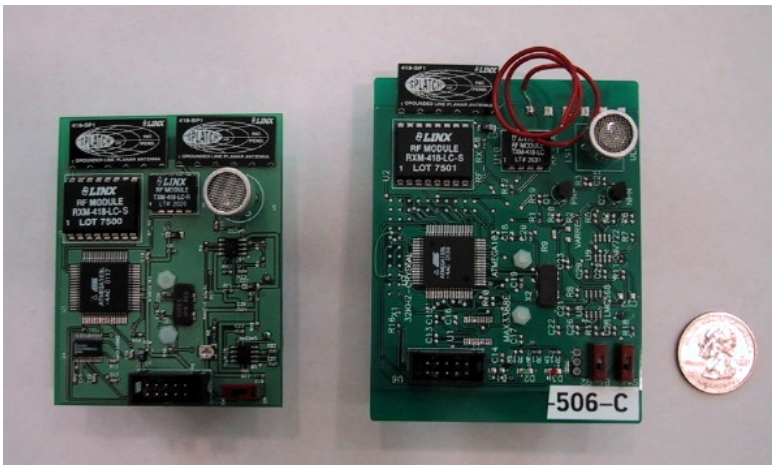

Figure 2-2: Version 1 listener (left) and beacon.

Figures 2-3 and 2-4 show the accuracy of the Version 1 Crickets, using data taken from an experiment in April 2003. The scatter plot in Figures 2-3 shows that most samples lie somewhat close to the ideal measurement, but occasional measurements lie far away from the ideal distance measurements. The CDF in Figure 2-4 shows that even at distances without outliers, the measurements are not terribly accurate.

The primary source of error in the Version 1 Crickets was the part of the listener circuitry that detected an incoming ultrasonic pulse. The detector circuit used a Phase-Lock-Loop, as shown in Figure 2-5, to detect the incoming 40 KHz ultrasonic signal. Unfortunately, the PLL was unreliable, and took a variable amount of time to lock on to the incoming ultrasonic pulse. The PLL error added approximately 500 μs of uncertainly to the time-of-flight measurement, which accounts for most of the error in the distance measurements.

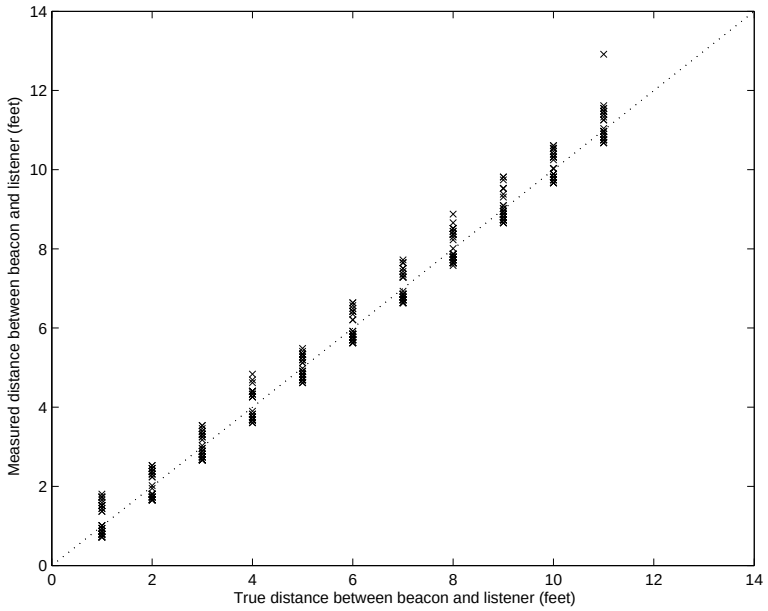

Figure 2-3: Scatterplot of distance measurements between a Version 1 listener and beacon at various separations. The dashed line represents where ideal measurements would lie.

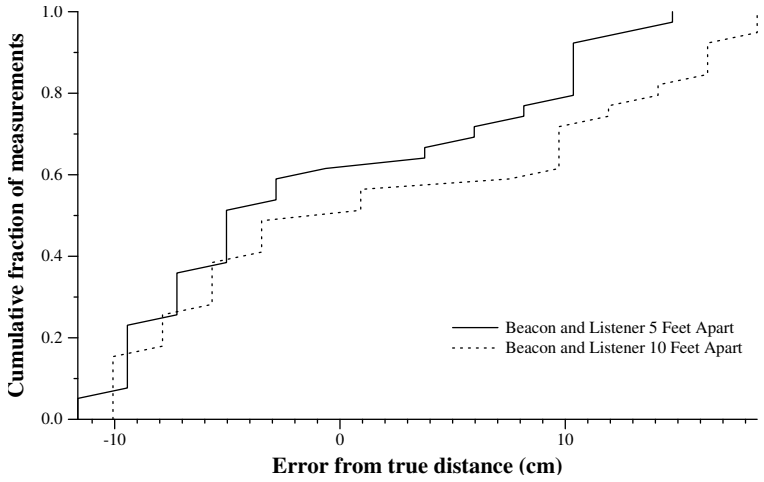

Figure 2-4: CDF of distance measurements between a Version 1 listener and beacon at 5 feet and 10 feet.

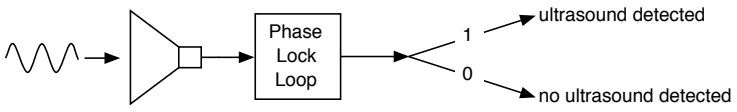

Figure 2-5: Block diagram of the ultrasound detection circuit used in the Version 1 Listener. The output of the Phase-Lock Loop is "1" when the PLL has locked onto the incoming 40 kHz signal, and "0" otherwise.

2.1.2 Cricket Version 2 Design

For the Version 2 Crickets, the primary goal was to get 1-centimeter accuracy with distance measurements, though the other two major shortcomings of the Version 1 Crickets (microprocessor functionality and circuit board differences between listeners and beacons) were also addressed. The V2 Crickets used the same radio and microprocessor used on the Berkeley Mica2 sensors, which allowed us to run Berkeley's TinyOS platform on the Crickets. This open-source operating system already had many users, and allowed us to speed development on the new Crickets. The V2 Crickets only used one circuit board, so beacons and listeners could be identical, except for the software they ran. This allowed us to test autoconfiguration algorithms that were not possible using the V1 Crickets. A Version 2 Cricket is shown in Figure 2-6. Ph.D candidate N. Bodhi Priyantha lead the design of the V2 Cricket.

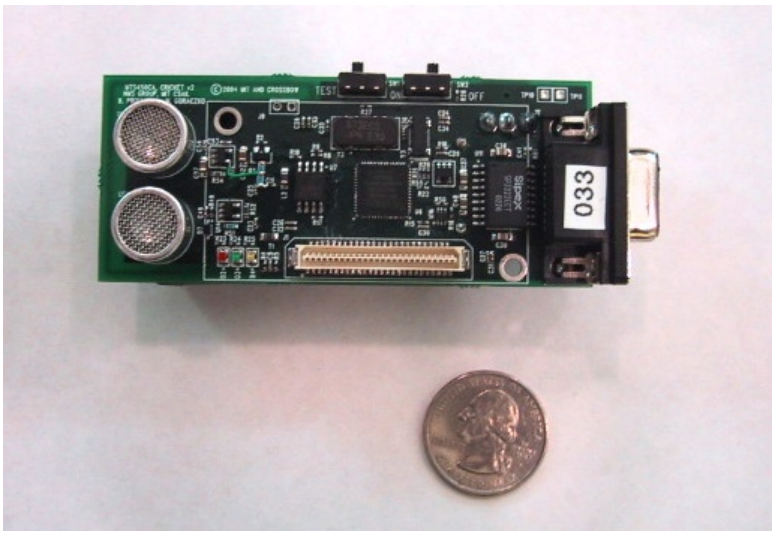

Figure 2-6: Version 2 listener/beacon combination.

Ultrasound Receive Circuit Improvements

The change to the hardware that made the most improvement to accuracy was the replacement of the Phase-Lock-Loop from the ultrasound receive circuit with a dual stage amplifier, followed by a comparator, as shown in Figure 2-7. The response rate of the new amplifier section is limited primarily by t_{tot}, the response time of the op-amps used, and therefore does not have a large variable delay in detecting ultrasonic pulses from the beacons.

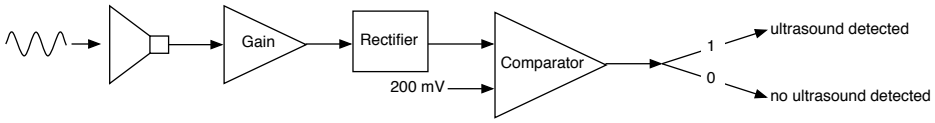

Figure 2-7: Block diagram of the ultrasound detection circuit used in the v2 listener. When the output of the comparator first rises to "1," the microprocessor automatically stops the hardware timer used to measure the ultrasound time-of-flight.

During testing, we did notice minor measurement deviations in the ultrasound receive circuit. Figure 2-8 shows the results of two tests: one with the beacon and listener separated by 30 cm, and another with a 60 cm separation. In each test, the beacon and listener were kept a fixed distance apart, and the output of the comparator was observed for its rising edge. Over multiple trials, we measured the window of time between the earliest and latest comparator output rising edge. This window, shown in the figure as hashed vertical purple lines, represents uncertainty in the measurement of the ultrasound time-of-flight. In each case, the uncertainty was less than one period of ultrasound (25 μs), which corresponds to less than 1 cm of measured distance.

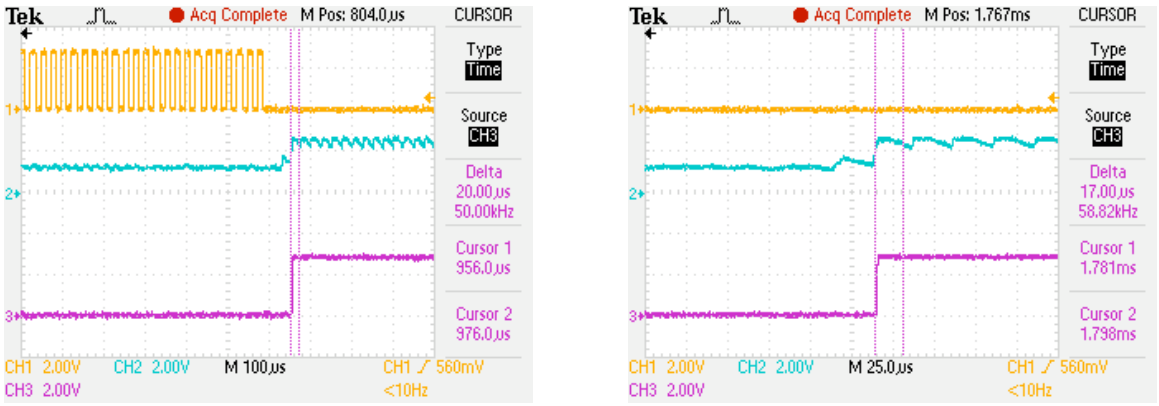

Figure 2-8: Oscilloscope screenshots of signals in the ultrasound receive path. In each screenshot, the orange signal (top) is ultrasound output pin on the beacon microprocessor. The turquoise signal (middle) is the input to the comparator in the ultrasound receive circuitry of the listener, while the purple signal (bottom) is the output of said comparator. For this experiment, a beacon and listener were placed on a table facing each other. The screenshot on the left represents the beacon and listener 30 cm apart, while the image on the right represents the beacon and listener 60 cm apart. The hashed purple vertical bars represent the window of the comparator output switching high over multiple trials.

Peak Detection Circuit

Figure 2-9 shows an example of the received ultrasonic signal by a listener, as measured after amplification, but before rectification. The higher frequency carrier is the 40 kHz ultrasonic signal, while the lower frequency envelope is a side-effect caused by the channel characteristics. It is important to note that while the received signal has a length of over 40 periods, the output circuit drives the transmitter with only a six period square wave. This extension is caused by the resonance of the ultrasound transmitter. The V2 ultrasound receive circuit, as shown in Figure 2-7, first amplifies this received signal, then rectifies the signal, passing only the positive half of each wave. The comparator then essentially acts as a peak detector, signaling whenever the wave exceeds a predetermined threshold (200 mV in this case).

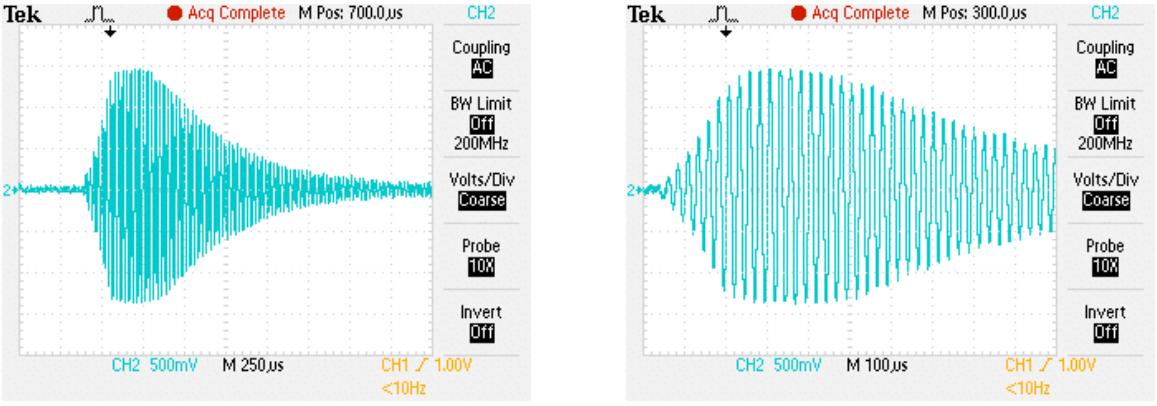

Figure 2-9: The figure on the left shows the ultrasound pulse "envelope," as measured after the cricket listener's amplification stage, but before rectification. The figure on the right shows a closer view of the same signal, so it is easier to see the main 40 kHz signal.

When a beacon is moved further away from a listener, the amplitude of the received ultrasound signal at the listener decreases, as is expected. This fact, when combined with the ultrasound receive circuit, leads to an interesting side effect: quantized error.

As an example, consider a beacon and listener placed n meters apart, and then subsequently placed $n + \Delta n$ meters apart. The first few periods of the received ultrasound waveforms will look like the waveforms shown in Figure 2-10. As shown, when the beacon and listener are placed n meters apart, the comparator threshold is first exceeded by the second peak of the ultrasound waveform. However, when the beacon and listener are placed $n + \Delta n$ meters apart, the comparator threshold is first exceeded by the third peak of the

ultrasound waveform, because the entire waveform has been attenuated by a factor of 0.75 (in this example). The attenuation of the ultrasound waveform is caused by the power loss as the ultrasound traverses a further distance. Since the comparator is "tripped" one period later when the beacon and listener are $n + \Delta n$ meters apart, the measured time-of-flight for the ultrasound has a additional error of about $25\mu s$. If the channel attenuation is quite significant, it may cause the threshold to be exceeded a few periods late, instead of just one period late. However, this error will always be in integer multiples of $\sim 25\mu s$.

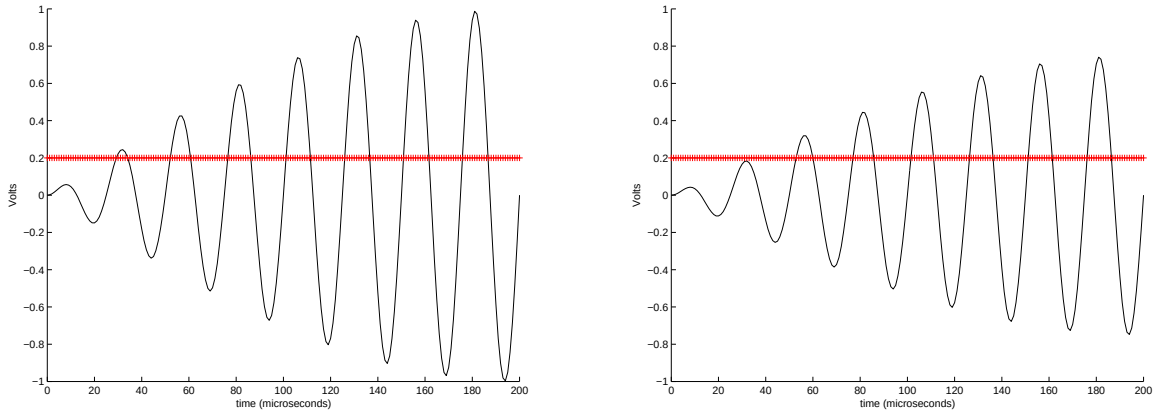

Figure 2-10: Illustration (simulated data) of the beginning of ultrasound pulses as measured after the receiver's amplification stage, but before rectification. In the graph on the right the signal has been attenuated, as compared to the graph on the left, because the beacon and listener are further apart. The red line represents the voltage threshold for the comparator (200 mV). In the graph on the left, the threshold is first exceeded by the second peak of the ultrasound waveform. However, in the graph on the left, due to the channel attenuation, the threshold is first exceeded by the third peak of the ultrasound waveform.

This source of error is directly caused by our method of ultrasound pulse detection. Cricket's accuracy could be improved if the listener could calculate exactly when it received the start of the pulse. Future research should examine pulse shaping and autocorrelation filters, which are techniques used in radar to obtain precise timing of reflected radio waves.

Ultrasound Output Circuit

While the new ultrasound detection circuit improved accuracy by eliminating variable delays, it was not as sensitive as the PLL version used in the V1 Crickets. To compensate for this decreased sensitivity, the V2 crickets included a higher-power ultrasound transmission circuit. Instead of driving the ultrasound transmitter with a 3 V_{pp} square wave, the im-

proved output circuit drove the transmitter with a 12 V_{pp} square wave. The increased power output on the ultrasound channel also improved the signal-to-noise ratio as measured at the receiver. This made the system more resilient to spurious sources of 40 kHz ultrasound noise, such as keys jingling or jackhammering.

Radio Byte Boundaries

The data bus between the radio and the microprocessor introduced another source of error. Luckily, this error was easily fixed, once we had isolated its source. Cricket radio messages are transmitted in packets, whose basic structure is shown in Figure 2-11.[2] As with most RF packets, Cricket's packet structure was designed so that the processor could miss the first few bits of the preamble, but receive and interpret the rest of the packet correctly. The preamble is used to determine byte boundaries for the packet so the processor car correctly decode the header and payload.

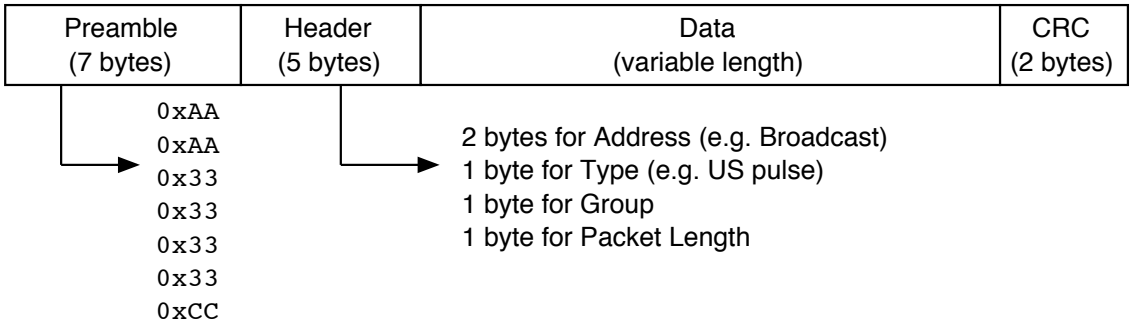

Figure 2-11: The structure of a sample Cricket radio packet. The packet is designed such that the microprocessor can miss the part of the preamble but still decode the packet's data.

Figure 2-12 shows how data is transmitted from the radio to the microprocessor. Bits are sent one at a time down a serial wire to the microprocessor, which caches them in an 8-bit buffer. When the buffer is full, it triggers an interrupt, and the microprocessor's software copies the data byte to memory, and flushes the buffer. However, the natural byte boundaries in the packet (as determined by the preamble) may not line up with the 1 byte buffer. This is because the bit slicer in the radio constantly sends bits to the microprocessor, regardless of whether the radio is receiving a Cricket packet or spurious signals. Most of the

[2]The Cricket packet structure is based on the standard TinyOS packet.

time, the packet byte boundaries will occur somewhere in the middle of the 8-bit buffer, so the microprocessor will actually have to wait for the buffer to be filled twice before it can decode a each byte. In steady state, of course, the microprocessor assembles one byte per interrupt.

This method of obtaining data in 8-bit chunks introduces problems for precise timing. Each listener starts its timer when it receives a RF packet of type "US Pulse." The listener determines the packet type once the header is decoded, but because of the variable position of byte boundaries, the listener may have to wait for one additional 8-bit buffer's worth of data before it can determine the packet type. Fortunately, since we know the radio data rate, as well as the relative bit offset between the buffer byte boundaries and the packet byte boundaries, we can back calculate a correction factor for each "US Pulse" packet. For example, Crickets transmit RF data at 19.2 Kbps. If the packet byte boundary is between the 7^{th} and 8^{th} bits of the buffer, then we know that the processor received the final interrupt of the packet 52.1 μs late. Table 2.1 summarizes the different bit offset possibilities and their corresponding compensation factors.

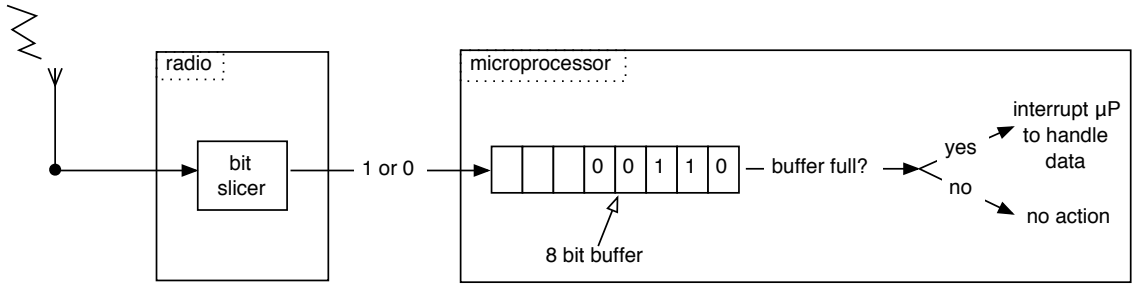

Figure 2-12: Diagram of the data bus between the radio and the microprocessor. When the radio is active, the bit slicer constantly sends data to the microprocessor via a serial data bus. As shown in the diagram, the microprocessor is only alerted to incoming data when the data has filled the microprocessor's 8 bit buffer.

Final header bit buffer contents (h = header bit, d = data bit)	Bit Offset	Compensation Factor (μs)
hhhhhhhh	0	0
hhhhhhhd	1	52.1
hhhhhhdd	2	104.2
hhhhhddd	3	156.3
hhhhdddd	4	208.3
hhhddddd	5	260.4
hhdddddd	6	312.5
hddddddd	7	364.6

Table 2.1: The different possibilities for the 8-bit buffer containing the final header bit, along with corresponding timer compensation factor. The compensation factor gives the number of microseconds that should be added to the timer to generate the correct ultrasound time-of-flight measurement. This table assumes a 19.2 Kbps RF data transmission rate.

Temperature Effects

Cricket's algorithms infer range using a parameter for the speed of sound. Unfortunately, both temperature and humidity affect the speed of sound in air. Of these two factors, temperature is much more important in determining the speed of sound. Table 2.2 summarizes the effect of temperature and humidity on the speed of sound. Even minor changes in room temperature, which may go unnoticed by occupants, can introduce a few centimeters of error into the Cricket distance measurements.

While the current V2 Crickets do not have onboard temperature sensors, we plan to add an accurate temperature sensor in the next minor revision of the hardware. This should allow listeners to compensate for changing temperatures, which will be useful in larger deployments, where the temperature will certainly not be constant between rooms.

Temperature (°C)	Relative humidity (%)	Speed of sound (m/s)
10	0	337.5
10	50	337.8
10	100	338.1
15	0	340.4
15	50	340.9
15	100	341.34
19	0	342.8
19	50	343.4
19	100	344.0
20	0	343.4
20	50	344.0
20	100	344.6
25	0	346.3
25	50	347.1
25	100	348.0

Table 2.2: Speed of sound at various air temperatures and humidity levels [2].

Even without a temperature sensor, the other modifications described in this section have lead to very accurate distance measurements. Figure 2-13 shows a CDF of distance measurements taken at 1.5 meters and 3 meters using V2 Crickets.

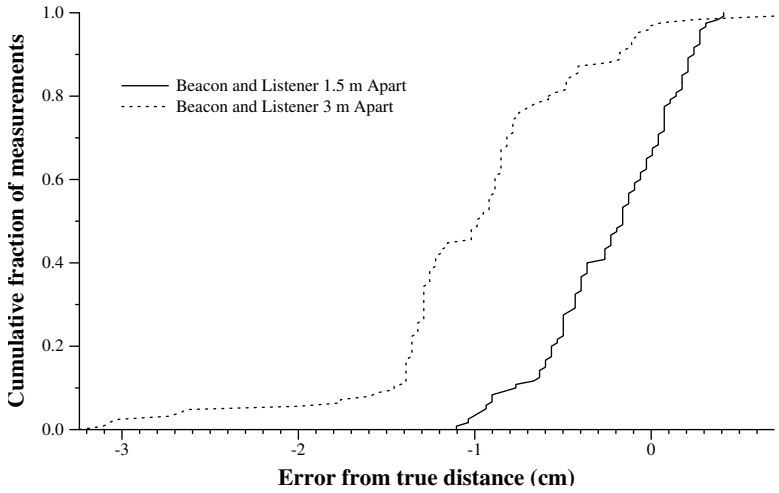

Figure 2-13: CDF of distance measurements between a Version 2 listener and beacon at 1.5 meters and 3 meters.

2.2 Beacon Configuration Methods

Accurate distance measurements only provide half the Cricket system infrastructure. For effective location-aware computing or navigation applications, users must be able to determine their 3D location in a meaningful context. This may mean users need to determine their 3D building coordinate that maps to a CAD file of their building, or it may mean that users need to determine their 3D building coordinate that maps to their internal representation of the building. In both of these cases, a consistent coordinate system is necessary for all the beacons in a building.

For scalability, it is not feasible to manually program each beacon with its 3D coordinate. Instead, we examined a variety of methods for assigning 3D coordinates to beacons.

2.2.1 Listener Assisted Configuration

Ph.D candidate Allen Miu was the first to attempt beacon auto-configuration. His "Listener Assisted" method used three assumptions; first, all beacons are coplanar (beacons are generally located on ceilings). Second, it is possible for a listener to get accurate distance measurements to all the beacons from each of three points in the room. Finally, each of these three points in the room must lie directly beneath a beacon. By exploiting the fact that the beacons are coplanar, distance measurements to all the beacons from these three points are adequate for determining a consistent coordinate system for all the beacons.

The Listener Assisted method is useful because it doesn't require any additional hardware for beacon configuration. Additionally, if the beacons are coplanar, it makes trilateration by the listener much easier after the configuration stage is over [4]. However, there are many drawbacks of this approach that make it unsuitable for large scale deployments. In larger rooms, or in deployments over multiple rooms, it is unlikely that there exists three points directly below beacons where a listener can range to all the beacons in the network. Additionally, this method alone does not provide any context for the beacon coordinate system. A user would have to integrate the coordinate system into a building CAD file or schematic to map the coordinate system to real locations. Lastly, while beacons are usually coplanar, a method that didn't include this requirement would be more robust in diverse deployments.

While the "Listener Assisted" method wasn't suitable for large scale beacon configuration, it did provide the inspiration for the final method used, as explained in Section 2.2.4.

2.2.2 Anchor-Free Localization

AFL, or Anchor-Free Localization, is a decentralized algorithm to determine a consistent coordinate system for nodes in a network, using only distances between neighboring nodes. AFL was developed by Ph.D candidate N. Bodhi Priyantha with the beacon configuration problem in mind.

The algorithm works by initially assigning a rough polar coordinate system to beacons by counting connectivity "hops" around the network. (A node is considered to be 1 hop away from its neighbor if it can measure the distance between itself and its neighbor.) Following the initial coordinate assignment, AFL works by incrementally decreasing the global energy, calculated using the sum of errors between estimated and measured distances between nodes [7]. Nodes must cooperate continuously to calculate the global energy.

AFL has the advantage of being completely decentralized, so it can work without any human intervention. Additionally, in tests AFL almost always converged on the correct graph. Unfortunately, due to Cricket hardware limitations, AFL cannot be applied if beacons are coplanar on the ceiling. The ultrasound transmitters and receivers are directional, which prevents lateral distance measurements between coplanar beacons if the beacons are facing the direction normal to the plane.

2.2.3 Robust Quadrilateral Configuration

Ph.D candidate David Moore's "Robust Quadrilateral" algorithm is also designed to determine consistent coordinates for nodes using only inter-node distances. The algorithm requires on a well connected network, but constructs the true embedding of the graph even with noisy inter-node distances. The key idea behind the algorithm is the identification of sets of 4 nodes which are all well connected to each other, in that they each have distance measurements to the other 3 nodes in the set. The coordinate system for the entire network is generated by successively combining these robust quadrilaterals [5]. Unlike AFL, each node generates coordinate systems independently, though all the coordinate systems will be identical except for a translation or flip.

The Robust Quadrilateral algorithm is fast, and can solve for a consistent coordinate system in much less time than AFL. However, since it also depends on inter-beacon distances, it has the same limitations as AFL, and will not work with coplanar beacons when all the beacons are facing the same direction.

While the Robust Quadrilateral algorithm was not used in beacon auto-configuration, it is quite useful for visualizing two dimensional Cricket beacon networks.

2.2.4 Frame Assisted Configuration

Creating a Coordinate System

The final algorithm for beacon configuration grew out of the "Listener Assisted" configuration method. The key idea to this "Frame Assisted" algorithm is that three listeners on a rigid frame can define their own 3D coordinate system. Figure 2-14 shows such a frame with three listeners, where two of these listeners, L_x and L_y, are located a distance r away from the third listener, L_o.

We can then assign the following coordinates to the listeners:

$$\begin{aligned} L_o &= (0,0,0) \\ L_x &= (r,0,0) \\ L_y &= (0,r,0) \end{aligned}$$

When the frame is placed in the vicinity of a beacon, as shown in Figure 2-15, the

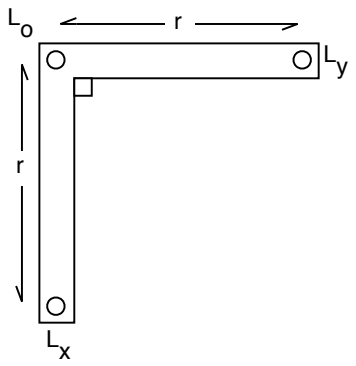

Figure 2-14: A rigid frame with three listeners defines a 3D coordinate system.

listeners measure their respective distances to the beacon.

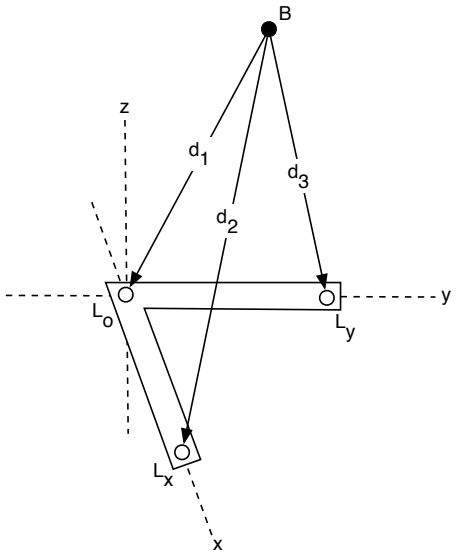

Figure 2-15: The frame can be used to assign coordinates to any beacon within range.

Using the distance measurements d_1, d_2, and d_3, we can set up the following equations, where (x, y, z) represents the unknown coordinates of the beacon:

$$d_1^2 = x^2 + y^2 + z^2 \tag{2.1}$$

$$d_2^2 = (x - r)^2 + y^2 + z^2 \tag{2.2}$$

$$d_3^2 = x^2 + (y - r)^2 + z^2 \tag{2.3}$$

We can then solve for the unknown coordinates (x, y, z):

$$x = \frac{1}{2r}(-d_2^2 + r^2 + d_1^2) \tag{2.4}$$

$$y = \frac{1}{2r}(-d_3^2 + r^2 + d_1^2) \tag{2.5}$$

$$z = \pm\frac{1}{2r}\sqrt{(-d_2^4 + 2d_2^2r^2 + 2d_2^2d_1^2 - 2r^4 - 2d_1^4 - d_3^4 + 2d_3^2r^2 + 2d_3^2d_1^2)} \tag{2.6}$$

Since the listeners use directional ultrasound receivers, we know the beacons must lie above the plane of the frame. Therefore, we take only the positive answer for z in Equation 2.6.

By measuring the distances to all the beacons within range, we can determine beacon coordinates that are consistent with the newly created coordinate system. However, a problem arises if all the beacons in a space are not within range of the frame in its initial position. The out-of-range beacons are excluded from the coordinate assignment, which is similar to what can happen when using the "Listener Assisted" configuration method. This problem may happen in a large room, and will definitely happen when deploying Cricket in multiple rooms.

The solution lies in finding the relationship between two different coordinate systems. Assume that the frame is used to assign coordinates in a room with 7 beacons: $B_1, B_2, \ldots B_7$. On the first pass the frame is within range of only five beacons $(B_1, B_2, \ldots B_5)$, and assigns coordinates to those beacons in the coordinate system C, which is our rest frame. The frame is then moved to a new location in the room, that is within range of beacons $(B_3, B_4, \ldots B_7)$. Using the new position of the frame, we assign coordinates to $B_3, B_4, \ldots B_7$ in the new coordinate system C'. Berthold Horn [3] showed that three points known in two coordinate systems are sufficient to determine the transformation between the two systems. Since the points represented by beacons B_3, B_4, and B_5 are common between the coordinate systems C and C', we can determine the relationship between the two systems. It is then easy to convert the coordinates of B_6 and B_7 in C' to the corresponding coordinates in the original system C.

If necessary, we can add beacons one at a time to the original coordinate system, provided we satisfy the constraint of the frame being within range of three beacons in that original coordinate system.

Mapping the Environment

Once all the beacons in a network have consistent coordinates, a listener can determine its coordinates in that system. However, for real-world applications we need to know how the coordinate system relates to objects in the real world. Aside from finding the CAD drawing for room of interest (and orienting the beacon network on the map), there are alternate methods for generating a map of the beacon environment.

One option is to walk around with a listener, and mark interesting objects in a database. For example, a user could mark out each wall vertex, along with windows, doorways, desks, etc. Later, or immediately, this database could be used to generate a map of the environment. This "virtual tagging" would be quite useful for applications, but would be tedious to set up initially.

A more attractive option is to implement virtual tagging using the frame and a laser range finder. Since the frame has three listeners attached, the orientation of the frame, in the rest frame, is known. Then, if a laser range finder is attached to the frame, the 3D coordinate of any spot or object highlighted by the laser can be calculated. The range finder simplifies the steps necessary to mark out walls, doors, or other structural elements.

To determine the frame's orientation in space, we need to determine the coordinates of each listener. Past versions of Cricket software have performed trilateration using the least squares error minimization method. This calculation is straightforward provided that one assumption is made; all the beacons are coplanar [4]. Since beacons may not always be coplanar, this is one assumption we would rather not make.

Alternatively, we can use Horn's algorithm to determine the frame's orientation in space. Provided that the frame is within range of three beacons, we can assign temporary coordinates to these three beacons. Then, by comparing our new coordinate system with the rest frame, we can determine the best transformation that maps the three new beacon coordinates to their rest frame coordinates. Using this transformation, we can determine the rest frame coordinates of the frame's listeners (which are located at $(0, 0, 0)$, $(r, 0, 0)$, and $(0, r, 0)$ in the new coordinate system).

After considering all four configuration methods, I chose to go forward with the Frame Assisted method.

Chapter 3

Implementation and Performance

In this chapter I describe my implementation of the Frame-Assisted Configuration of beacons, as described in Chapter 2. The goal of this configuration method was to allow an end user to walk into a room, install beacons, and generate a map of the beacons, along with the room's walls, windows, and doors. We require the 3D coordinates for the beacons, but it's sufficient to have the 2D coordinates for room features, so we can generate a CAD-like map of the room. A sample usage scenario follows:

- Phase I
  - Place the frame on the ground, and launch the configuration application on a connected PC.
  - Place beacons on ceiling, pointing down.
  - The application will create a "rest frame" coordinate system by assigning coordinates to the beacons, using their distance measurements to the three listeners on the frame. The three listeners are in a known configuration on the frame, so their rest frame coordinates are known, and define the coordinate system for the room.
  - Once the application displays the beacons and their assigned coordinates, freeze the map to begin tracking the frame.
- Phase II
  - Pick up the frame, and use the range finder to highlight a room feature, such as a door, window, or wall vertex. The software keeps track of features using pairs

of points, and draws lines connecting each pair of points. To mark a door, point the range finder at the base of the doorway. To mark a window, point the range finder at the upper corner of the window. To mark a wall, point the range finder at the corner of the wall at waist height. The software uses the height of the feature to categorize it correctly.

- When you confirm that the application has localized the frame correctly, push the trigger button on the range finder to mark the first point of the feature.
- Highlight the second point of the feature, and push the trigger button again. The software will connect the pair of points, color-coding the line as a door, wall, or window.

3.1 Building the Frame

The first task required was the creation of the frame. Using a milling machine and a band saw, I constructed the frame shown in Figure 3-1. The listeners were placed in specially cut holes such that the listeners at X and Y were 50 cm away from the listener at the origin.

We can then determine a beacon's coordinates (x, y, z in centimeters) using only relative distance measurements (in centimeters) to the listeners:

$$x = \frac{-1}{100}d_2^2 + 25 + \frac{1}{100}d_1^2 \quad (3.1)$$

$$y = \frac{-1}{100}d_3^2 + 25 + \frac{1}{100}d_1^2 \quad (3.2)$$

$$z = \frac{1}{100}\sqrt{(-12500000 - d_2^4 + 5000d_2^2 + 2d_2^2d_1^2 + 2d_3^2d_1^2 - 2d_1^4 - d_3^4 + 5000d_3^2)} \quad (3.3)$$

where d_1 is the distance from the beacon to the listener (L_o) at the origin, d_2 is the distance from the beacon to the listener (L_x) at $(50, 0, 0)$, and d_3 is the distance from the beacon to the listener (L_y) at $(0, 50, 0)$.

3.2 Acquiring the Data

The Crickets were all programmed with CricketNodeShare, a TinyOS application written by Ph.D candidate David Moore. I made minor modifications to the data output to simplify data parsing. CricketNodeShare enables one Cricket to collate and report the distance

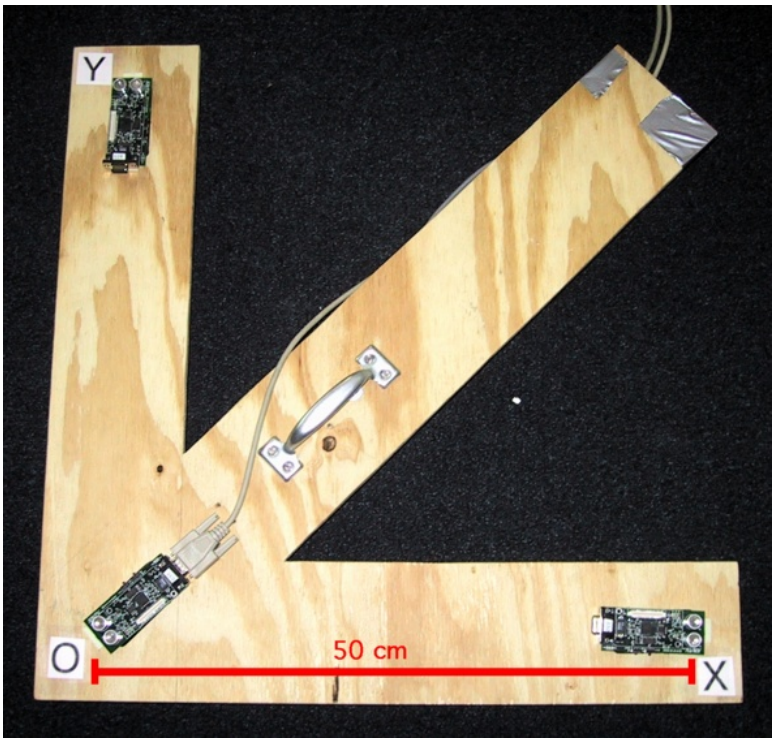

Figure 3-1: This is the frame I made for beacon configuration. In this frame $r = 50$ cm, where r is the distance between listeners, as shown in Figure 2-14.

measurements received by all its immediate neighbors. In this manner, I used the Cricket at the frame's origin to report all relevant distance measurements.

I chose to write the configuration application in Java for portability. Crickets transmit data using a standard RS-232 serial interface, but Java's serial support is poor, so I chose to retransmit the serial data over TCP/IP to the configuration application. By using *rwser*[1] to read the data from the serial port, and *netcat*[2] to retransmit the data, the configuration application simply opens a network port and reads the data line by line. Using the network instead of a direct serial connection also gives users the option of running the configuration application on a different machine than the one connected to the frame.

After the initial assignment of beacon coordinates, distances from the laser-range finder to points in the room are used to determine the coordinates for features in the room. The range finder, a Leica Disto Memo, transmits data using the same serial interface as the Crickets, so range measurements are forwarded to the configuration application through

[1] Written by David Moore
[2] Available from http://netcat.sourceforge.net/

the same rwser/netcat combination used for acquiring distance measurements.

3.3 Phase I

3.3.1 Determining Beacon Coordinates

To provide unobstructed ultrasound coverage of the room, beacons are generally placed on the ceiling, as seen in Figure 3-2. While the beacons in this configuration are coplanar, the system should also handle non-coplanar beacons. This is useful in rooms with tiered ceilings, and it provides flexibility as users may want to install beacons on walls as well as ceilings.

Figure 3-2: Beacons are generally placed on the ceiling to provide coverage throughout the room. In this photograph, the Cricket beacons are circled in green.

All the data collection, processing, and visualization was executed in a Java application, *WallsDemo3*. WD3 was designed to handle data from 12 beacons simultaneously, keeping for each beacon the most recent distance measurement to each of the three listeners on the frame. Every time a listener receives a new distance measurement from a beacon, WD3 calculates new coordinates for the beacon, as outlined in Sections 2.2.4 and 3.1. By keeping the last five calculated coordinates for each beacon, WD3 is able to display a trail of the estimated beacon position versus time. As shown in Figure 3-3, the main display in WD3 shows an overhead view of the room, with the frame, beacons, and any marked features.

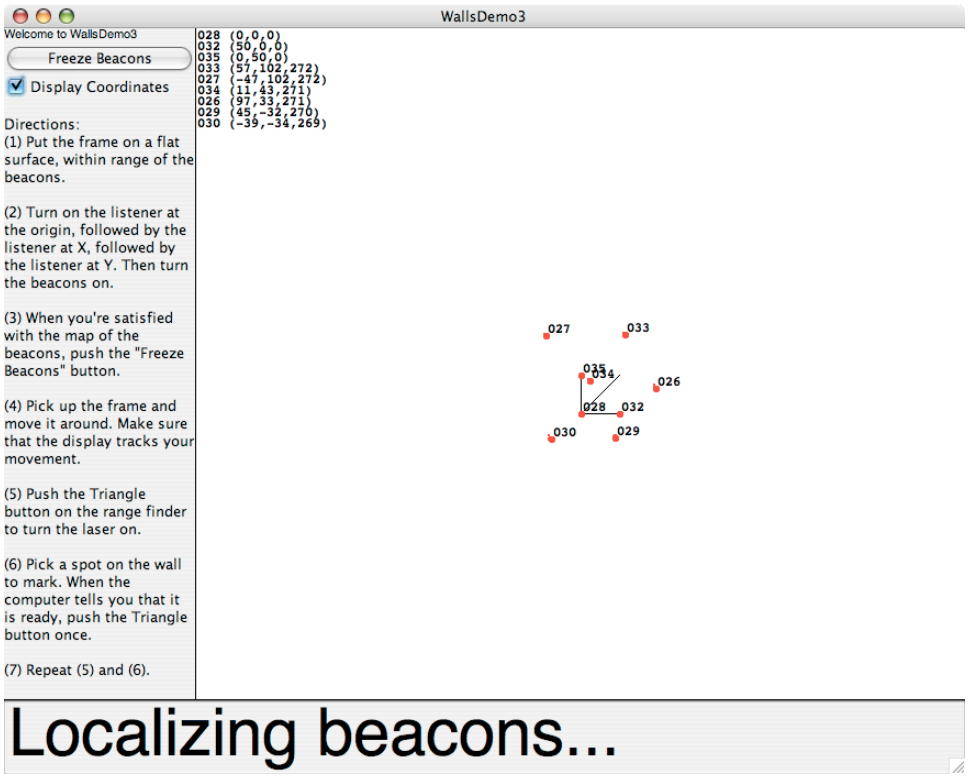

Figure 3-3: A screenshot of WallsDemo3, showing it calculating the beacon coordinates. The frame is shown in the center of the screen, and each beacon shows a trail of its last five calculated positions.

The user can tell when the beacons have been localized correctly because the trails disappear; the past five calculated coordinates overlap. Normally, this takes no more than a few seconds. When the user is satisfied with the beacon map, s/he pushes the "Freeze Beacons" button to stop updating the beacon coordinates on the screen. The beacon coordinates assigned during Phase I are frozen, and these coordinates make up the "rest frame" of the system.

3.4 Phase II

In Phase II of configuration, WD3 uses the rest frame coordinates of the beacons to track the frame as the user moves it. The user then triggers the range finder to mark features in the room; WD3 draws lines on the screen to connect pairs of feature points. These lines are color-coded to represent different types of features, including doors, walls, and windows.

3.4.1 Tracking the Frame

Once the user is satisfied with the calculated beacon coordinates, WD3 begins tracking the frame. As described in Section 2.2.4, if the beacons are known to be coplanar, a linear least-squares algorithm can be used to determine the position of listeners [4]. To avoid constraining the beacons to be coplanar, I chose to use Horn's solution for finding absolute orientation using unit quaternions [3].

Horn's algorithm determines the best transformation to align two different coordinate systems, using coordinates for at least three points as measured in each coordinate system. I consider the rest frame coordinate system to be the reference coordinate system. During Phase II, WD3 continues to track distance measurements from beacons to the listeners, and continually calculates new coordinates for the beacons in the new system. Every time the frame is moved, it sets up a new coordinate system, and relative to that system, every beacon will have different coordinates. Figure 3-4 shows how the same point has two different coordinates when the frame is moved.

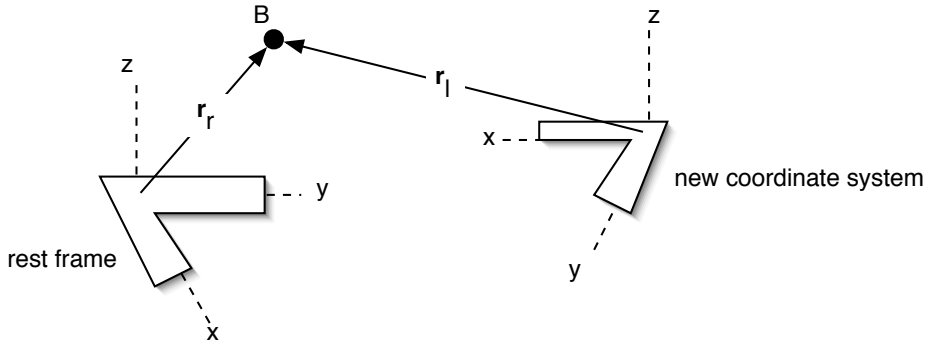

Figure 3-4: When the frame is moved, it creates a new coordinate system. The vector to beacon B in one coordinate system, $\mathbf{r}_r$, is not equal to the vector to B in the other coordinate system, $\mathbf{r}_l$.

Horn's solution for absolute orientation is used to determine the best-fit transformation between the two coordinate systems. While Horn's generalized algorithm operates on any number of points, the computation required and complexity are greatly reduced when only three points are chosen for transformation. In this application, choosing only three points has the added benefit of preventing unreliable or inaccurate distance measurements from adversely affecting the calculation of the necessary transformation. WD3 chooses the three

closest beacons to the origin of the frame, and applies Horn's algorithm to them. The chosen beacons are highlighted in pink on the display, as shown in Figure 3-5. The three closest beacons will also have, on average, the most accurate distance measurements. This is because the measurement error increases with the distance between a beacon and listener.

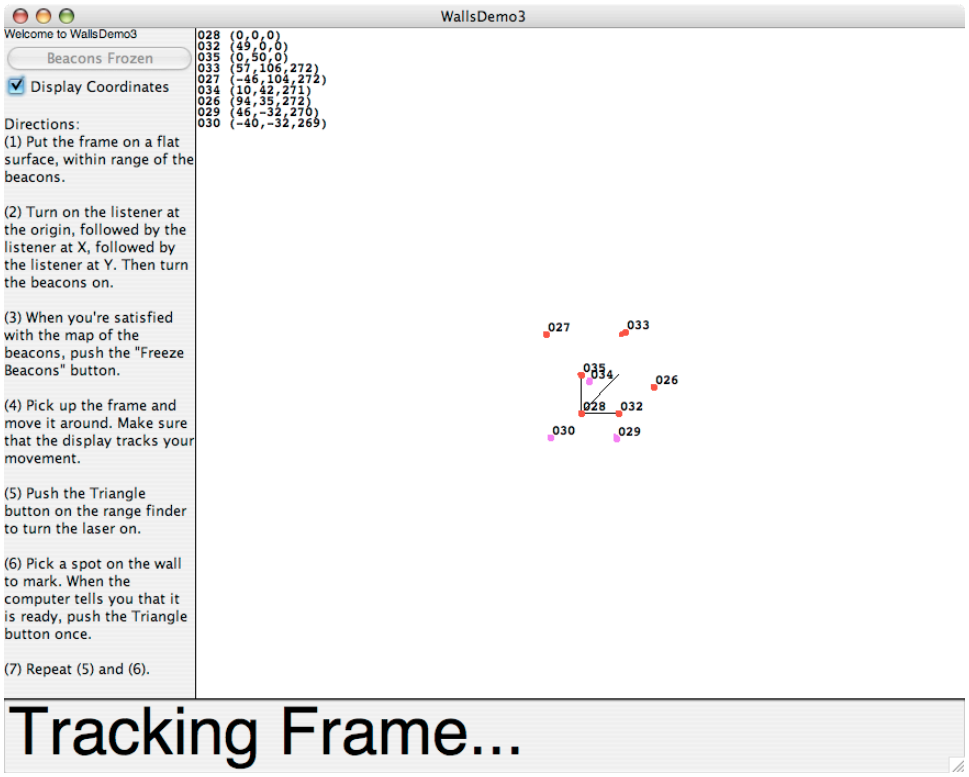

Figure 3-5: A screenshot of WallsDemo3, showing it tracking the frame's position. The three beacons used in Horn's transformation algorithm are highlighted in pink, while the rest of the Crickets are shown in red.

Since distance measurements are constantly acquired, WD3 runs Horn's algorithm many times every second, and continually updates the display with the most recent estimate of the frame position. Once the user stops moving the frame, the frame on the screen stops moving, since new position estimates return the same answer as previous estimates.

Transformation of Coordinates

I have summarized below Horn's algorithm for determining the best transformation from three left-handed points $\{\mathbf{r}_{l,1}, \mathbf{r}_{l,2}, \mathbf{r}_{l,3}\}$, to three right-handed points $\{\mathbf{r}_{r,1}, \mathbf{r}_{r,2}, \mathbf{r}_{r,3}\}$.[3] The

[3] I am actually determining the best transformation between two right-handed coordinate systems, but using the notation for left-handed and right-handed makes the math easier to follow.

best transformation is determined in three steps:

1. Determine the translation necessary to match the centroids of the points.

2. Determine the rotation necessary to cause the two planes defined by each set of three points to coincide.

3. Determine the best rotation in the plane that matches up each of the three points with its mate.

The best transformation takes the form: $\mathbf{r}_r = sR(\mathbf{r}_l) + \mathbf{r}_0$, where s is the scale factor, R is the rotation, and $\mathbf{r}_0$ is the translation. For my purposes, I forced $s = 1.0$, since I knew a priori that both coordinate systems used centimeters as units.

The centroids for $\mathbf{r}_{l,i}$ and $\mathbf{r}_{r,i}$ are defined as $\bar{\mathbf{r}}_l = \frac{1}{3}\sum_{i=1}^{3}\mathbf{r}_{l,i}$, and $\bar{\mathbf{r}}_r = \frac{1}{3}\sum_{i=1}^{3}\mathbf{r}_{r,i}$, respectively. We can then define new coordinates, $\mathbf{r}'_{l,i} = \mathbf{r}_{l,i} - \bar{\mathbf{r}}_{l,i}$ and $\mathbf{r}'_{r,i} = \mathbf{r}_{r,i} - \bar{\mathbf{r}}_{r,i}$. The translation from $\bar{\mathbf{r}}_l$ to $\bar{\mathbf{r}}_r$ is the difference between the right centroid and the scaled and rotated left centroid: $\mathbf{r}_0 = \bar{\mathbf{r}}_r - sR(\bar{\mathbf{r}}_l)$.

Our next task is to align the planes defined by each set of three points. We can represent each plane by its normal vector, $\mathbf{n}$, defined as the cross product of any two nonparallel vectors in the plane. Thus, $\mathbf{n}_l = \mathbf{r}'_{l,1} \times \mathbf{r}'_{l,2}$, and $\mathbf{n}_r = \mathbf{r}'_{r,1} \times \mathbf{r}'_{r,2}$. As shown in Figure 3-6, ϕ is the angle of rotation necessary to align the planes. Therefore, $\cos\phi = \hat{n}_l \cdot \hat{n}_r$, and $\sin\phi = ||\hat{n}_l \times \hat{n}_r||$. We then define $\mathbf{r}''_{l,i}$ to be the rotated version of $\mathbf{r}'_{l,i}$.

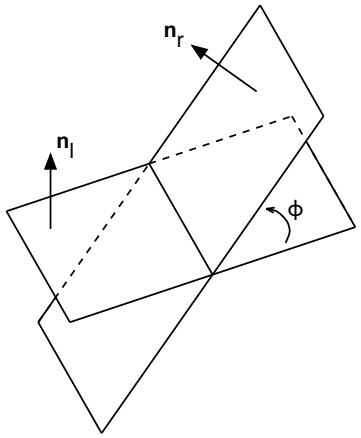

Figure 3-6: The plane defined by $\mathbf{n}_l$ must be rotated through ϕ radians until it aligns with the plane defined by $\mathbf{n}_r$.

Finally, we have to rotate the left hand plane, as shown in Figure 3-7 to minimize the sum of the squared error between measurements: $\sum_{i=1}^{3} \left|\left|\mathbf{r}'_{r,i} - \mathbf{r}'_{l,i}\right|\right|^2$.

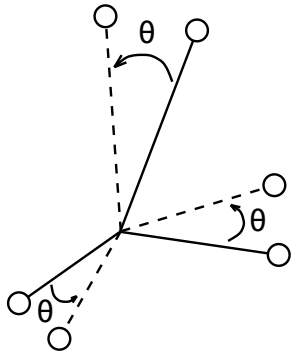

Figure 3-7: Once the two planes are aligned, points in the left hand plane must be rotated by θ radians to minimize the sum of the squared error between pairs of measurements. In a transformation with no measurement error, the points will align perfectly.

Horn [3] then shows that minimizing the sum of the squared error is equivalent to maximizing $C\cos\theta + S\sin\theta$, where $C = \sum_{i=1}^{3}(\mathbf{r}'_{r,i} \cdot \mathbf{r}''_{l,i})$, and $S = \left(\sum_{i=1}^{3} \mathbf{r}'_{r,i} \times \mathbf{r}''_{l,i}\right) \cdot \hat{n}_r$. The maximum of that expression occurs when $\sin\theta = \frac{S}{\sqrt{S^2+C^2}}$, and $\cos\theta = \frac{C}{\sqrt{S^2+C^2}}$.

The total rotation, R, is the composite of the rotation of the planes and the rotation of points within the plane. I used quaternions to represent rotation, so the total rotation was simply the product of the two separate rotation quaternions.

Once we have determined the best transformation, $\mathbf{r}_r = sR(\mathbf{r}_l) + \mathbf{r}_0$, it is easy to determine the coordinates of the frame's listeners in rest frame coordinates. We simply insert the three listener coordinates in for $\mathbf{r}_l$ ($(0,0,0)$, $(50,0,0)$, and $(0,50,0)$), and the corresponding $\mathbf{r}_r$ values are the coordinates of the listeners in the rest frame.

3.4.2 Marking Room Features

Once the software determines the rest frame coordinates of the frame, it can determine the coordinates of any spot the user marks with the range finder. The range finder is mounted below the frame in a fixed orientation, as shown in Figure 3-8. Figure 3-9 shows the frame being used to mark a wall vertex.

WD3 maps walls, windows, and doors by drawing line segments between pairs of user-chosen features. As stated earlier, the user categorizes each feature at the time it is being

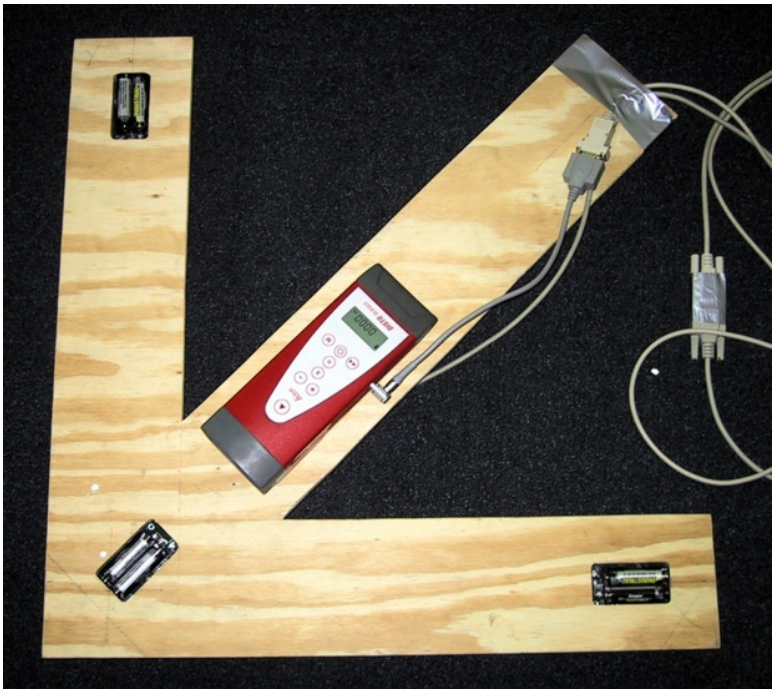

Figure 3-8: The range finder is mounted on the underside of the frame, and measures the distance from the frame to any spot highlighted by the user.

marked. Each feature can either be a door, wall, or window. The height of the marked feature determines its type; e.g. any feature marked at waist height will be considered a wall. In a similar fashion, any feature near the floor is considered a door, and any feature at eye level is considered a window. I chose to draw doors in green, walls in black, and windows in blue.

Since WD3 connects pairs of features, the software tells the user when it is ready to mark the first or second point in each pair. Other than marking features in pairs, points may be chosen in any order. Figure 3-10 shows two screenshots of the WD3 application as the user marks a wall.

Figure 3-9: A user marks a wall vertex using the frame and range finder. The range finder projects a red spot on the wall (shown here circled in green), which indicates to the user which feature s/he is about to mark.

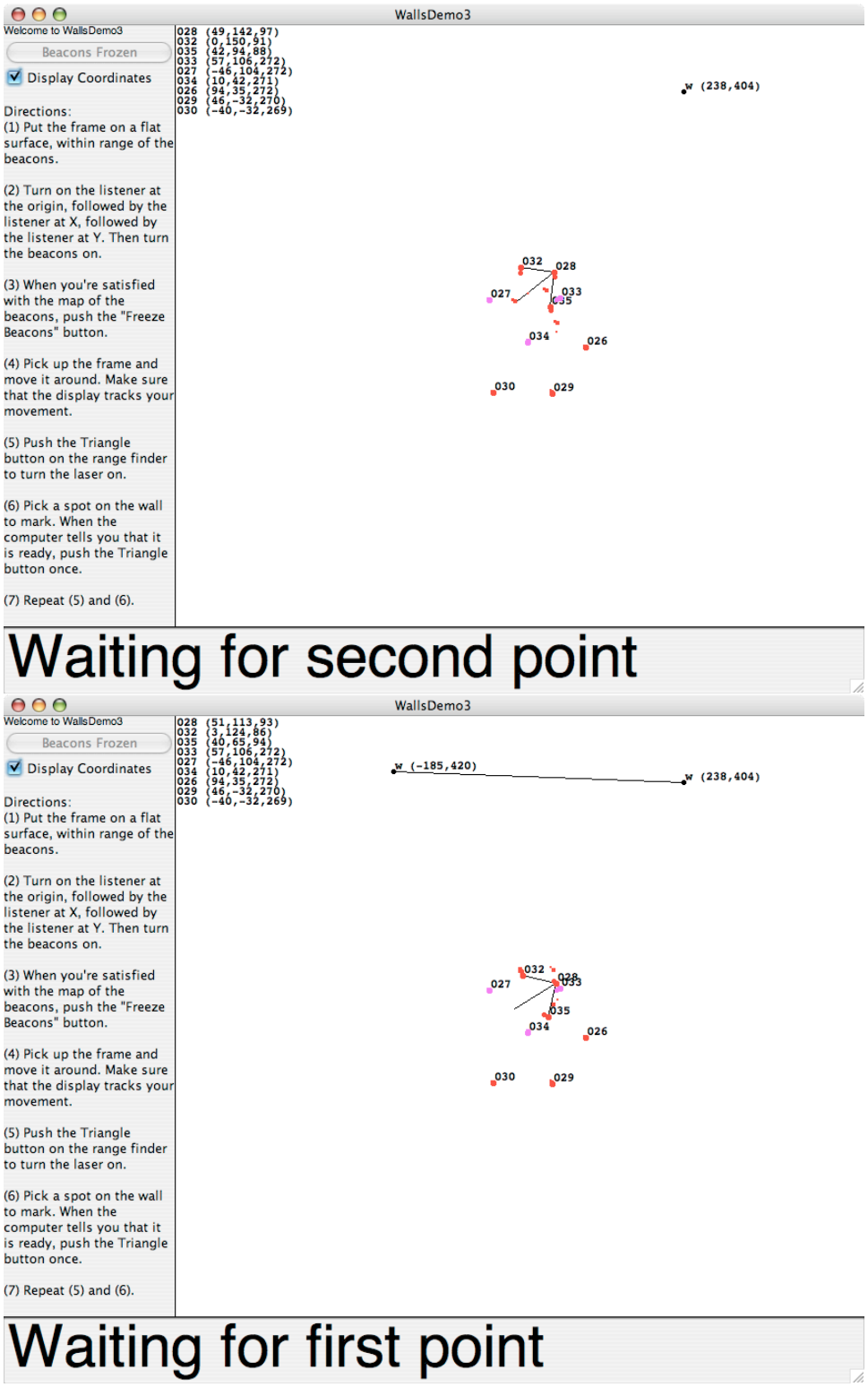

Figure 3-10: Two screenshots of WallsDemo3, showing a user marking out walls on the same map with beacon positions. (Above) The user has marked one wall vertex, and the software is waiting for the second vertex of the pair. (Below) The user has marked the second wall vertex, and the software has drawn the line of the corresponding wall.

3.5 Performance

For comparison, I also implemented a system to track the frame using direct three-point trilateration. Surprisingly, this gave better results than the least-squares solution using more than three points, and had the added benefit of not constraining the beacons to be coplanar. I believe the three-point trilateration offered better results because I chose the closest three points, which again have the most accurate measurements. I mapped 32-D451 twice, once using three-point trilateration for frame tracking, and once using Horn's coordinate transformation algorithm. Figure 3-11 shows the results when using trilateration to track the frame, and Figure 3-12 shows the results when using Horn's algorithm to track the frame. Unsurprisingly, Horn's algorithm performs better, as it includes the extra constraint that the three listeners must be localized in a rigid orientation.

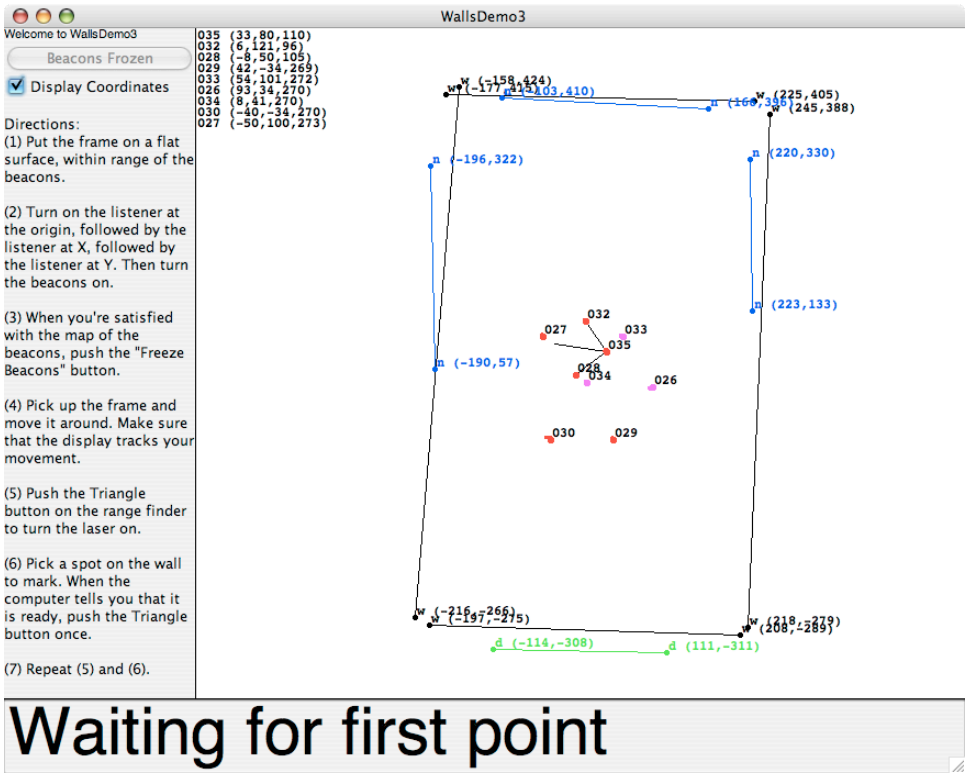

Figure 3-11: Screenshot of WallsDemo3 after creating a map of 32-D451. In this example, the frame was tracked using three beacon trilateration.

To verify that our generated map is accurate, we overlay our WD3 map on a supplied CAD drawing of the same room. Figure 3-13 shows this overlay, and demonstrates that this configuration system is quite accurate – down to the column in the rightmost wall.

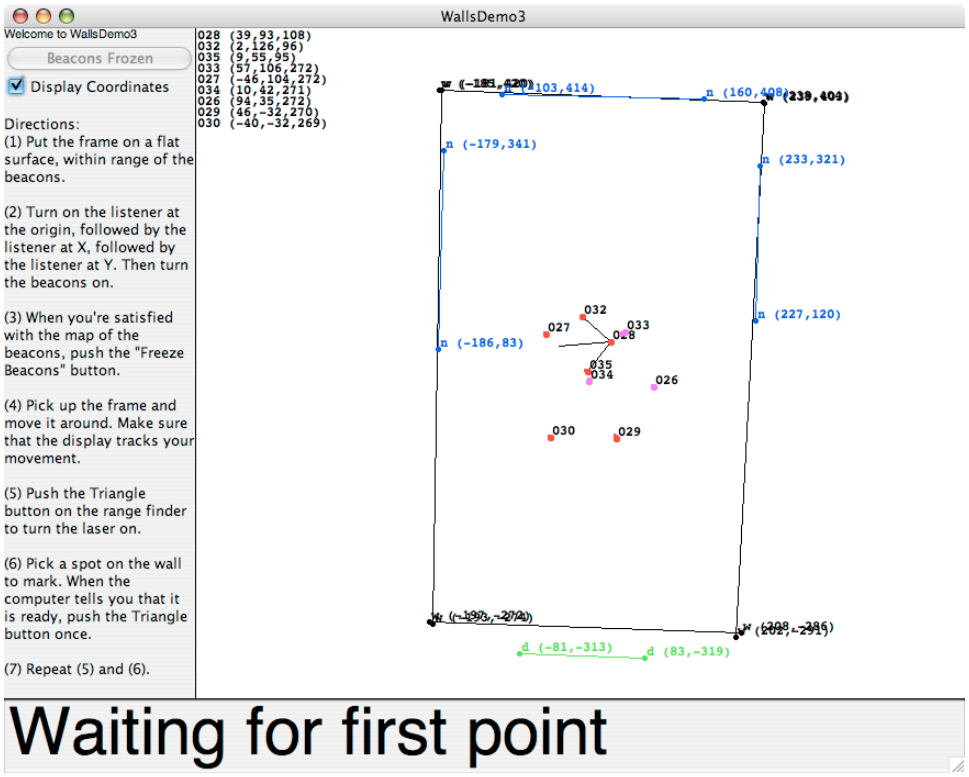

Figure 3-12: Screenshot of WallsDemo3 after creating a map of 32-D451. In this example, the frame was tracked using Horn's coordinate transformation algorithm. The improvement in accuracy is primarily because Horn's algorithm constrains the three frame listeners to be localized in a rigid orientation.

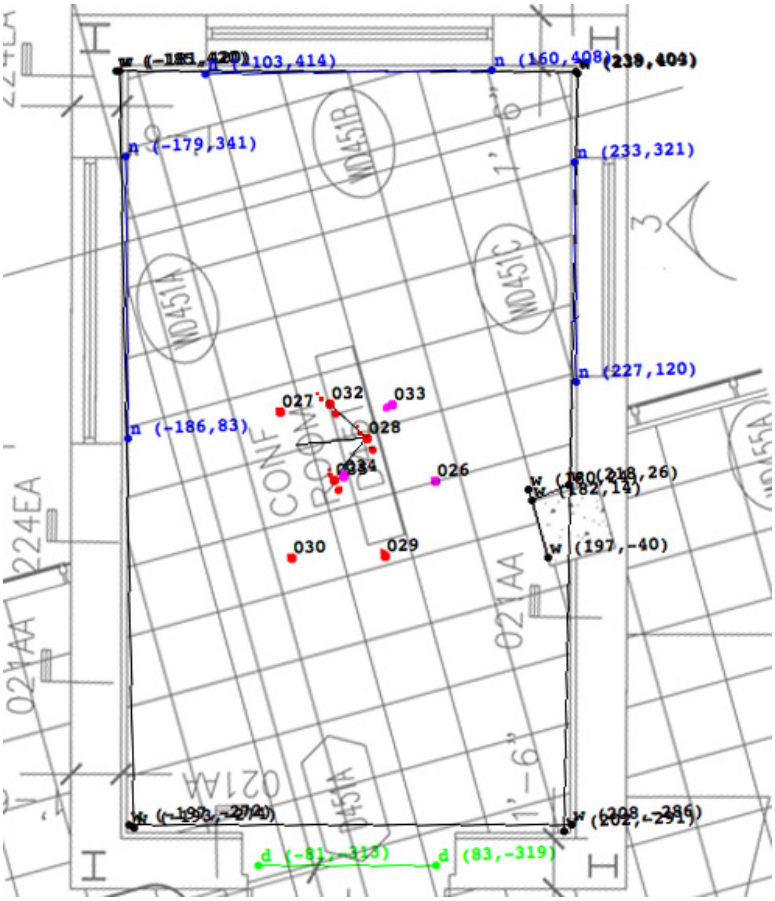

Figure 3-13: Manual overlay of the WD3-generated map on the architect's CAD drawing for 32-D451.

Chapter 4

Contributions and Future Work

In this thesis I describe how to go into a new room, and return a short time later with a map of the room, complete with coordinates for features such as doors, windows, walls, and Cricket beacons in the room.

This procedure involves using three listeners on a frame to set up a local coordinate system in the room, which is then used to assign consistent coordinates to Cricket beacons in the room. After the rest frame has been created, the user marks features in the room, using the frame and a laser range finder. By using different gestures, the user can mark doors, windows, and walls, which each show up in different colors on a display. The result is a map of the room, with 2D coordinates for walls, doors, and windows, and 3D coordinates for the beacons. The coordinate system is given in centimeters, so a real-world scale for the room is known.

By testing this system in a real room, I generated a map that compares favorably with the CAD drawing supplied by the building's architects.

Future work on this project includes:

- Have the software display a true 3D view of the room, since 3D coordinates for beacons and marked points are known.
- Use the range finder to return constant range information, so there is real-time feedback for the user on the display on what they are highlighting.
- Add an additional gesture for columns, or other vertical edges in the building.
- Draw the variance of distance measurements around each beacon on the display, to

provide simple feedback of the accuracy of recent measurements.

- Make the software tolerant of a restart halfway through the setup. Currently, any user error or hardware failure during configuration requires a full restart of the entire system.

- Offer the user a grid display overlaid on the screen, so the user can see the real scale on the display.

- Add functionality to combine different beacon networks together into one consistent coordinate system. Currently, only beacons within range of the frame are assigned rest frame coordinates. It should be possible to assign rest frame coordinates to beacons in different spaces or rooms.

- The software should extrapolate more information from features. For example, the software could connect all the walls together by drawing multiple planes, which would better resemble real rooms.

- Use the full version of Horn's algorithm, so the best transformation is calculated using information from all available beacons, not just the closest three. This may not work as well, as beacons further away will have more error in their distance measurements, but it should be investigated.

Appendix A

Required Materials

Hardware

Frame The frame is a wooden arrow-shaped device with a metal handle. It has three milled slots for holding Crickets, along with a screw for attaching the laser range finder.

Crickets I tested this system using V2 Crickets with serial ports. The RS-232 Crickets manufactured by Crossbow work fine, though the Compact Flash versions have not been tested. Theoretically, only the Cricket at the origin of the frame needs to have a serial port; the rest can be any V2 Crickets.

Laser Range Finder I used the Leica Disto Memo range finder. If other Leica range finders output data the same format over their RS-232 interfaces, they can also be used. The Disto Memo includes a physical adapter to break out its proprietary serial interface to the standard DE-9 D-sub 9-pin port.

Computer At a minimum, this system requires a GNU/Linux computer with two serial ports. Alternatively, the demonstration software can be run on any computer, and the necessary Cricket data and range finder data can be forwarded to it from one or two other GNU/Linux computers with serial ports.

Serial Cables You will need two serial cables. The Disto Memo range finder requires a null modem (e.g. crossover) serial cable. Crickets use standard serial cables.

Velcro Typically we use adhesive-backed velcro for attaching Crickets to the ceiling. Our standard at CSAIL is to use the "hard" side on the backs of Crickets, and the "soft" side on the ceiling.

Software

TinyOS TinyOS is an open-source embedded operating system developed at Berkeley. The Crickets run TinyOS; I used version 1.1.0. The source code and necessary libraries for TinyOS are available from http://webs.cs.berkeley.edu/tos/.

CricketNodeShare CNS is a TinyOS application that must be run by Crickets used by this system. David Moore wrote CNS, though one minor modification is necessary for it to be used with WallsDemo3. Table A.1 lists the CVS revision numbers of

CricketNodeShare files that I tested with WallsDemo3. CNS is available in the NMS Cricket CVS repository at Cricket/tos/apps/CricketNodeShare/. The modified version of BeacManageM.nc is available at Cricket/src/app/WallsDemo3/.

rwser David Moore wrote this small utility that writes serial data to standard out. Rwser is necessary to read and forward data from the Crickets to the visualization application. To read and forward the data from the laser range finder, you can either user rwser, or the Unix utilities stty and cat. The rwser source is available in the NMS Cricket CVS repository at Cricket/src/rwser/.

netcat Netcat is used to forward data from standard to the network, over TCP/IP. Netcat is free software, and is available from http://netcat.sourceforge.net/.

WallsDemo3 The WallsDemo3 software is written in Java, and has been tested on Mac OS X and GNU/Linux systems. Source code is available in the NMS Cricket CVS repository at Cricket/src/app/WallsDemo3/.

- Java 1.4.2 is required to run WallsDemo3. You will need both the Java compiler (javac) and the Java runtime environment (java).
- The JAMA Java linear algebra package is required for matrix math operations. JAMA is available from http://math.nist.gov/javanumerics/jama/. I tested this system using JAMA version 1.0.1.
- Java3D is required for vector math operations. The javax.vecmath library, a subset of Java3D, is sufficient for running WallsDemo3. I tested this system using Java3D version 1.3.1.

Filename	CVS Revision Number
BeacManage.nc	1.2
BeacManageM.nc	1.5 (with patch)
Cricket.h	1.6
Cricket.nc	1.4
CricketM.nc	1.8
Serial.nc	1.2
SerialM.nc	1.3

Table A.1: This table shows the CVS revision numbers for each file used by CricketNodeShare. These specific revisions are necessary when using CNS with WallsDemo3.

Appendix B

Demonstration Instructions

This appendix includes detailed instructions for obtaining the software, building the software, setting up the demonstration, and running the demonstration. I assume that you have a working installation of TinyOS, and that you can program Crickets using that TinyOS installation.

1. Read Appendix A to make sure you have all the required hardware and software. Appendix A also includes pointers to all the necessary software, including CVS paths.

2. Program at least six Crickets with CricketNodeShare. Prior to compiling CricketNodeShare, replace the file "BeacManageM.nc" in the CricketNodeShare directory with the version in the WallsDemo3 directory. You may want to verify that the CVS revision numbers of files in the CricketNodeShare directory match those given in Table A.1.

3. Install three Crickets onto the frame, ensuring the orientation of the Crickets match the photo on Page 33. The ultrasound receivers and transmitters on each Cricket should be closest to the paper labels on the frame.

4. Build rwser by executing the command:

   ```
   make all
   ```

   in the rwser directory.

5. Test serial communication with the Crickets. Connect the Cricket at the frame's origin to your computer's serial port, using a regular serial cable. Turn the Cricket on, and execute the command:

   ```
   ./rwser -i -d /dev/ttyS1
   ```

   in the rwser directory. You should replace the serial device name with the correct serial device for your computer. GNU/Linux usually begins numbering the serial ports from 0, so if the Cricket is attached to the first serial port, you should use the `-d /dev/ttyS0` flag. Likewise, if the Cricket is attached to the second serial port, you should use the `-d /dev/ttyS1` flag. To verify that the Cricket's serial port is operating correctly, type:

   ```
   DEBUG ON
   ```

into rwser. Your screen should begin printing debug information from the Cricket, including lines with distance measurements to other active Crickets in the vicinity. Type Ctrl-c to quit rwser.

6. Attach the range finder to the frame, as shown in the photo on Page 40. The range finder should point toward the frame's origin, and should be mounted on the underside of the frame.

7. Test serial communication with the range finder.[1] Connect the range finder to a free serial port on your computer, using a null modem serial cable. Execute the command:

   ```
   stty -F /dev/ttyS0 parenb -parodd cs7 9600 -cstopb crtscts
   ```

   to set your serial device to the correct settings: 9600 baud, even parity, 7 data bits, 1 stop bit, use hardware flow control. These are default settings for the range finder, though they can be changed manually. Be sure to replace `/dev/ttyS0` with the name of the serial port to which the range finder is attached. Now execute the command:

   ```
   cat /dev/ttyS0
   ```

   and turn on the range finder. (Use the same serial device name you used in the previous command.) Push the triangle button on the range finder once to turn on the laser, and one more time to take a range measurement. Your screen should output the measurement in the following format:

   ```
   31..00+00002437 51....+0000+000
   ```

   Type Ctrl-c to quit cat.

8. Change to the WallsDemo3 directory. Prior to compiling WallsDemo3, you should edit the following variables:

   (a) `PANELSIZE` in WallsDemo.java and WallsDisplay.java. This int determines the height (and width), in pixels, of the display on the screen. For a 1024x768 display, I usually set `PANELSIZE = 700`.

   (b) `SF` in WallsDisplay.java. This double determines the scale factor for drawing the coordinate system on the screen. If `SF = 1.0`, 1 pixel = 1 cm. If `SF = 0.5`, 0.5 pixels = 1 cm; 1 pixel = 2 cm. If `SF = 2`, 2 pixels = 1 cm; 1 pixel = 0.5 cm.

   (c) `MAX_NUM_BEACONS` in WallsDisplay.java. This int determines the maximum number of Crickets you can use, including the three on the frame.

   (d) `MAX_NUM_FEATURES` in WallsDisplay.java. This int determines the maximum number of features you can mark in WallsDemo3.

   (e) `USE_HORN` in WallsDisplay.java and BeaconData.java. This boolean determines which method will be used for tracking the frame. If `USE_HORN = true`, then Horn's three point coordinate transformation algorithm will be used to track the frame. Beacons may be non-coplanar. If `USE_HORN = false`, then three-point trilateration will be used to track the frame. Beacons must be coplanar.

[1] The file "Leica_Memo_Serial_Commands.pdf" in Cricket/src/Cricket3D/ includes a full description of the serial specifications for the Disto Memo range finder. However, it neglects to mention that the range finder uses hardware flow control.

(f) `LASER_IP` in WallsDisplay.java. This string determines the domain name (or IP address) of the computer that has the laser range finder attached.

(g) `CRICKET_IP` in WallsDisplay.java. This string determines the domain name (or IP address) of the computer that has the Cricket (at the frame's origin) attached.

(h) `FLOOR_OFFSET` in FeatureData.java. This int tells WallsDemo3 the height of the frame, in centimeters, during beacon localization. In general, when the frame is kept on the floor, `FLOOR_OFFSET = 0`. This variable exists in case the frame is kept on a table during beacon localization. In that case, `FLOOR_OFFSET` should be set to be the height of the table, in centimeters.

(i) `nodelabellist` in CricketClient.java. This hash table determines the mapping between Cricket hex IDs and their sticker labels. Each key in the hash table is a Cricket hex ID, and the value associated with each key is the label on that Cricket's sticker. The sticker label is displayed on the screen, so users can match the beacons on the screen with Crickets in real life. You should add your Crickets' hex IDs and sticker labels to this hash table.

(j) `OVERRIDE_SCALE` in rotSolver.java. This boolean determines if the scale for the best coordinate transformation will be calculated. If `OVERRIDE_SCALE = true` then the scale for the best transformation will always be 1.0. If `OVERRIDE_SCALE = false`, the scale will be calculated using Horn's three point algorithm. The default value for this variable is true, as that leads to the best results.

9. Compile WD3 by executing the command:

```
make all
```

in the WallsDemo3 directory. Be sure that you have Java3D and Jama installed!

10. Turn off all the Crickets. Start the serial/network connections to the range finder and Crickets by executing the following commands:

```
./rwser -i -d /dev/ttyS1 | nc -l -p2947
cat /dev/ttyS0 | nc -l -p2948
```

Be sure to use the same serial device names you used earlier. These commands open the network ports to which WD3 connects.

11. Put the frame on the floor, with the Crickets facing up.

12. Run WallsDemo3 by executing the command:

```
make run
```

WallsDemo3 should start up, and your console should print out two messages stating successful connections to the LaserClient (LC) and CricketClient (CC). If you get errors stating that the "Connection failed," check to make sure that you have the correct IP addresses or domain names listed in the file WallsDisplay.java.

13. Every time WallsDemo3 is run, it overwrites the file "featuredata.txt" in the WallsDemo3 directory. This file contains the 3D coordinates for all beacons and features marked during the most recent run of WallsDemo3. Each time you run WallsDemo3, you may want to copy this file to another directory to retain the data.

14. Turn the on the Cricket at the origin. Then turn on the Cricket at "X," followed by the Cricket at "Y." The Crickets on the frame *must* be turned on in this order, so WallsDemo3 knows which Crickets are on the frame.

15. Execute the command:

    ```
    DEBUG ON
    ```

    in the rwser window. This tells the Cricket at the origin to begin sending serial data.

16. Verify that display now shows labels for the three Crickets on the frame. If the sticker labels were not found in the hash table, WallsDemo3 will display "??" instead of a label.

17. When running WallsDemo3, you can select the "Display Coordinates" check box if you want to display the beacon and frame coordinates on the screen. Feature coordinates are always displayed.

18. Turn on one beacon, and make sure the display tracks the beacon as you move it slowly over the frame.

19. Attach the beacon to the ceiling, and repeat the previous step for subsequent beacons. You must use at least three beacons.

20. When you are done attaching all the beacons to the ceiling, verify that the display shows all the beacons in their correct locations. If the software has trouble localizing the beacons, you may have to move the beacons to bring them within range of the frame. Once you are satisfied that WallsDemo3 has correctly localized all the beacons, click the "Freeze Beacons" button.

21. Pick up the frame, and verify that the display tracks the frame's movement.

22. Push the triangle button on the laser range finder once to turn the laser on. Use the laser to highlight the first feature you would like to mark. WD3 marks features in pairs, connecting the two points with a line on the screen. WD3 tells you near the bottom of the screen whether it is waiting for the first or second point of each pair.

23. When you are ready to mark the feature, verify that the display has tracked the frame correctly. Push the triangle button once to mark the feature. The range finder will beep, and the laser will turn off. The 3D point should show up on the screen. If the marked point is the second in a pair of points, WD3 will draw the line connecting the two points.

24. When marking wall vertices, point the laser at waist level on the wall. When marking window vertices, point the laser as high as possible. When marking doorway vertices, point the laser at the ground. WD3 uses the height of the marked point to determine whether the feature is a door, window, or wall.

25. If the laser range finder turns off due to idle time, push the triangle button once to turn back on.

26. Continue marking new points as outlined in Steps 22 and 23.

27. When you are done with the map, you can quit the application. (Closing the main Java window quits the application.) The 3D coordinates for the beacons and marked features are kept in the file "featuredata.txt," in the WallsDemo3 directory.

28. Quit rwser and cat by typing Ctrl-c in each window.

Bibliography

[1] Paramvir Bahl and Venkata N. Padmanabhan. RADAR: An in-building RF-based user location and tracking system. In *Proc. IEEE INFOCOM*, Tel-Aviv, Israel, March 2000. IEEE.

[2] Owen Cramer. The variation of the specific heat ratio and the speed of sound in air with temperature, pressure, humidity, and CO_2 concentration. *Journal of the Acoustical Society of America*, 93(5):2510–2516, May 1993. [An online version of Cramer's formula is available at http://www.measure.demon.co.uk/Acoustics_Software/speed.html].

[3] Berthold K.P. Horn. Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America*, 4:629, April 1987.

[4] Allen Ka Lun Miu. Design and implementation of an indoor mobile navigation system. Master of Science, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, January 2002.

[5] David Moore, John Leonard, Daniela Rus, and Seth Teller. Robust distributed network localization with noisy range measurements. (preprint), May 2004.

[6] Bradford W. Parkinson and James J. Spiker Jr., editors. *Global Positioning System: Theory and Applications*, volume 1. American Institute of Aeronautics and Astronautics, Inc, 1996.

[7] Nissanka B. Priyantha, Hari Balakrishnan, Erik D. Demaine, and Seth Teller. Anchor-free distributed localization in sensor networks. LCS Tech Report 892, Massachusetts Institute of Technology, April 2003.

[8] Gilbert Strang and Kai Borre. *Linear Algebra, Geodesy, and GPS*. Wellesey-Cambridge Press, Wellesey, Massachusetts, 1997.

[9] Andy Ward, Alan Jones, and Andy Hopper. A new location technique for the active office. *IEEE Personal Communications*, 4(5):42–47, October 1997.

[10] M. Youssef, A. Agrawala, and U. Shankar. WLAN location determination via clustering and probability distributions. In *Proc. IEEE PerCom*, Fort Worth, TX, March 2003. IEEE.